User's Guide

VERSION 4.0

PowerBuilder

December 1994

Contents

About This Manual

Subject

This manual describes the PowerBuilder development environment. It shows you how to use the variety of tools that PowerBuilder provides to build client/server applications.

Audience

This manual is for anyone who will be building applications with PowerBuilder. It assumes that:

- You are familiar with the user interface guidelines for the computing platform you will be developing and deploying your applications on. If not, consult a book that covers the user-interface conventions.
- You have a basic familiarity with SQL. If not, consult a book that describes SQL statements.

PART ONE

The PowerBuilder Environment

This part describes the basics of using PowerBuilder and how to set up and maintain an application.

CHAPTER 1

The World of PowerBuilder

About this chapter

This chapter describes the basics of working in PowerBuilder.

Contents

Before you begin

If you are new to PowerBuilder, you should first do the tutorial in *Getting Started*. The tutorial guides you through the process of building a PowerBuilder application.

What is PowerBuilder?

PowerBuilder is a graphical client/server application development environment. Using PowerBuilder, you can easily develop powerful graphical applications that access your server databases. PowerBuilder provides all the tools you need to build industrial-strength applications, such as order entry, accounting, and manufacturing systems.

PowerBuilder applications consist of windows that contain controls that users interact with. You can use all the standard controls—such as buttons, checkboxes, dropdown listboxes, and edit boxes—as well as special PowerBuilder controls that make your applications easy to develop and easy to use.

PowerBuilder supports cross-platform development and deployment. For example, you can develop an application using PowerBuilder under Windows and deploy the very same application—without changes—on the Macintosh. Or vice versa. You can even have a cross-platform team of developers, some using Windows and some using the Macintosh, developing the same application at the same time. They can freely share PowerBuilder objects used in the application, because the objects are the same across the different computing platforms that PowerBuilder supports.

For more information about cross-platform development using PowerBuilder, use the Powersoft FaxLine system, which provides up-to-the-minute documents about PowerBuilder that you can have faxed to you. Call the system and get a catalog of available documents, then request the document you want.

About the painters

You build the components of your application using **painters**, which provide an assortment of tools for building objects.

For example, you build a window in the Window painter. There you define the properties of the window and add controls, such as buttons and edit boxes.

PowerBuilder provides a painter for each type of object you build.

About events and scripts

PowerBuilder applications are event-driven: users control the flow of the application by the actions they take. For example, when a user clicks a button, chooses an item from a menu, or enters data into an edit 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 whenever the user clicks the button. Similarly, edit boxes have a Modified event, which is triggered each time the user changes a value in the box.

You write scripts using PowerScript, the PowerBuilder language. Scripts consist of PowerScript commands, functions, and statements that perform processing in response to an event.

For example, the script for a button's Clicked event might retrieve and display information from the database; the script for an edit box'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 Clicked event in a button might open another window, which triggers the Open event in that window.

About functions

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

In addition, you can build your own functions to define processing unique to your application.

About libraries

You save your objects, such as windows and menus, in PowerBuilder libraries (PBL files). When you run your application, PowerBuilder retrieves the objects from the library. PowerBuilder provides a Library painter for you to manage your libraries.

Creating an executable

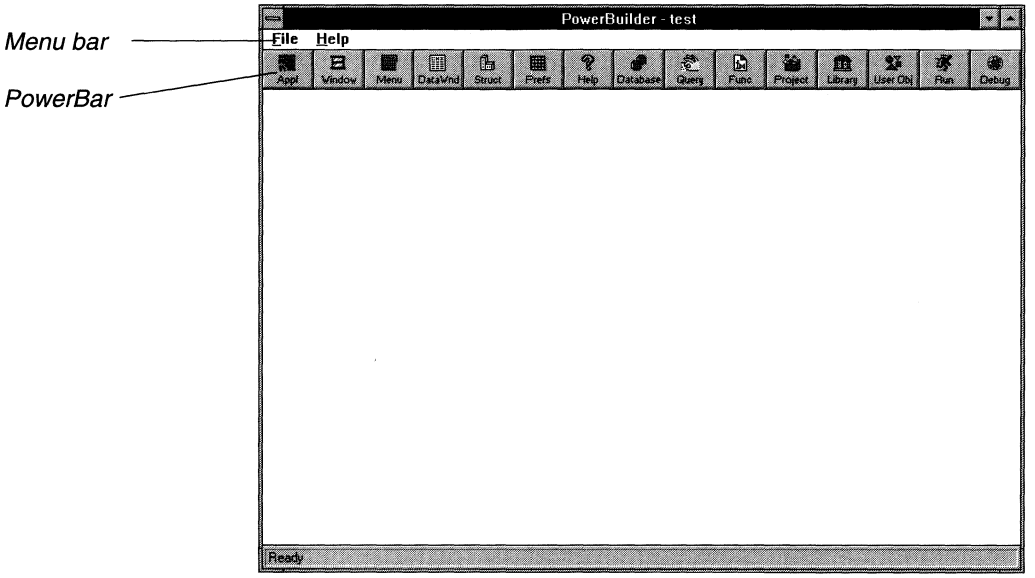
When you have completed your application, you create an executable version to give to your users. PowerBuilder provides an easy way to package your application for distribution.

The PowerBuilder environment

When you start PowerBuilder, it opens in a window that contains a menu bar and the PowerBar.

Starting PowerBuilder

For complete information about starting PowerBuilder, see *Getting Started.*



You can open painters and perform other tasks by clicking buttons in the PowerBar.

About the PowerBar

The PowerBar displays when you begin a PowerBuilder session. The PowerBar is the main control point for building PowerBuilder applications. From the PowerBar, you can open a PowerBuilder painter, debug or run the current application, request help, or customize PowerBuilder to meet your needs.

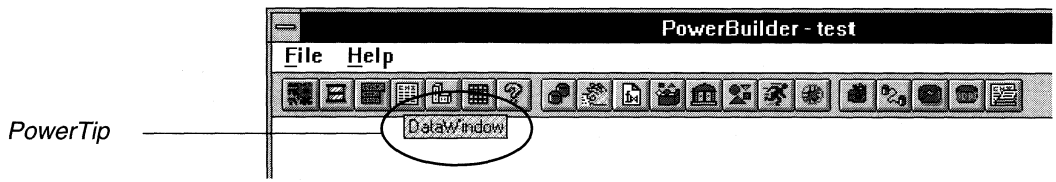
Customizing the PowerBar

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

For more information, see "Using the toolbars" on page 22.

About PowerTips

By default, when text is not displayed in the PowerBar (and in PainterBars, which are toolbars for the painters), PowerBuilder displays a brief description of the button (a "PowerTip") when you leave the mouse pointer over a button for a second or two.



About the PowerPanel

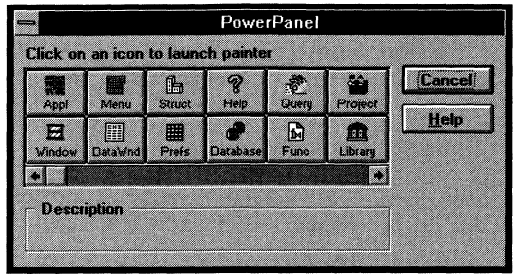
PowerBuilder also provides a PowerPanel, which, like the PowerBar, provides buttons for opening painters and performing other activities. While the PowerPanel is not customizable, it does contain all painters and tools that are globally available throughout PowerBuilder.

You will usually use the PowerBar to open painters, but the PowerPanel is handy if you want to open a painter that is not currently in the PowerBar.

❖ To use the PowerPanel:

1 Select File➤PowerPanel from the menu bar.
or
Press CTRL+P.

The PowerPanel displays.



2 Click the button that represents the painter you want to open, as described in the next section.

The painter opens.

Using the PowerBar and PowerPanel

The buttons in the PowerBar and PowerPanel represent each of the main painters and tools frequently used in PowerBuilder:

Application painter, in which you specify information about your application, such as its name and the PowerBuilder libraries in which the application's objects will be saved

Window painter, in which you build the windows that will be used in the application

Menu painter, in which you build menus that the windows will use

DataWindow painter, in which you build intelligent objects called DataWindow objects that present information from the database

Structure painter, in which you define structures (groups of variables) for use in your application

Preferences, which you use to customize PowerBuilder by setting its properties

Help, which invokes the PowerBuilder online Help system to give you instant answers to questions

Database painter, in which you maintain databases, control user access to databases, and manipulate data in databases

Query painter, in which you graphically define and save SQL SELECT statements for reuse with DataWindow objects

Function painter, in which you build functions to perform processing specific to your application

Project painter, in which you create your executable by specifying the components that go into the application

Library painter, which you use to create and maintain libraries of PowerBuilder objects

User Object painter, which you use to build custom objects that you can save and use repeatedly in windows

Run, which runs your current PowerBuilder application just as your users would run it

Debug, which allows you to set breakpoints, run your application a statement at a time, and look at variables during execution

Report painter, in which you build reports (a type of DataWindow object)

Database Profiles, in which you specify how to connect to your database

Configure ODBC, in which you define a data source that uses ODBC

Data Pipeline painter, in which you transfer data from one data source to another

Database Administration painter, in which you perform DBA tasks, such as maintaining users and security

Using the painters

This section describes features that are common to all painters.

Opening a painter or tool

❖ To open a painter or use a tool from the PowerBar or PowerPanel:

- Click the button in the PowerBar or PowerPanel that represents the painter or tool.

> **Opening a recently used object**
> You can quickly open any of the last four objects you worked on—PowerBuilder lists them at the bottom of the File menu. To open one of them, open the File menu and select the object.

Using shortcut keys

You can also use the following shortcut keys from anywhere within PowerBuilder to open a painter or tool:

To open the	Press
Application painter	SHIFT+F1
Window painter	SHIFT+F2
Menu painter	SHIFT+F3
DataWindow painter	SHIFT+F4
Structure editor	SHIFT+F5
File editor	SHIFT+F6
Database painter	SHIFT+F7
Query painter	SHIFT+F8
Function painter	SHIFT+F9
Library painter	SHIFT+F10
User Object painter	SHIFT+F11

Each painter or tool is described in detail later in this manual.

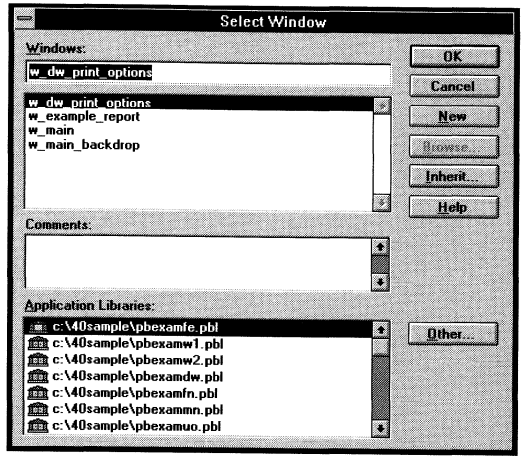
Opening an object

When you open a painter, PowerBuilder displays a Select *object* dialog box, allowing you to specify which object you want to open.

❖ To open an object in a painter:

1 Open the painter.

PowerBuilder displays a dialog box. All the painters have a Select dialog box that works the same way. Here is the Select Window dialog box displayed in the Window painter.



2 Select an existing object.

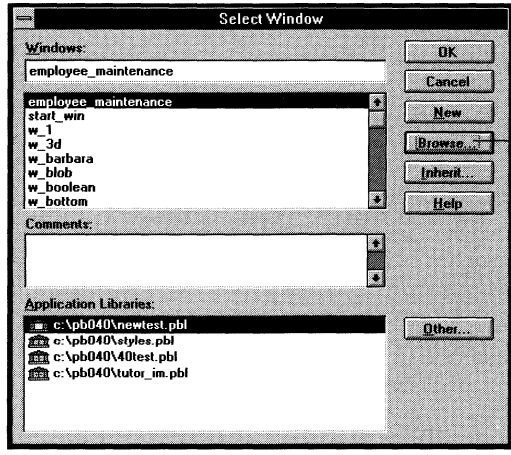
- If the object is in the current library, select it in the list and click OK.
- If the object is in another library on the application's library search path, select the library in the Application Libraries box, then select the object and click OK.
- If the object is in a library that is not in the application's library search path, click the Other button. PowerBuilder displays a dialog box that lets you specify another library to add to the list in the Application Libraries box; objects in that library display in the list at the top of the dialog box. Select the object and click OK.

You can display one library that is not in search path
In the Select *object* dialog boxes, you can display up to one library that is not in the application's library search path. The second time you click Other and select a library, it replaces the previously selected library that is not in the search path.

- If you want to search for an object based on its contents, use the Browser (see the next section).

3 PowerBuilder opens the object in the painter.

Using the browser



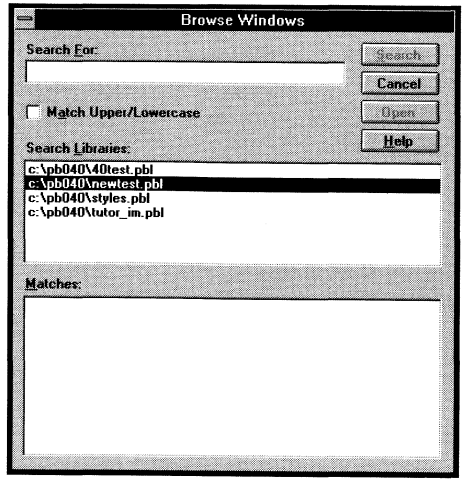
Click here to search for objects using a browser

To help you find the object you want to work on, PowerBuilder provides a browser that you can use to list all objects that contain a specified text string.

❖ To use the browser:

1 Click the Browse button in the Select dialog box.

The Browse dialog box displays. Here is the Browse dialog box in the Window painter:



2 To search for an object, enter all or part of the text you want to locate (the search string) in the top box. Wildcards are not allowed.

3 Specify the libraries that you want searched (all libraries in the application's library search path are listed).

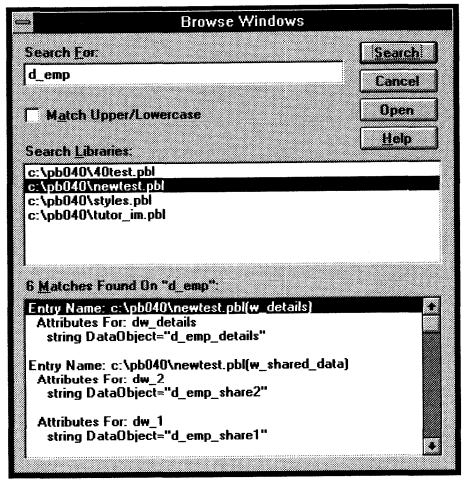
4 Click Search.

What happens

PowerBuilder begins searching for the text. If you are in the Window painter, PowerBuilder searches through all windows; if you are in the Menu painter, PowerBuilder searches through all menus; and so on. PowerBuilder searches all attributes, scripts, variables, functions, and structures for the search string.

PowerBuilder considers a match to be any text that contains the search string. All matches are displayed in the Match box. For each match, PowerBuilder displays the matching text and information about where the match was found (for example, the line of script or the attribute of an object or control that contains the search string).

Here is the result of a search for the string *d_emp*, which shows all the windows that use a DataWindow object whose name begins with d_emp. PowerBuilder found six matches:



Opening the object

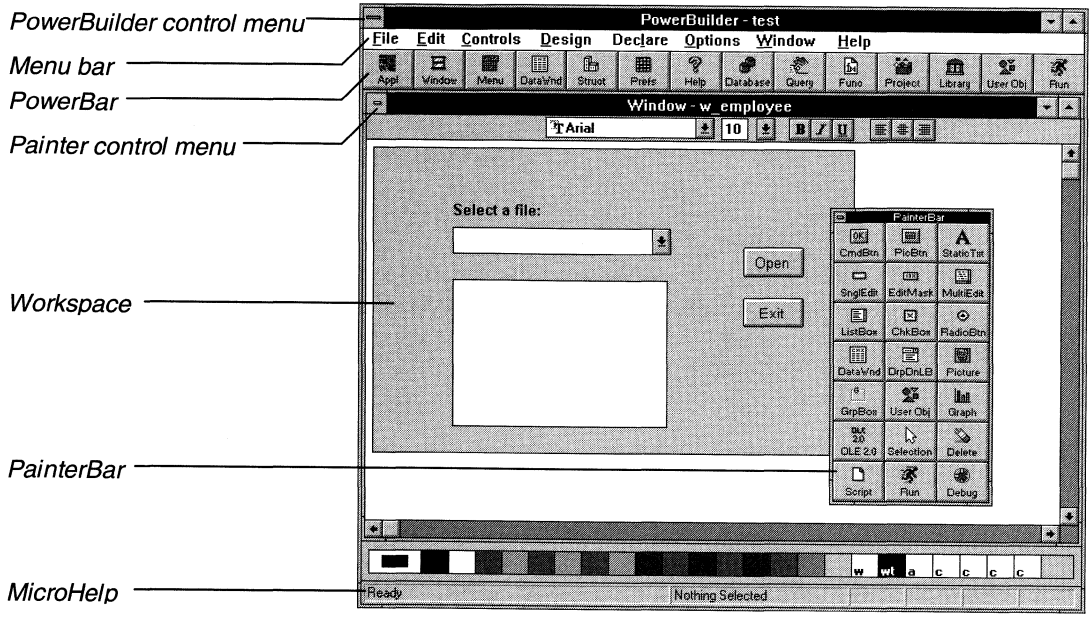
After you locate the object, you can directly open it.

❖ To open the object:

- Double-click the object you want to open or select the object from the list in the Match listbox and click the Open button.

Components in all painters

Although each painter is unique, certain components are common to all painters. You can see these components in the following screen, which shows the Window painter.



Control menus

The *PowerBuilder* control menu is in the upper-left corner of the PowerBuilder window. The *PainterBar* control menu is in the upper-left corner of the painter window. Both menus allow you to use the keyboard instead of the mouse to manipulate windows.

The items in the control menus are the standard Microsoft Windows items:

Menu item	Use to
Restore	Return a window to its previous size and make it the active window
Move	Move the active window to a new screen location
Size	Change the size of the active window
Minimize	Reduce the active window to a button

Menu item	Use to
Maximize	Enlarge the active window to full screen size
Close (ALT+F4)	(PowerBuilder control menu) Close the active windows and exit PowerBuilder
Close (CTRL+F4)	(Painter control menu) Close the active painter, but not exit PowerBuilder
Next (CTRL+F6)	(Painter control menu) Switch to the next open painter
Switch To (CTRL+ESC)	(PowerBuilder control menu) Display the Windows Task List window, which lists all open applications

Using the keyboard
You can use the keys shown in parentheses in the table to select an item when the item is not displayed. To select an item with the keyboard when the item is displayed, use the underlined letter in the item name.

The title bar

The title bar (top center) identifies the window. The PowerBuilder title bar displays the name of the current application. The painter title bar displays the object type and the name of the current object. In the preceding Window painter window, the title bar indicates that you are working on a window named w_employee.

Minimize and Maximize boxes

The Minimize and Maximize boxes enlarge the active painter to fill the PowerBuilder workspace or reduce it to a button, which displays at the bottom of the PowerBuilder window.

After you maximize the window, the Maximize box becomes a Restore box that you can click to restore the painter to its previous size.

The menu bar

The menu bar (under the title bar) lists the top-level menu items for the active painter. Each item on the menu bar has a dropdown menu that lists available items that are related to the top-level item.

The items in the menu bar and the items on the dropdown menus vary for each painter. The following table lists the menu items that display in most painters, and their use:

Menu bar item	Items	Use to
File	New	Clear the painter workspace so you can build a new object (such as a new window or menu). If there is an unsaved object in the workspace, PowerBuilder prompts you to save the object before it opens the new object.
	Open	Open an object that was created in the painter (such as a window in the Window painter). If there is an unsaved object in the workspace, PowerBuilder prompts you to save it before it opens the object.
	Close (CTRL+F4)	Close the painter. PowerBuilder prompts you to save your work, then closes the painter.
	Save	Save the current object under its current name and in the same PowerBuilder library.
	Save As	Save the current object under a new name in the same library or a different library.
	Run (CTRL+R)	Run the current application.
	Debug (CTRL+D)	Debug the current application.
	PowerPanel (CTRL+P)	Display the PowerPanel.
	Print	Print the definition of the current object.
	Printer Setup	Open the standard Windows Printer Setup dialog box.
	Exit	End your PowerBuilder session. PowerBuilder prompts you to save your work, then closes all painters and terminates your PowerBuilder session.
	1, 2, 3, 4	Open a recently used object.
Edit		Display editing options provided by the painter.
Window		Arrange the open windows and buttons, manipulate the toolbars, or activate another open window.

Menu bar item	Items	Use to
Help		Access online Help.

Using the keyboard
You can use the keys shown in parentheses in the table to select an item when the item is not displayed. To select an item with the keyboard when the item is displayed, use the underlined letter in the item name.

MicroHelp

At the bottom of the PowerBuilder window is a MicroHelp line, which PowerBuilder uses to display status information throughout a session. For example, if you want to see what a particular menu item does, select it and read its description in MicroHelp.

The workspace

The painters provide a workspace, in which you work with the object you are building. For example, in the Window painter workspace you build the window by placing controls. In the Database painter workspace you work with database tables, indexes, and keys.

Toolbars

PowerBuilder provides several toolbars, which make working quick and easy.

For more information, see "Using the toolbars" on page 22.

Changing fonts

You can change the fonts that PowerBuilder uses in some painters and in MicroHelp.

In the PowerScript painter, Database Administration painter, and file editor

To change the font used in the PowerScript painter, Database Administration painter, and file editor, add or modify the following variables in the [pb] section in the PB.INI file (you can use the Preferences painter if you want, as described in Chapter 24, "Customizing PowerBuilder"):

```
[pb]
EditorFontName=NameOfFont
EditorFontHeight=PointSize
EditorFontBold=0 or 1
EditorFontFixed=0 or 1
EditorTabWidth=NumberOfCharacters
```

Item	Description
EditorFontName	Name of the font. Don't use quotation marks.
EditorFontHeight	Point size of the font.
EditorFontBold	Specifies format of the text. Values are: 1 — boldface text 0 — not bold text
EditorFontFixed	Specifies whether PowerBuilder should substitute a fixed-width font or a variable-width font. Used only if PowerBuilder cannot find the named font. Values are: 1 — fixed-width font 0 — variable-width font
EditorTabWidth	Specifies the tab settings. Default is 3 characters.

For example, the following lines specify that 11-point Arial be used in the PowerScript painter, Database Administration painter, and file editor:

```
[pb]
EditorFontName=Arial
EditorFontHeight=11
```

In the Application painter, Library painter, and MicroHelp

To change the font used in the Application painter, the Library painter, and MicroHelp, add or modify the following variables in the [pb] section of PB.INI.

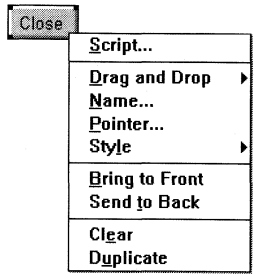
```
[pb]
FontName=NameOfFont
FontHeight=PointSize
FontBold=0 or 1
FontFixed=0 or 1
```

The variables work the same way as the corresponding variables that begin with *Editor*, described in the preceding section.

Using the popup menu

PowerBuilder provides a context-sensitive popup menu that lists items appropriate to the currently selected object on the screen or the current position of the pointer. The popup menu is available everywhere in PowerBuilder. You will find that the popup menu helps you work quickly.

For example, if the pointer is on a command button that you have placed in a window in the Window painter, the popup menu lists items that apply to a command button.



❖ To use a popup menu:

1 Select one or more objects, or position the pointer on an object or in open space.

To select *one object*, click it. To select *multiple objects*, press and hold CTRL, and click each of the objects.

2 Click the right mouse button. (On the Macintosh, hold down the CONTROL key and press the mouse button.)

The appropriate popup menu displays.

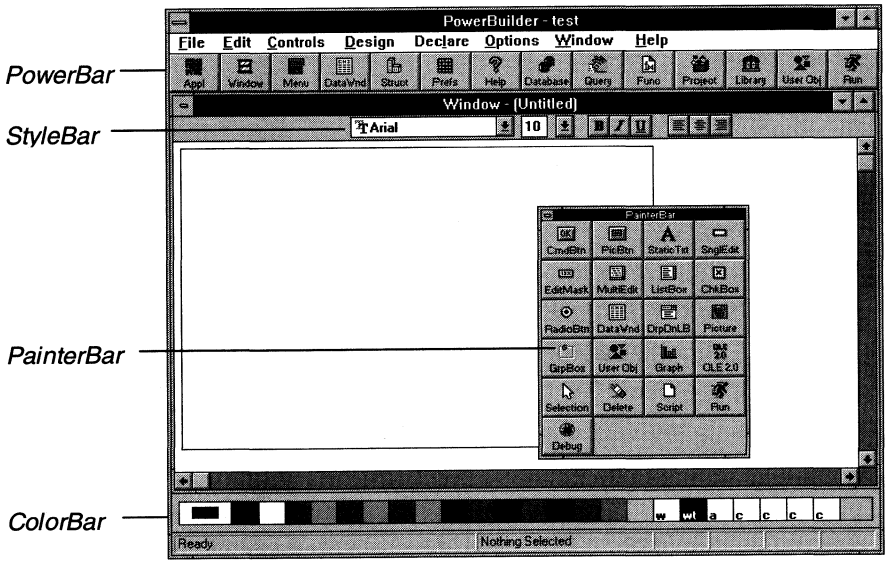
3 Click the item you want. If other windows display, continue supplying information as appropriate.

Using the toolbars

Toolbars make working with PowerBuilder easier and faster. They contain buttons that you can use to open painters and tools, manipulate controls, and execute commands.

About toolbars

PowerBuilder uses four toolbars. Here is the Window painter with all toolbars displayed. The PainterBar is shown as floating.



PowerBar The PowerBar has buttons for opening painters and other tools.

StyleBar The StyleBar has buttons for changing the attributes of text such as typeface, point size, and font.

PainterBar The PainterBar has the buttons for manipulating components in the current painter.

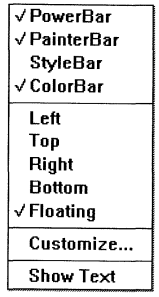
ColorBar The ColorBar has buttons for changing colors of components in the current painter and for defining custom colors.

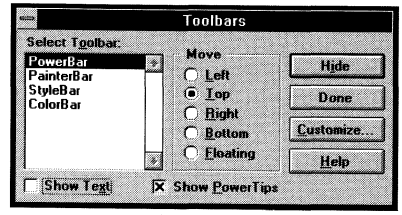
Controlling the display of toolbars

You can control whether to display individual toolbars. If you display a toolbar, you can choose where to display it and whether to display text on the buttons. Choosing to display text affects all toolbars: either all toolbars display text or all omit text.

If you are not displaying text in the buttons, you can decide whether to display PowerTips.

You can use either the popup menu or the Toolbars dialog box to control toolbars.





Both let you specify:

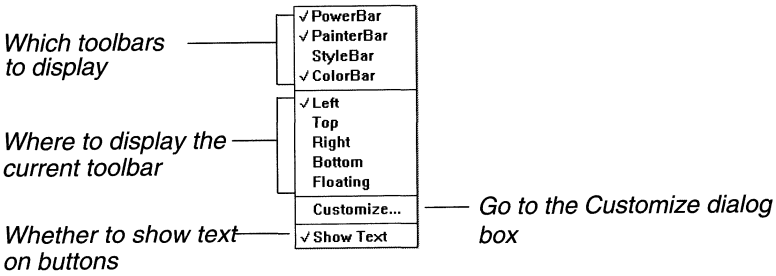
- Whether to display a toolbar
- Where to display it
- Whether to display text on buttons
- That you want to customize a toolbar

You can also specify whether you want to use PowerTips using the Toolbars dialog box.

❖ To control a toolbar using the popup menu:

1 Position the pointer on the toolbar and display the popup menu.

The popup menu for the toolbars displays.

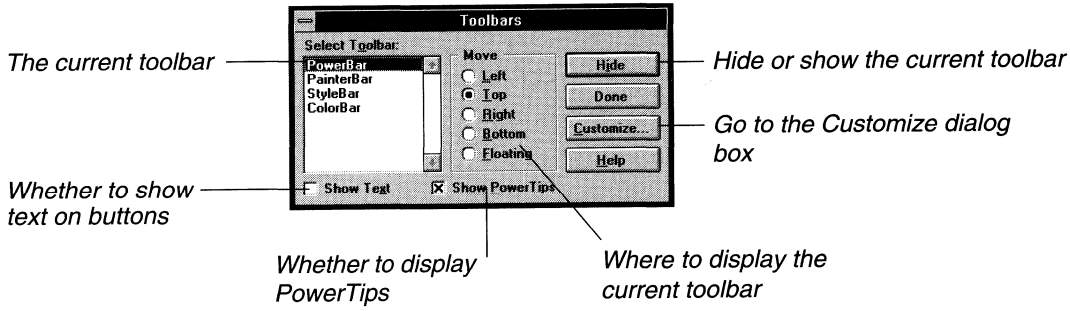


2 Click the items you want. A checkmark means the item is currently selected.

❖ To control a toolbar using the Toolbars dialog box:

1 Select Windows➤Toolbars from the menu bar. (If no painter is open, select File➤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.

3 Specify whether you want to display text in toolbars and, if not, specify whether you want to display PowerTips.

4 Click Done when you have finished.

How the information is saved
PowerBuilder records your toolbar preferences in PB.PBT in the PowerBuilder directory.

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 empty space within the toolbar.

2 Press and hold the left mouse button.

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

As you move, an outlined box shows the current position and type of the toolbar. When you get close to an edge, the box is a narrow rectangle, which means the toolbar is located at that edge (for example, the bottom). When you are in the middle area of the screen, the box is more square, which means the toolbar is floating.

Customizing toolbars

You can customize the toolbars with PowerBuilder buttons and with buttons that invoke other applications, such as a calculator or notepad.

Adding, moving, and deleting buttons

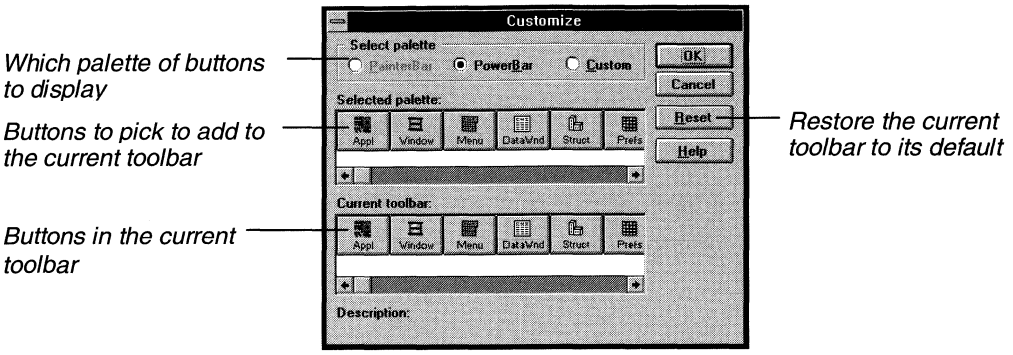
You can add, move, and delete buttons in the PowerBar and in the PainterBars. You cannot access buttons in the StyleBar or ColorBar.

❖ To add a button to a toolbar:

1 Position the pointer on the toolbar and press the right mouse button to display the popup menu.

2 Select Customize.

The Customize dialog box displays.



3 Click the palette of buttons you want to use in the Select palette group.

4 Choose a button from the Selected palette area (you may need to scroll to see all available buttons). If you choose a button from the Custom palette, another dialog box displays so you can define the button.

For more on custom buttons, see "Adding a custom button" on page 27.

Seeing what's available in the PowerBar
PowerBuilder provides several handy buttons that do not display by default on 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.

5 Position the pointer on your choice, press and hold the left mouse button, drag the button to the position you want in the Current toolbar box, and drop the button.

PowerBuilder adds the button to the current toolbar.

❖ To move a button on a toolbar:

1 Display the Customize dialog box.

2 Position the pointer on the button to move in the Current toolbar box at the bottom of the dialog box.

3 Press and hold the left mouse button, then drag the button to the position you want.

❖ To delete a button from a toolbar:

1 Display the Customize dialog box.

2 Position the pointer on the button to delete in the Current toolbar box at the bottom of the dialog box.

3 Press and hold the left mouse button, then drag and drop the button anywhere outside the Current toolbar box.

Resetting a toolbar

Whenever you want, you can restore the original setup of buttons on a toolbar.

❖ To reset a toolbar:

1 Position the pointer on the toolbar and click the right mouse button to display the popup menu.

2 Select Customize.

The Customize dialog box displays.

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

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 outside PowerBuilder
- Run a query or report
- 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:

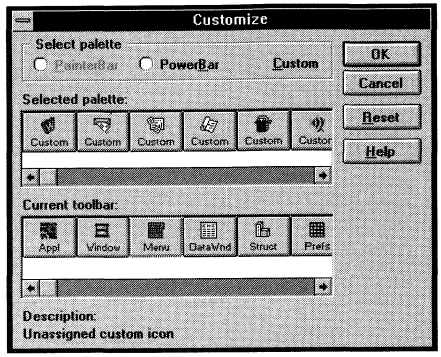
1 Position the pointer on the toolbar and click the right mouse button to display the popup menu.

2 Select Customize.

The Customize dialog box displays.

3 Click Custom in the Select palette group.

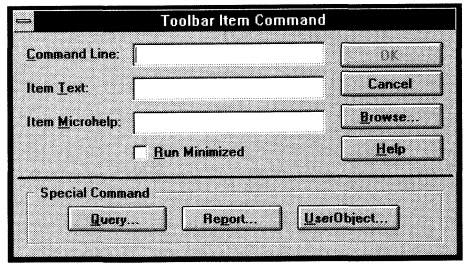
The custom buttons display in the Selected palette group.



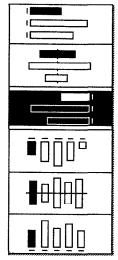
4 Choose a custom button.

5 Position the pointer on your choice, press and hold the left mouse button, then drag the button to the position you want for it in the Current toolbar box.

As soon as you drop it, the Toolbar Item Command dialog box displays. The dialog box contains three edit boxes. It also contains buttons to make it easier to add custom functionality. Which buttons display depend on which toolbar you are customizing, as described in the next step.



6 The Command Line box specifies the action that the custom button should perform. Fill in the Command Line box as follows:

To have the button	Do this
Invoke a PowerBuilder menu item 	Type ***@MenuBarItem.MenuItem*** in the Command Line box. For example, to have the button mimic the Open item on the File menu, type **@File.Open** You can also use a number to refer to a MenuItem. The first item in a dropdown/cascading menu is 1, the second item is 2, and so on. Separator lines in the menu count as items. For example, to assign a button to the third selectable item in the Edit➢Align Controls menu in the Window painter, type **@Edit.Align Controls.5** (there are two separator lines above the item, so the item is numbered 5).
Run an executable outside PowerBuilder	Type the name of the executable in the Command Line box. Specify the full path name if the executable is not in the current search path. To use the browser to find the filename, click the Browse button.
Run a query	Click the Query button and select the query from the displayed list.
Run a report (same as previewing a DataWindow object)	Click the Report button and select a report from the displayed list of DataWindow objects. You can then specify command-line arguments in the Command Line box, as described below. Reports and DataWindow objects are fundamentally the same. Reports are created in the Report painter. For more information, see Chapter 13, "Defining DataWindow Objects."
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.

To have the button	Do this
Assign a display format to a column in a DataWindow object	(DataWindow painter only) Click the Format button; the Display Formats window displays. 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 15, "Displaying and Validating Data."
Create a computed field in a DataWindow object	(DataWindow painter only) Click the Function button; the Function for Toolbar window displays. Select the function from the list.

7 In the Item Text box, specify the text associated with the button in two parts: the text that displays on the button and text for the button's PowerTip. Separate the two parts with a comma, as follows:

ButtonText, PowerTip

For example, specifying

```
Save, Save File
```

puts the text **Save** on the button when text is displayed; when text is not displayed, **Save File** displays as the button's PowerTip.

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

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

9 Click OK.

PowerBuilder adds the custom button to the toolbar.

Supplying arguments with reports

If you defined the button to run a report, you can specify arguments in the command line in the Toolbar Item Command dialog box.

Argument	Meaning
/l *LibraryName*	Specifies the library containing the report
/o *ReportName*	Specifies the report
/r	Runs the report

Argument	Meaning
/ro	Runs the report but does not provide design mode for modifying the report
/a "*Arguments*"	Specifies arguments to pass to the report

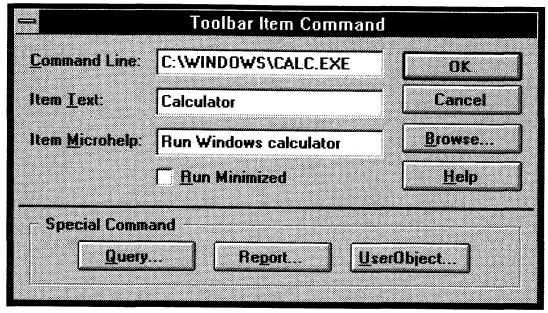
The default command line is:

Report /o *ReportName* /ro

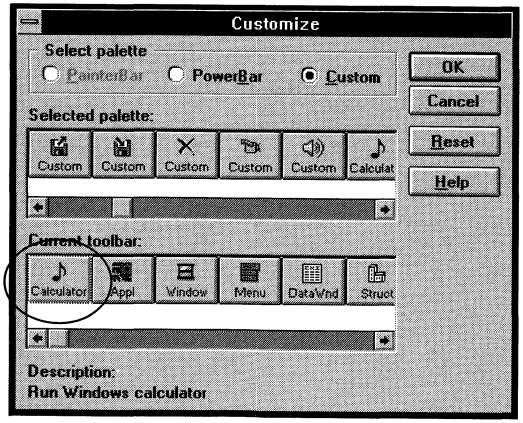
Example

In the following example, the Windows calculator is associated with the musical note button.

Here is the filled-in Toolbar Item Command dialog box:



Here is the resulting Customize dialog box. The calculator button is in the PowerBar and has text and MicroHelp:



Modifying a custom button

❖ To modify a custom button:

1 Open the Customize dialog box.

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 27.

Working with PowerBuilder windows

Each PowerBuilder painter displays in its own window. You can open several painters at once, and you can open several instances of a painter at once. For example, if you want to see several windows that you are building at once, you open the Window painter several times and specify a different window each time.

Opening multiple windows

❖ To open each window:

1 Click one of the painter buttons in the PowerBar.

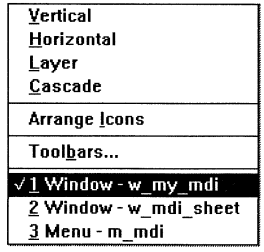
The Select dialog box for the object (window, DataWindow object, and so on) displays.

2 Select an existing object or begin working on a new object.

The painter opens in a window.

Open windows are listed

Like other Windows applications, PowerBuilder lists open windows as items on the Window menu in the menu bar. The following menu indicates that there are three open windows: two containing a window and one containing a menu. The window with the checkmark is the active one.



Making a window active

A window must be the active window for you to work in it.

❖ To make a visible window the active window:

- Click anywhere in the window.

❖ To make a window not currently displayed the active window:

1. Select Window from the menu bar.
2. Choose the window from the list.

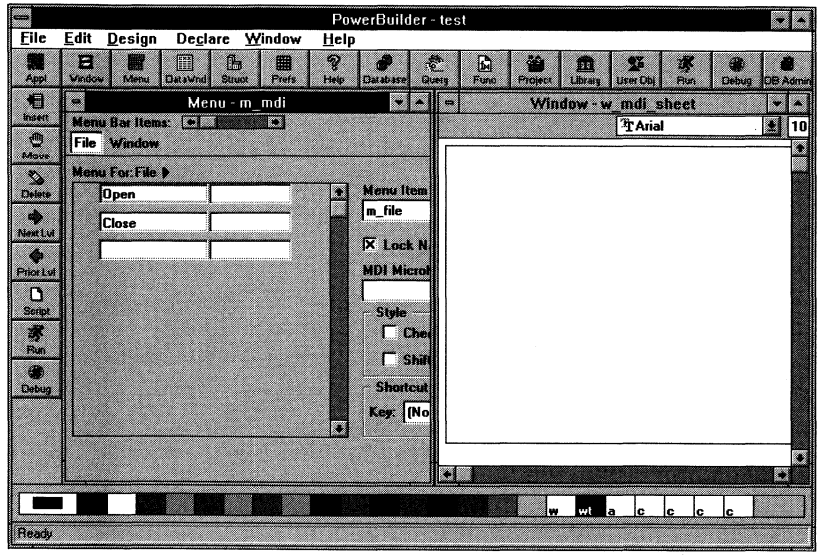
Tiling windows

Tiling allows you to see the contents of more than one window at once. PowerBuilder allows you to tile windows horizontally or vertically. When you tile windows, PowerBuilder displays the contents of all currently open windows in the available space.

For example, you can display the Database painter next to the DataWindow painter and switch between the two as you work on a DataWindow object. Or you may want to display a menu in one window and a window that uses that menu in another window.

This screen shows two windows that are tiled vertically.

The window on the left is m_mdi open in the Menu painter. The window on the right is w_mdi_sheet open in the Window painter.



When you work on a tiled screen, the PainterBar you see goes with the currently active window. Scrollbars allow you to move to areas of the active window not currently displayed

❖ To tile windows:

- Select Window➤Horizontal from the menu bar to tile horizontally.
  or
  Select Window➤Vertical from the menu bar to tile vertically.

  All open windows display in the workspace.

Layering windows

When you specify Layer, PowerBuilder displays the currently active window in the entire workspace.

❖ To layer windows:

- Select Window➤Layer from the menu bar.

  The currently active window fills the workspace. To display another open window, select its name from the Window menu.

Cascading windows

When you specify Cascade, PowerBuilder displays all currently open windows, one on top of another. The currently active window is on top, with its contents displayed. The other windows are behind, with title bars displayed. To make another window active, click in the window's title bar.

❖ To cascade windows:

- Select Window➤Cascade from the menu bar.

  The open windows display in a stack.

Using the file editor

PowerBuilder provides a text editor that is available anywhere in PowerBuilder. Using the editor, you can view and modify text files (such as INI files, files containing SQL statements, and TXT files) without leaving PowerBuilder.

❖ To open the file editor:

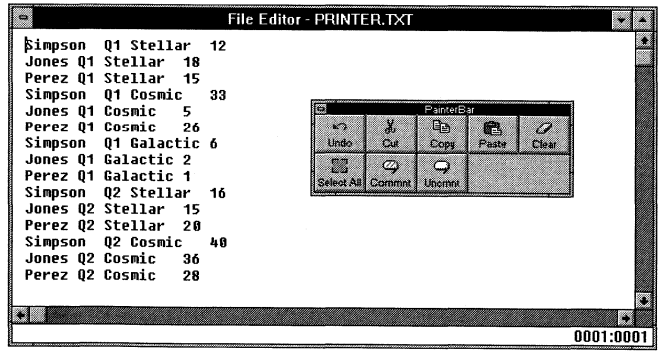
1 Press SHIFT+F6 anywhere in PowerBuilder.

> **Adding a button**
> You can add a button that invokes the file editor to a toolbar. The button is available from the PowerBar palette.
>
> For more information, see "Customizing toolbars" on page 25.

The File Open dialog box displays.

2 Specify the file you want to edit.

The file editor opens.



The editor provides the same editing capabilities as the PowerScript painter.

For more information, see Chapter 3, "Writing Scripts."

Executing AppleScript scripts on the Macintosh

On the Macintosh, you can execute an AppleScript script (either in source or compiled form) from within the file editor.

❖ To open an AppleScript file for execution:

1 Open the file editor.

2 Do one of the following:

- Select File➤Open from the menu bar and select a source file containing AppleScript.

  PowerBuilder reads the contents of the source file into the file editor.

- Type the name of a file in the empty file editor (do not enclose the filename with quotation marks). The file can be either a source file containing AppleScript or a compiled AppleScript script.

- Create a new script by typing AppleScript commands in the file editor. You can save the new script using File➤Save from the menu bar.

❖ To execute an AppleScript script:

- Select Edit➤Execute AppleScript from the menu bar.

  The contents of the file editor are executed as an AppleScript script. If you have typed the name of a file in the editor, PowerBuilder executes the script in the specified file.

  PowerBuilder displays a dialog box with the results.

Executing AppleScript scripts during execution
You can use the DoScript function to execute an AppleScript script during execution.

For more information, see the *Function Reference*.

Using online Help

PowerBuilder has extensive online Help, which supplements the information in the PowerBuilder manuals and provides both reference and task-oriented information.

❖ To access PowerBuilder online Help:

- Use the Help menu on the menu bar, the Help button in the PowerBar, the Help button in a dialog box, or press F1.

When you use the Help PowerBar button or F1, PowerBuilder displays the Help contents topic. From there you can move to other topics.

When you click the Help button in a dialog box, PowerBuilder displays information about the dialog box. From there you can move to other topics.

❖ To learn to use online Help:

- From anywhere *within* online Help, press F1.

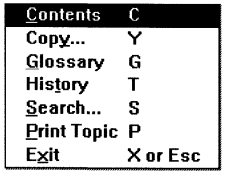
  You go to the topic "How to use Help."

Getting context-sensitive help

You can press SHIFT+F1 to get context-sensitive help in the PowerScript painter and Function painter.

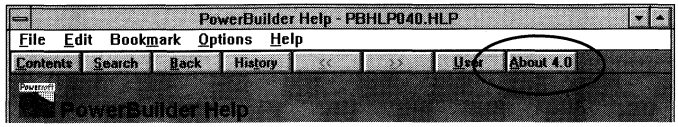
For more information, see Chapter 3, "Writing Scripts."

Using the popup menu



PowerBuilder online Help provides a popup menu with shortcuts to features available on the menu bar.

Learning about new features

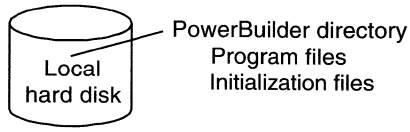


Click the About 4.0 button at the top of the PowerBuilder Help window to learn about features that are new in Version 4.0.

PowerBuilder initialization files

Normally, you have PowerBuilder installed on your personal computer and run it locally. When starting, by default PowerBuilder looks for two initialization files in the PowerBuilder directory to set up your environment:

Initialization file	Purpose
PB.INI	Records your preferences
PB.PBT	Records your toolbar settings



For more about PB.INI, see Chapter 24, "Customizing PowerBuilder."

For more about PB.PBT and toolbar settings, see "Controlling the display of toolbars" on page 23.

Running PowerBuilder on a network

If you are in a networked site, you can also have PowerBuilder installed on the network and run it from the network. But if you run PowerBuilder on a network, you don't want to use initialization files in the PowerBuilder directory on the network; you want to use your own initialization files installed on your computer.

Telling PowerBuilder where your initialization files are

If your initialization files are not in your PowerBuilder directory, you need to tell PowerBuilder where it can find them by adding the following lines to the WIN.INI file in the Windows directory on your PC:

```
[PowerBuilder]
INITPATH040=drive:directory
```

where *drive:directory* is the drive and directory containing the initialization files on your PC.

For example, if your PB.INI and PB.PBT files are in the directory c:\pbfiles, you would specify the following in your WIN.INI file:

```
[PowerBuilder]
INITPATH040=c:\pbfiles
```

How PowerBuilder finds your initialization files

When starting, PowerBuilder Version 4.0 looks first for a definition of INITPATH040 in the [PowerBuilder] section of WIN.INI for where to find your initialization files. If INITPATH040 is not defined, PowerBuilder looks for a definition of INITPATH in the [PowerBuilder] section of WIN.INI (this is for compatibility with PowerBuilder Version 3.0, which looks first for the INITPATH variable). If neither variable is defined, PowerBuilder uses the PB.INI and PB.PBT files in the PowerBuilder directory (if they do not exist, PowerBuilder creates new ones with default specifications).

Building an application

This section describes the basic steps you follow when building a PowerBuilder 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.

> **Use the *Building Applications* manual**
> ☞ For complete information about the process of building applications in PowerBuilder, see *Building Applications*.

❖ To build an application:

1 **Create the application object** This is the entry point into the application. The application object names the application, specifies which libraries to use to save the objects, and specifies the application-level scripts.

☞ See Chapter 2, "Working with Applications."

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

☞ See Part Three, "Working with Windows."

3 **Create DataWindow objects** Use these objects to retrieve data from the database, format and validate data, analyze data through graphs and crosstabs, create reports, and update the database.

☞ See Part Four, "Working with Databases."

4 **Create menus** Menus in your windows can include a menu bar, dropdown menus, and cascading menus. You can also create popup menus in an application. You define the menu items and write scripts that execute when the items are selected.

☞ See Chapter 9, "Working with Menus."

5 **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 instead of having to redefine the components.

☞ See Chapter 10, "Working with User Objects."

6 **Create functions and structures** To support your scripts, you probably want to define functions to perform processing unique to your application and structures to hold related pieces of data.

See Chapter 4, "Working with User-Defined Functions," and Chapter 5, "Working with Structures."

7 **Test and debug your application** You can run your application anytime. If you discover problems, you can debug your application by setting breakpoints, stepping through your code statement by statement, and looking at variable values during execution.

See Chapter 21, "Debugging and Running Applications."

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

See Chapter 22, "Creating an Executable."

CHAPTER 2

Working with Applications

About this chapter

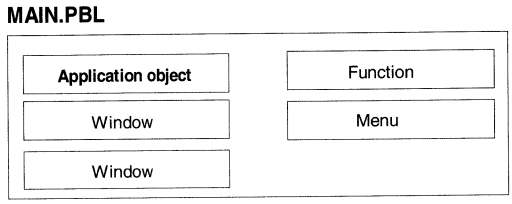
In PowerBuilder, you are always working within the context of an application. The entry point to an application is the application object. This chapter describes application objects.

Contents

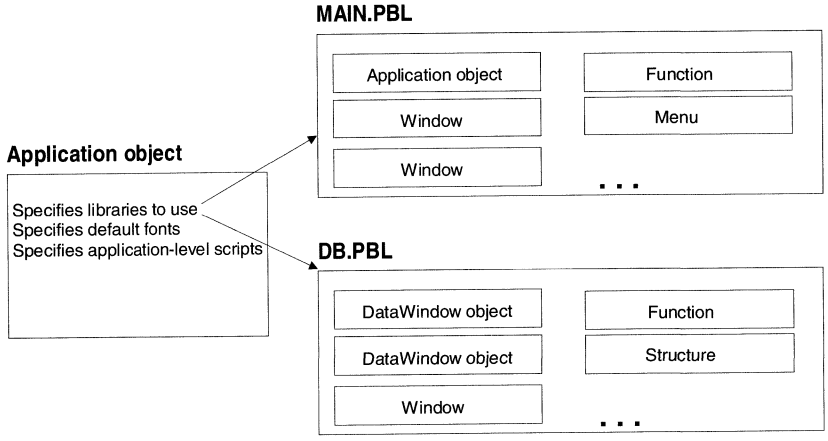
Overview

An **application** is a collection of PowerBuilder windows that perform related activities, such as order entry, accounting, or manufacturing activities. It is what you deliver to your users.

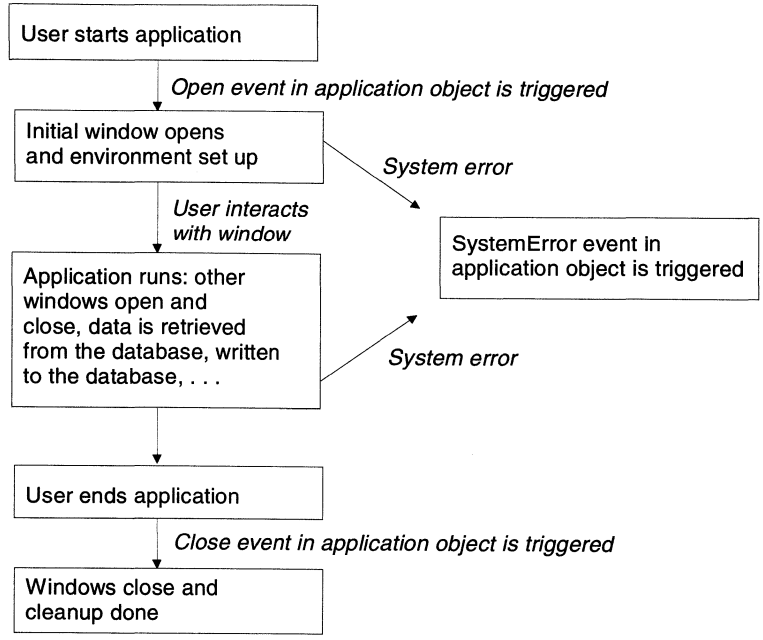
The **application object** is the entry point into the windows that perform these activities. It is a discrete object that is saved in a PowerBuilder library, just like a window, menu, function, or DataWindow object.



The application object defines application-level behavior, such as which libraries contain the objects that are used in the application, which fonts are used by default for text, and what processing should occur when the application begins and ends.



When a user runs the application, an Open event is triggered in the application object. The script you write for the Open event initiates the activity in the application. When the user ends the application, the Close event in the application object is triggered. The script you write for the Close event typically does all the cleanup required, such as closing a database or writing out a preferences file. If there are serious errors during execution, the application object's SystemError event is triggered.



Creating a new application object

The first step in building a new PowerBuilder application is to create an application object for the application. In the application object, you:

- Assign the application a name and icon
- Establish the default text colors, sizes, styles, and fonts for the application
- Specify the libraries the application can use, in the sequence in which you want them to be searched during execution
- Build the application-level scripts

You use the Application painter to create an application object and specify its properties.

Using an existing application object

If you want to work with an existing application, see "Working with other application objects" on page 52.

❖ To create an application object:

1 Click the Application painter button in the PowerBar or PowerPanel. What you do next depends on whether you previously worked with an application object.

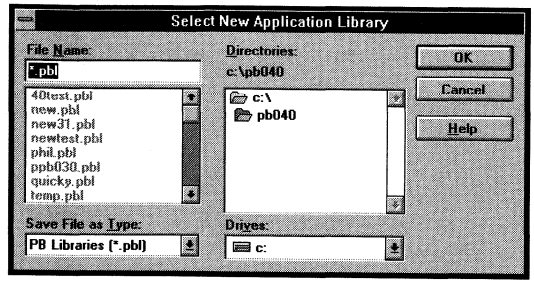
If you have never worked with an application object

The Select Application Library dialog box displays. You don't want to use an existing application, so click the Cancel button. Click New in the Application dialog box.

If you previously worked with an application object

The application object opens and displays in the workspace. To create a new application, select File➤New from the menu bar.

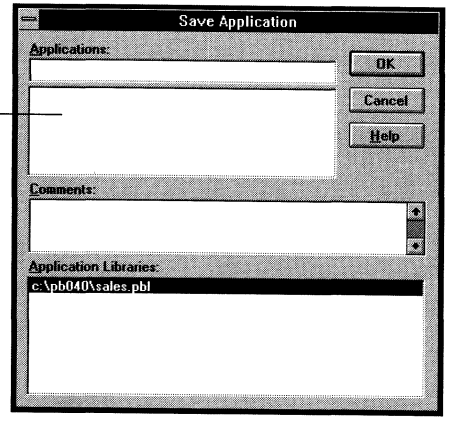
The Select New Application Library dialog box displays.



2 Specify the PowerBuilder library in which you want to store the application object. (The application itself can use multiple libraries. The library you are specifying here is the library in which to store the *application object*.) You can name a new library or specify an existing library.

The Save Application dialog box displays.

Lists other application objects in library



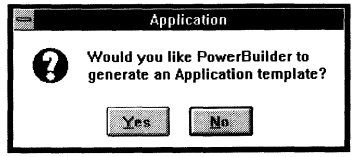
Putting more than one application object in a library
Any application objects in the specified library are listed in the second box in the Save Application dialog box. Standard practice is to have no more than one application object in a library, although you can have more (for ease of migrating applications to new versions of PowerBuilder, you should have only one application object in a library). If the library you have chosen already has an application object, you might want to click Cancel and choose another library.

3 Name the application object: enter a 1- to 40-character name for the application you are building. The application object name will display in the PowerBuilder title bar while you are working on the application.

4 (Optional) In the Comments box, enter comments to document the application you are building and to help other developers understand the application. The comments display in the Library painter.

5 Click OK.

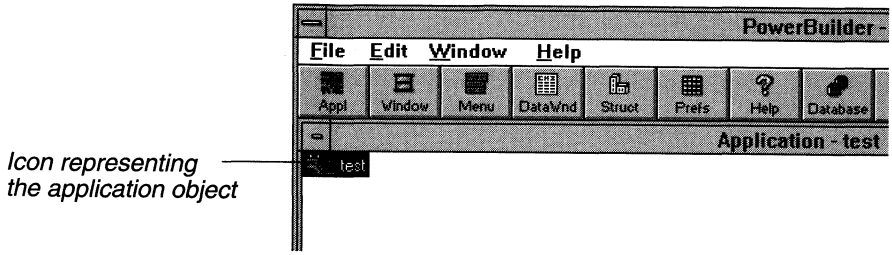
The Application dialog box displays, asking whether you want to build a template for your application.



For more information, see "Using the Quick Application feature" on page 54.

6 Click Yes to generate a template for your application. Click No to start your application from scratch.

You go to the Application painter workspace. The new application object is displayed as an icon in the workspace.



You have now created the application object and placed it in a PowerBuilder library.

For information about defining the application object's properties, see "Specifying application properties" on page 60.

Working with other application objects

When you open the Application painter, PowerBuilder opens and displays the application object you worked on last. You can switch to work with another application object (and thus work on another application) anytime.

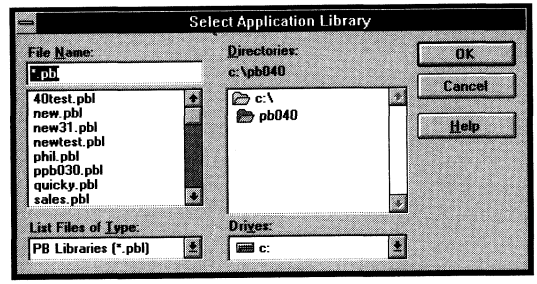
❖ To work with another application object:

1 Open the Application painter.

The last application object you worked with opens.

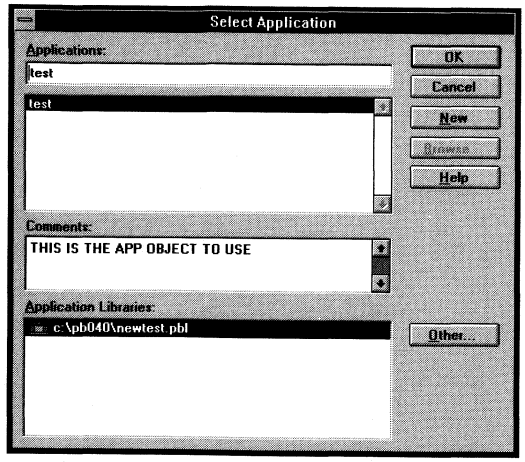
2 Click the Open button.
or
Select File➤Open from the menu bar.

The Select Application Library dialog box displays.



3 Select the library containing the application object and click OK.

The Select Application dialog box displays the application objects in the specified PowerBuilder library.

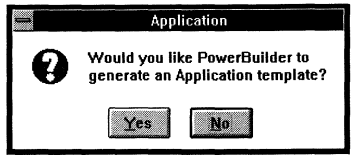


4 Choose the application object from the list. You can choose an application object from a different PowerBuilder library by clicking the Other button.

The chosen application object opens and displays in the workspace.

Using the Quick Application feature

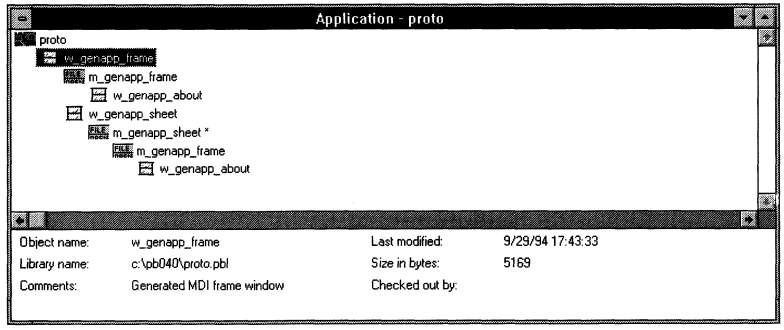
When you are creating a new application object, PowerBuilder offers to create an application template for you.



You can use this template to begin your application, instead of having to define all of your objects from scratch.

If you click Yes, PowerBuilder generates the shell of a basic MDI application that includes an MDI frame (complete with window functions that do such things as open or close a sheet), a sheet, an About dialog box, menus, toolbars, and scripts.

The Application painter workspace shows the objects in the application, as described in the next section.



You can run the application immediately by clicking the Run button on the PainterBar. You can open sheets, display an About box, and select items from menus.

You can use this application as a starting point for your MDI application.

For more information about building MDI applications, see *Building Applications.*

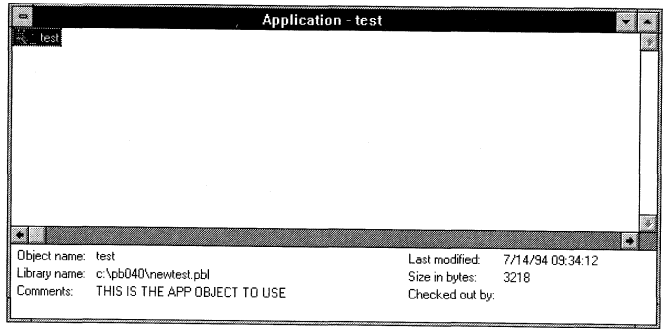
Looking at an application's structure

Once you have selected an application object, it displays as an icon in the Application painter workspace. 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 Application painter.

❖ To display the application's structure:

1 Open the Application painter and select the application object you want to work with.

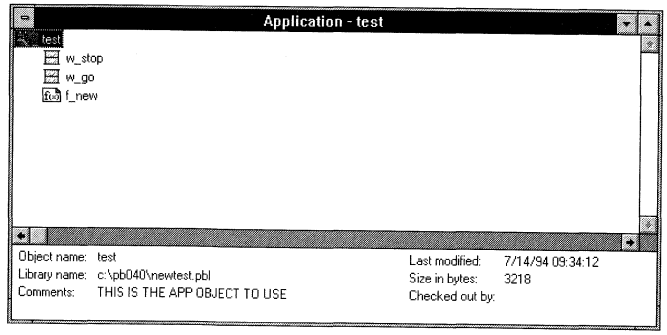
The application object displays as an icon in the workspace. Information about the object displays in the bottom of the window.



2 Double-click the icon.

PowerBuilder expands the display to show all the global objects that are referenced in a script for the application object.

For complete information about exactly which objects are shown in the workspace, see "Which objects are displayed" on page 57.



3 Work with the objects in the workspace as described next.

Working in the workspace

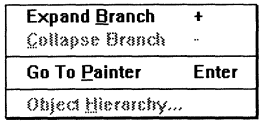
The Application painter workspace displays referenced objects in an outline format.

Using the popup menu

The Application painter workspace provides a popup menu that offers shortcuts to working with displayed objects in the workspace.

❖ **To use the popup menu:**

1 Position the mouse pointer over the application object or one of the displayed objects.



2 Click the right mouse button.

The popup menu displays.

3 Select one of the items on the menu.

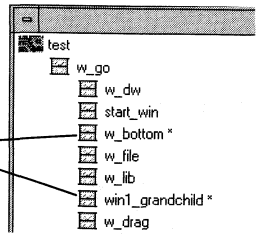
Changing the display of objects

To	Do this
Display the objects referenced by an object	Double-click the object, select Expand Branch from the popup menu, or press +
Hide the objects referenced by an object	Double-click an expanded object, select Collapse Branch from the popup menu, or press –
Go to a painter for an object	Press ENTER or select Go To Painter from the popup menu

Working with inherited objects

An asterisk following an object name indicates that the object is a descendant of another object.

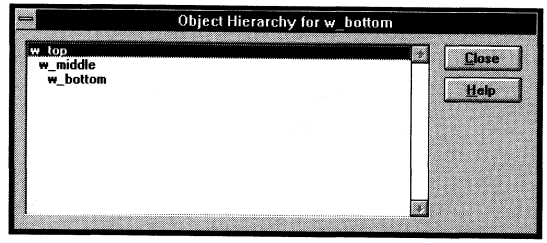
These windows are descended from other windows



❖ **To see the inheritance hierarchy for a descendent object:**

- Select Object Hierarchy from the popup menu.

  PowerBuilder displays a window showing the inheritance hierarchy for the selected object.



Which objects are displayed

The Application painter workspace 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.

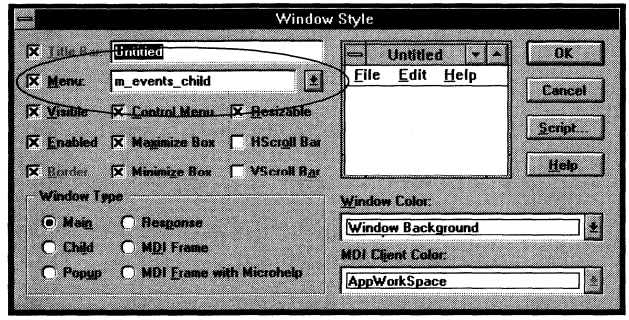
Which references are displayed

The workspace displays the following types of references when an object is expanded.

Objects that are referenced in painters

For example:

- 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 that are directly referenced in scripts

For example:

- If a window script contains the following statement:

```
Open(w_continue)
```

Then w_continue displays when the window is opened.

> **Which referenced windows display in the workspace**
> Windows are only considered referenced when they are *opened* from within a script. A use of another window's attribute or instance variable will not cause the Application painter 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(EnteredValue)
```

  Then f_calc displays when the menu is expanded.

- If a window uses a popup menu through the following statements:

```
m_new    mymenu
mymenu = create m_new
mymenu.m_file.PopMenu( PointerX(), PointerY() )
```

  Then m_new displays when the window is expanded.

Which references are not displayed

The workspace does not display the following types of references.

Objects that are referenced only through instance variables or attributes

For example:

- If w_go has this statement (and no other statement referencing w_emp):

```
w_emp.Title = "Managers"
```

  Then w_emp does not display as a reference for w_go.

Objects that are referenced dynamically through string variables

For example:

- If a window script has the following statements:

```
window   mywin
string   winname = "w_go"
Open(mywin,winname)
```

  Then the window w_go does not display when the window is expanded. The window w_go is named only in a string.

- If the DataWindow object d_emp is associated with a DataWindow control dynamically through the following statement:

```
dw_info.DataObject = "d_emp"
```

  Then d_emp does not display when the window containing the DataWindow control is expanded.

Specifying application properties

The application object specifies the following properties of the application:

- Default text attributes (font, style, size, and color)
- The library search path for the application
- The icon for the application
- Default global objects

You specify all these properties in the Application painter.

Delivering an executable
You can also use the Application painter to create an executable version of your application.

For more information, see Chapter 22, "Creating an Executable."

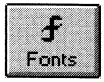
Specifying default text attributes

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

Type	Where used
Text	*Text* in this context refers to static text in windows, user objects, and DataWindow objects
Data	Data retrieved in a DataWindow
Heading	Column descriptions that display above column data in tabular and grid DataWindow objects
Label	Column descriptions that display next to column data in freeform DataWindow objects

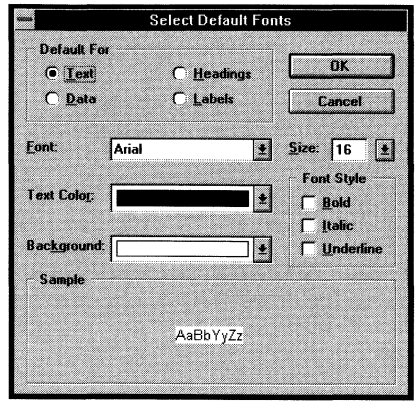
PowerBuilder provides default settings for the font, size, and style for each of these and a default color for text and the background. You can change these settings for an application in the Application painter and can override the settings for a window, user object, or DataWindow object.

❖ To change the text defaults for an application:



1 Click the Fonts button.
or
Select Edit➤Default Fonts from the menu bar.

The Select Default Fonts dialog box displays the current settings for the font, size, style, and color for Text.



The sample text at the bottom of the window illustrates the current settings.

2 Select Text, Data, Headings, or Labels in the Default For group.

3 Review the settings and make any necessary changes:

- To change the font, select a font from the list in the Font listbox.
- To change the size, select a size from the list in the Size listbox or type a valid size in the listbox.
- To change the style, click one or more Font Style checkboxes (Bold, Italic, and Underline).
- To change the text color, select a color from the Text Color listbox. (You don't specify colors for data, headings, and labels here. You do that in the DataWindow painter.)

> **Using custom colors**
> When specifying a text color, you can choose a custom color. You define custom colors in the Window painter or DataWindow painter.

4 When you have made all the changes, click OK.

Specifying the library search path

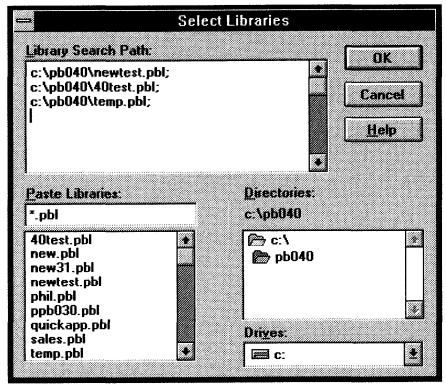
The objects you create in painters are stored in PowerBuilder libraries. You can use objects from one library or multiple libraries in an application. You define each library the application uses in the library search path.

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.

❖ To define a library search path:

1 Click the Library List button.
or
Select Edit➤Library List from the menu bar.

The Select Libraries dialog box displays the current library search path and lists the PowerBuilder libraries in the current directory.



2 Double-click the name of each library you want to include in the search path in the Paste Libraries listbox. Each time you double-click a library name, PowerBuilder appends it to the list in the Library Search Path listbox. You can include libraries from different drives and directories in the search path.

> **Deleting a library from the list**
> To delete a library from the search path, select the library from the list in the Library Search Path listbox and press DELETE.

3 Click OK.

PowerBuilder updates the search path for the application.

For more information

There are several strategies you can use to organize your application into libraries and optimize your environment and easily work with other developers on a large application.

The *Building Applications* manual describes these strategies in detail.

Specifying an icon

Users might 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 Click the Icon button.
or
Select Edit➤Application Icon from the menu bar.

The Select Icon dialog box displays.

2 Specify a file containing an button (an ICO file).

The button displays at the right of the Button Name box.

3 Click OK to associate the button with the application.

Specifying default global objects

PowerBuilder provides five built-in global objects that are predefined in all applications.

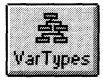
Global object	Description
SQLCA	Transaction object, used to communicate with your database
SQLDA	DynamicDescriptionArea, used in dynamic SQL
SQLSA	DynamicStagingArea, used in dynamic SQL
Error	Used to report errors during execution
Message	Used to process messages that are not PowerBuilder-defined events and pass parameters between windows

You can create your own versions of these objects by going to the User Object painter and defining a standard class user object that inherits 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 10, "Working with User Objects."

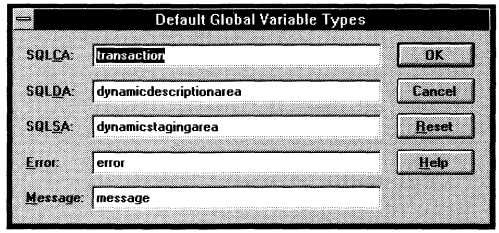
After you do this, in the Application painter you can tell PowerBuilder that you want to use your version of the object in your application as the default, instead of the built-in version.

❖ To specify the default global objects:



1 Click the Variable Types button.
or
Select Edit➤Default Global Variables from the menu bar.

The Default Global Variable Types dialog box 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.

Changing your mind
To restore the built-in versions of all the global objects, click Reset in the Default Global Variable Types dialog box.

Writing application-level scripts

When a PowerBuilder application is run, an Open event is triggered in the application object. You write a script for the Open event that specifies the processing that takes place when the application is opened. Typically this script opens the first window of the application and sets up the environment.

Batch applications
If your application performs only batch processing, all processing takes place in the script for the application Open event.

Here are all the events that can occur in the application object. The only event that requires a script is Open.

Event	Occurs when
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).
Idle	The Idle PowerScript function has been called and the specified number of seconds have elapsed with no mouse or keyboard activity.
Open	The user starts the application.
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 during execution" in Chapter 21, "Debugging and Running Applications."

Setting application attributes

The application object has several attributes, which specify application-level properties. For example, the attribute ToolbarText specifies whether text displays on toolbars in an MDI application.

You can reference these attributes in any script in the application using this syntax:

AppName.attribute

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 don't need to qualify the attribute name with the application name.)

For a complete list of the attributes of the application object, see *Objects and Controls*.

PART TWO

Coding Fundamentals

This part describes how to code your application. It covers the basics of the PowerScript language, how to use the PowerScript painter, and how to create functions and structures to make your code more powerful and easier to maintain.

CHAPTER 3

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 PowerScript painter to write scripts using the PowerScript language.

Contents

For more information

For complete information about the PowerScript language, see *PowerScript Language*. For complete information about its built-in functions, see the *Function Reference*.

The process of writing scripts

❖ **To create or modify a script:**

1 Open the PowerScript painter from the object that the script is for.

For example, if the script is for a window, open the Window painter, then open the PowerScript painter.

2 Code the script.

3 Compile the script and return to the previous painter.

Opening the PowerScript painter

You can open the PowerScript painter from the Application, Window, Menu, or User Object painter (because you can attach scripts only to application objects, windows and their controls, menus, and user objects).

❖ To open the PowerScript painter:

1. Select the object or control you want to write a script for. To write a script for a window (as opposed to a control in the window), make sure no control is selected.
2. Do one of the following:
   - Click the Script button in the PainterBar (see below).
   - Use the popup menu (see below).
   - Press CTRL+S.

Using the Script button

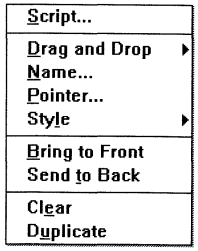
The Script button has two appearances:

An empty button indicates that the currently selected object or control has no scripts.

A button containing a text representation indicates that the current object or control has at least one script.

Using the popup menu

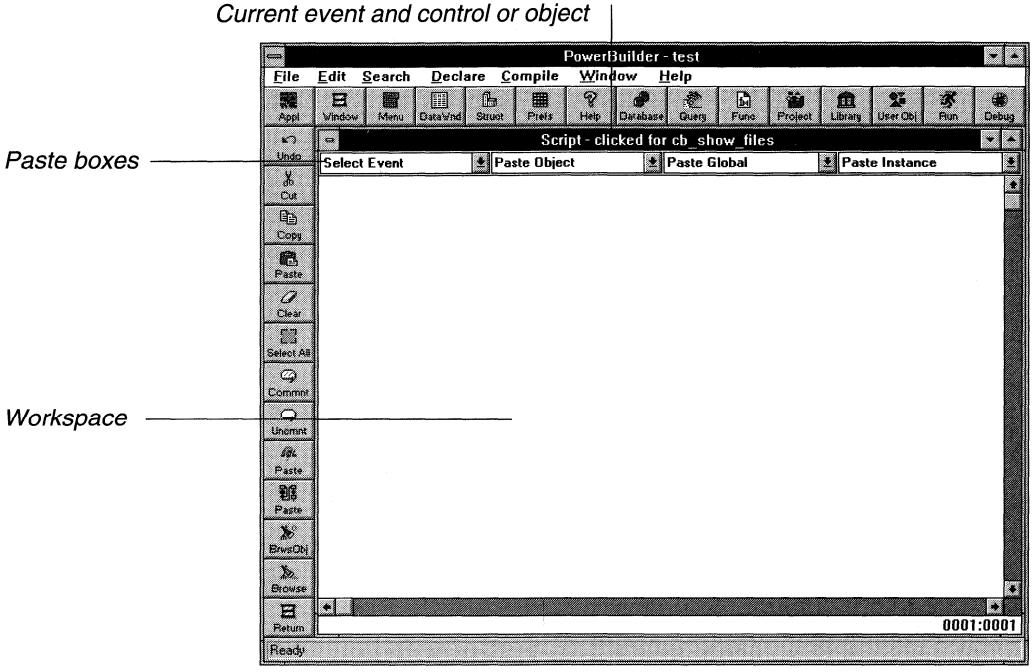
You can also select an object or control and open the PowerScript painter in one step:



- Place the mouse pointer on the object or control, display the popup menu, then choose Script from the menu.

What happens

The PowerScript painter opens in a separate window.



When you open the PowerScript painter, the following information displays:

- The name of the current event, which displays in the painter's title bar.

  If multiple events can occur in the object you are working with, the current event is the last event for which a script was displayed. If there is no script, the current event is an event that typically has a script.
- The name of the current object or control, also displayed in the title bar.
- A Select Event dropdown listbox, which lists events for the current object or control.
- A Paste Object dropdown listbox, which lists objects and controls whose names you can paste into the script.
- A Paste Global dropdown listbox, which lists global variables you can use in the script.

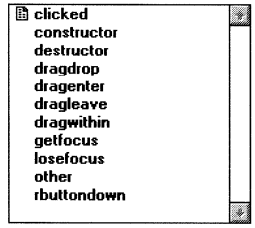
- A Paste Instance dropdown listbox, which lists instance variables you can use in the script.

  For more about instance variables, see *PowerScript Language*.

- The script for the current event. If there is no script, the workspace is blank.

Changing the current event

If the object or control has multiple events, you can change the current event by selecting an event from the list in the Select Event dropdown listbox.



Seeing which events have scripts

The PowerScript painter indicates which events have scripts, as follows:

- If the event has a script written for the object or control you are working with, a script icon appears next to the event.

- If the event has a script in an ancestor object or control only, the script icon in the Select Event dropdown listbox is displayed in color (shown in gray here).

- If the event has a script in an ancestor as well as in the object or control you are working with, the icon is displayed half in color.

Working in the painter

You write scripts in the PowerScript painter. It provides all the features needed for writing and modifying scripts. For example, you can cut, copy, and paste text, as well as search for and replace text.

The painter also provides many features that make it easy to use the PowerScript language, such as facilities for pasting information into scripts.

Changing the font
You can change the font that is used in the PowerScript painter.

For more information, see Chapter 1, "The World of PowerBuilder."

Using the PainterBar

Like the other painters, the PowerScript painter has a PainterBar that provides a shortcut for performing frequently used activities. There is also a menu item corresponding to each activity:

Button	Activity	Menu item
	Undoes the most recent edit	Edit➢Undo
	Cuts selected text to the clipboard	Edit➢Cut
	Copies selected text to the clipboard	Edit➢Copy
	Pastes the contents of the clipboard at the current cursor location; replaces any selected text	Edit➢Paste
	Deletes selected text; does not place the text in the clipboard	Edit➢Clear

Button	Activity	Menu item
	Selects all text in the workspace	Edit➤Select All
	Comments out current line or all lines containing selected text	Edit➤Comment Selection
	Uncomments current line or all lines containing selected text	Edit➤Uncomment Selection
	Pastes a SQL statement at current cursor location (see page 89)	Edit➤Paste SQL
	Pastes a PowerScript statement at current cursor location (see page 88)	Edit➤Paste Statement
	Opens the Quick browser (see page 80)	Edit➤Browse Object
	Opens the Object browser (see page 81)	Edit➤Browse Objects
(various)	Returns to the painter from which you opened the PowerScript painter; for example, if you are defining a script for a window control, this button shows a window and when clicked returns to the Window painter	File➤Return

Manipulating text

You can perform standard editing tasks in the PowerScript painter.

Cutting, copying, and pasting text

You can use the PainterBar or items on the Edit menu to cut, copy, and paste text using the clipboard. You can also use shortcut keys.

Operation	Shortcut key
Cut	CTRL+X
Copy	CTRL+C
Paste	CTRL+V

Undoing changes
You can undo the most recent editing change by selecting Edit➤Undo or pressing CTRL+Z.

Copying code
If you have code in one script that you want to use in another script, you can copy it to the clipboard, then paste it in the second script.

Searching and replacing text

You can choose the items from the Search menu to find specific text in the script and optionally replace the text.

Commenting out text

❖ **To designate one or more lines as comments:**

1 Place the cursor anywhere in the line you want to comment out. To comment out more than one line, select at least one character in each line.

2 Click the Comment Selection button in the PainterBar.
or
Edit➤Select Comment Selection from the menu bar.

PowerBuilder inserts two slashes before the first character in each line.

❖ To uncomment a block of code:

1 Place the cursor anywhere in the line you want to uncomment. To uncomment more than one line, select at least one character in each line.

2 Click the Uncomment Selection button in the PainterBar.
or
Select Edit➤Uncomment Selection from the menu bar.

PowerBuilder removes the two slashes before the first character in each line.

Printing your script

You can print the current script on the default printer by selecting Print from the File menu. To change the printer or its settings, select File➤Printer Setup from the menu bar before printing.

Getting context-sensitive Help

In addition to accessing Help through the Help menu and F1 key, you can use context-sensitive Help in the PowerScript painter 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.

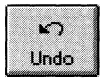
The Help window displays information about the reserved word or function.

Copying Help text
You can copy text from the Help window into the PowerScript painter. This is an easy way to get information about parameters required by the built-in functions.

Pasting information

Scripts frequently reference objects and controls or the names of variables and built-in functions. To quickly access these entities, you can paste information directly into scripts:

Use	To paste
Paste listboxes above workspace	Objects, controls, and variables
Quick browser	Attributes and functions
Object browser	Attributes, data types, functions, structures, variables, and objects
OLE Class browser	Names of OLE classes
Edit➤Paste	Contents of clipboard
Paste Statement button or Edit➤Paste Statement	PowerScript statements
Paste SQL button or Edit➤Paste SQL	SQL statements
Edit➤Paste Function	Built-in, user-defined, and external functions
File➤Import	Contents of text files



Undoing a paste
If you paste information into your script by mistake, immediately click the Undo button or select Edit➤Undo from the menu bar to delete the paste.

Using the Paste listboxes

You can use the Paste dropdown listboxes just above the painter workspace to paste the name of an object, control, or variable that is currently available. The listboxes are accessible using the mouse or the keyboard.

❖ To use the Paste listboxes using the mouse:

1 Move the cursor where you want to paste the object, control, or variable.

2 Click the Paste dropdown listbox for the type of information you want to paste. (See below for information about each of the listboxes.)

A list of available objects and controls or variables displays.

3 Click the entity you want to paste.

PowerBuilder closes the list and pastes the selected object, control, or variable at the insertion point in the script.

❖ To use the Paste listboxes using the keyboard:

1 Move the cursor where you want to paste the object, control, or variable.

2 Press CTRL+*number* to drop down the listbox (CTRL+1 drops down the Select Event listbox, CTRL+2 drops down the Paste Object listbox, and so on).

3 Use the arrow keys to select the entity.

4 Press ENTER.

If you accessed the Select Event listbox, you go to the script for the selected event. If you accessed one of the paste listboxes, the selected entity is pasted in the script.

The Paste Object listbox

What appears in this listbox depends on which painter you opened the PowerScript painter from:

Painter	What painter displays
Application	All windows defined for the application (all windows in the library search path for the application object)
Window	All controls contained in the window, along with the name of the window itself
Menu	All MenuItems in the current menu
User Object	All controls and user objects contained in the user object, along with the name of the user object itself

For example, if you are writing a script for a window control, you can use the Paste Object listbox to paste the name of another control in the current window.

The Paste Global listbox

This listbox displays all global variables defined for the application. (To define additional global variables, choose Declare➤Global Variables from the menu bar.)

The Paste Instance listbox

This listbox displays all instance variables defined for the corresponding application object, window, menu, or user object, or one of its ancestors. (To define instance variables, choose Declare➤Instance Variables from the menu bar.)

Shared variables are not displayed in the listbox. To access them, choose Declare➤Shared Variables from the menu bar or use the Object browser.

Using the Quick browser

You can use the Quick browser to paste the name of any attribute or the prototype for any object-level function for the object or control you are currently writing a script for.

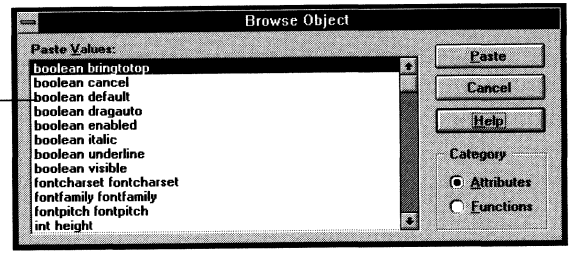
❖ **To use the Quick browser:**

1 Move the cursor to where you want to paste the information. Select any text you want to replace by the pasting.

2 Click the Browse Object button.
or
Press CTRL+O.
or
Select Edit➤Browse Objects.

The Quick browser displays.

Displays attributes or functions of current object or control



3 Click the Attributes or Functions radio button, depending on whether you want to paste an attribute or function prototype.

4 Double-click the name you want to paste.
or
Highlight it and click the Paste button.

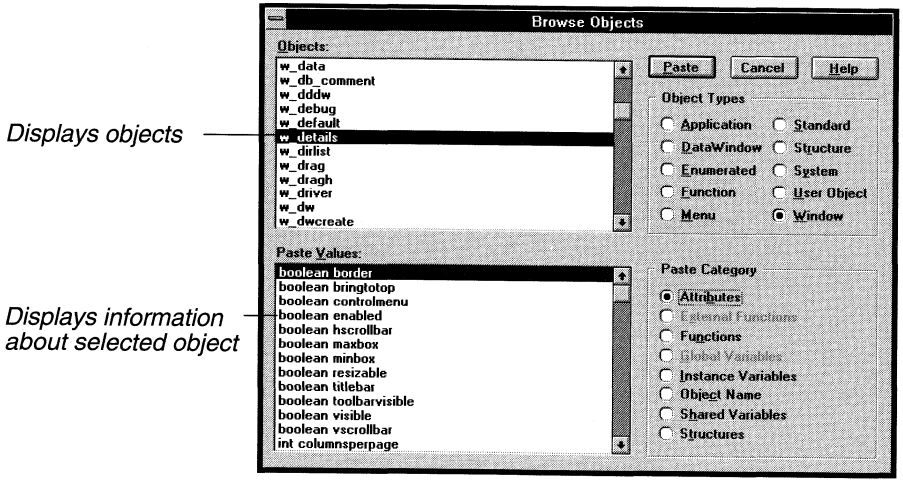
The name of the attribute or the function prototype is pasted into the script.

Using the Object browser

You can use the Object browser to paste the name of any attribute, data type, function, structure, variable, or object in the application.

About the Object browser

The Object browser has two parts:



The top part displays one type of object, such as window or menu. The bottom part displays some aspect of the selected object, such as attributes of the selected window.

❖ To use the Object browser:

1 Move the cursor where you want to paste the information. Select any text you want replaced by the pasting.

2 Click the Browse Objects button, press CTRL+B, or select Browse Objects from the Edit menu.

The Object browser displays.

3 Choose the type of object you want to display by clicking a radio button in the Object Types box.

For more information, see "Choosing the object type and paste category" below.

4 Select the entry in the Objects box that you want information about.

5 Select the category of information you want to display by clicking a radio button in the Paste Category box (see below).

Information for the selected category displays in the Paste Values box.

6 Select the information you want to paste.

7 Double-click the information you want.
or
Select the information and click Paste.

PowerBuilder closes the Object browser and displays the information at the insertion point in the script, replacing any selected text.

Opening the browser from the Library painter or PowerBar
You can also open the Object browser from the Library painter: Select Utilities➤Browse Objects from the menu bar.

You can also open the Object browser from the PowerBar by customizing the PowerBar to add the Browse Objects button.

For more about customizing toolbars, see Chapter 1, "The World of PowerBuilder."

Choosing the object type and paste category

When you open the Object browser, it displays all objects of the current type, with the current object selected. For example, if you open the Object browser from a script associated with a window, all windows in the current application display, with the current window selected.

Use the following table to decide which object type and paste category to use to paste a particular kind of information into a script:

To paste	Select this Object Type	Select this Paste Category
Object name	The corresponding object type	Object Name
Attributes for an object	The corresponding object type	Attribute
Names of controls in a window or user object (see below)	Window or User Object	Object Name
Attributes of controls (see below)	Window or User Object	Attributes
Data types	Standard or Enumerated	Object Name
Object-level built-in functions	System, then select the type of object in the Objects box	Functions
Built-in functions not related to an object type	System, then select SystemFunctions in the Objects box	Functions
Built-in functions associated with a particular window, menu, or user object	Window, Menu, or User Object	Functions
User-defined global functions	Function	Functions
User-defined functions associated with a window, menu, or user object	Window, Menu, or User Object	Functions
External functions	Application	Functions
Variables in global structures	Structure	Attributes
Variables in structures associated with a window, menu, or user object	Window, Menu, or User Object	Structures

To paste	Select this Object Type	Select this Paste Category
Global variables (built-in and user-defined)	Application	Global Variables
Instance or shared variables associated with a window, menu or application object	Window, Menu, or Application	Instance or Shared Variables

Displaying controls and MenuItems

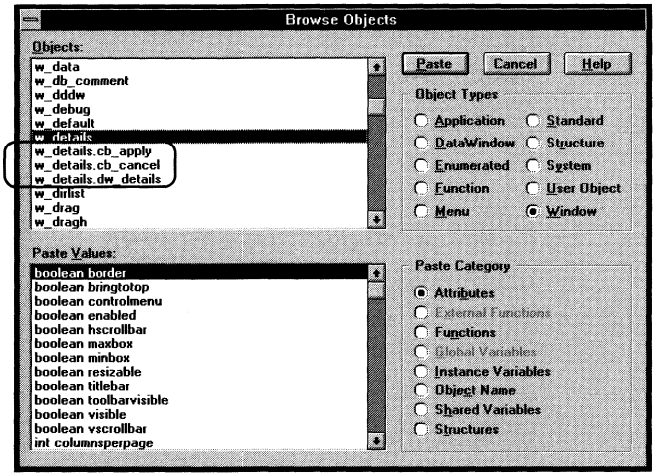
To paste a control or MenuItem in a script, you need to select controls or MenuItems in the Objects box.

❖ To select a control in the Objects box:

1 List all windows or user objects in the Objects box by clicking the Window or User Object button.

2 Double-click the window or user object containing the control you are interested in.

The Object browser lists all controls just below the object name.



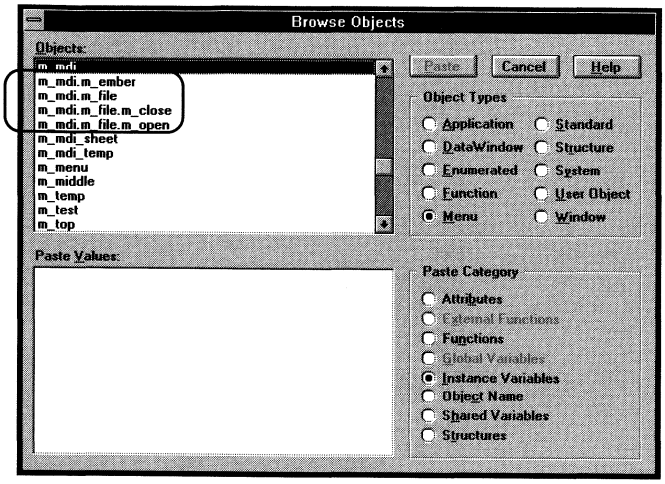
3 Select the control you want.

❖ To select a MenuItem in the Objects box:

1 List all menus in the Objects box by clicking the Menu button.

2 Double-click the menu containing the object you are interested in.

The Object browser lists all MenuItems in that menu.



3 Select the MenuItem you want.

About system objects

When you click System in the Object Types box, PowerBuilder displays all system objects. System objects include the *types* of objects you define and store in a library, such as windows and menus, as well as the types of controls and other predefined entities that you can reference in your application, such as the Message object and Error object.

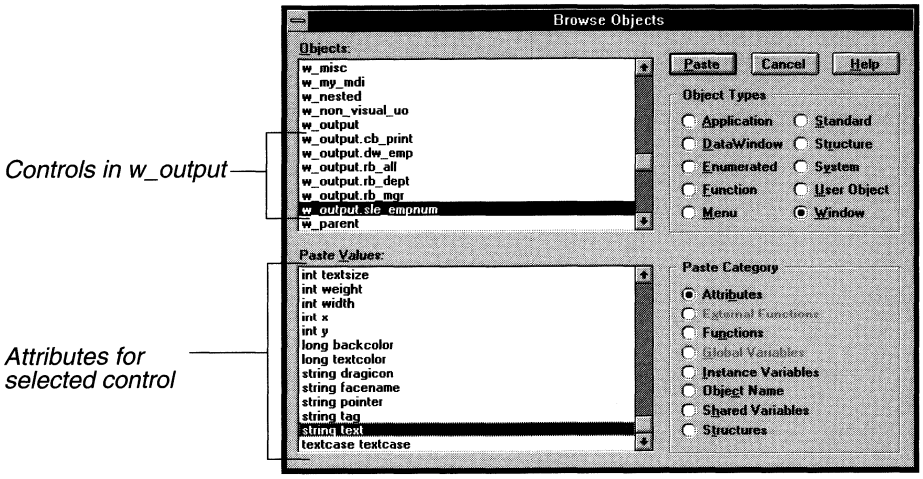
For more about the predefined system objects, see *Objects and Controls*.

Examples of using the Object browser

This section describes several common situations where the Object browser is useful.

Situation You are writing a script for a CommandButton in the window w_output. As part of the processing, you want to change the text displayed in a SingleLineEdit control in the window.

Solution You open the Object browser, which displays the windows with w_output displayed. Double-click the window to display the list of controls. Select the SingleLineEdit you want and click the Attributes button in the Paste Category box. Select Text from the list and click Paste to paste the information into the script.



PowerBuilder pastes in the unqualified name of the control, along with the attribute name.

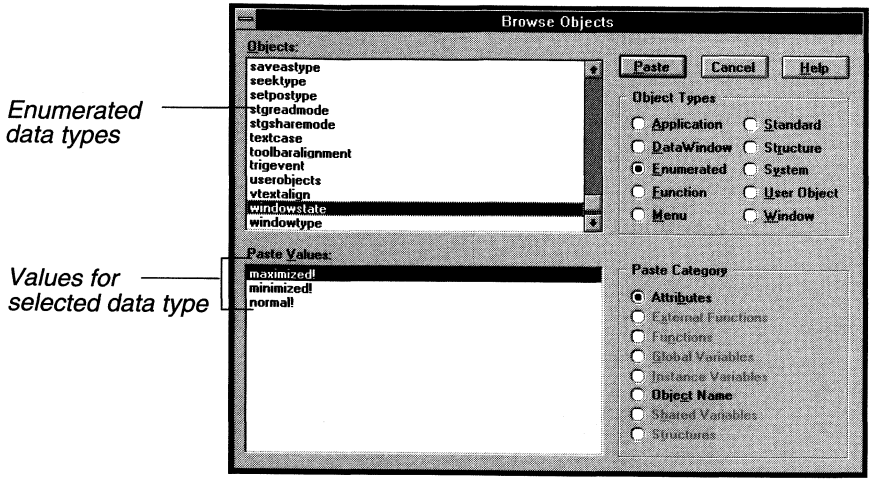
Situation You are writing a script for a MenuItem and want to refer to a control in the window you will attach the menu to.

Solution Click Window in the Object Types box. Double-click the name of the window you are interested in; the browser displays the window's controls. Select the control and click Attributes as the Paste Category. Select the attribute you want to reference and click Paste.

PowerBuilder pastes in the *qualified* name of the window control, along with the attribute name. (PowerBuilder knows that you need to use a fully qualified name of a window and its control in a menu.)

Situation You want to modify a window's state but can't remember what the enumerated values for window state are.

Solution Click Enumerated in the Object Types box. PowerBuilder displays all enumerated data types. Select WindowState. PowerBuilder displays the values for WindowState in the Paste Values box. Select the value you want and click Paste.



Situation You want to manipulate a DropDownListBox in the current window but forget the name of the built-in function to use.

Solution Click Window in the Object Types box. Double-click the current window to display its controls. Select the DropDownListBox you want to manipulate. Click Functions in Paste Category to display all functions associated with DropDownListBoxes. Select the function you want and click Paste.

PowerBuilder pastes the function, along with placeholders for the parameters.

Situation You have defined a window-level function and want to use it in a script for a control for that window.

Solution Select the window in the Objects box and click Functions as the Paste Category. PowerBuilder displays all functions associated with the window (all built-in functions that apply to all windows plus any user-defined functions for that window). Select the user-defined function you want and click Paste.

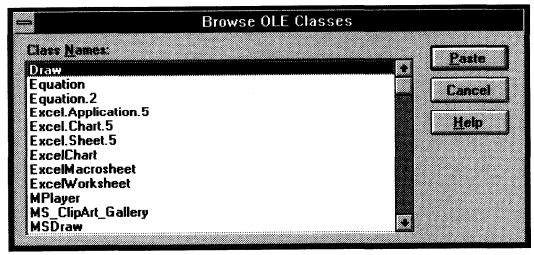
Using the OLE Class browser

You can use the OLE Class browser to paste the name of any OLE class defined on your computer into a script. This is useful when you are incorporating OLE 2.0 into your application.

❖ To use the OLE Class browser:

1 Select Edit➤Browse OLE Classes from the menu bar.

The OLE Class browser displays, listing all OLE classes defined on your computer.



2 Select a class and click Paste to paste the name into your script.

For information about using OLE in a PowerBuilder application, see *Building Applications*.

Pasting statements

You can paste a template for the following PowerScript statements:

- IF...THEN
- DO...LOOP
- FOR...NEXT
- CHOOSE CASE

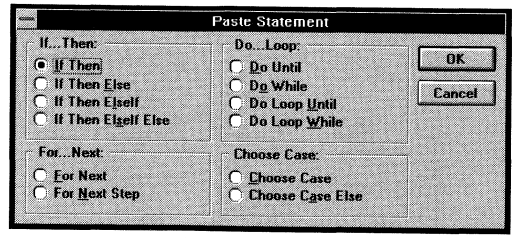
When you paste these statements into a script, prototype values display in the syntax to indicate conditions or actions.

❖ To paste a PowerScript statement into the script:

1 Move the cursor where you want to paste the function.

2 Select the Paste Statement button from the PainterBar or choose Paste Statement from the Edit menu.

The Paste Statement dialog box displays.



3 Select the statement you want to paste into the script and click OK.

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 *PowerScript Language*.

Pasting SQL

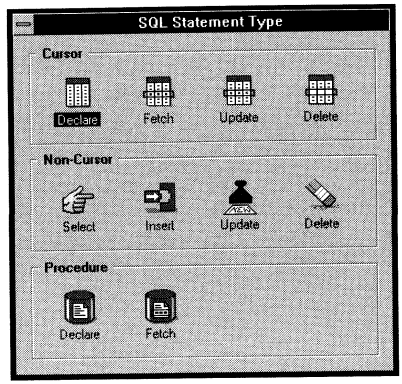
You can paste a SQL statement into your script instead of typing the statement.

❖ To paste a SQL statement:

1 Place the insertion point where you want to execute the SQL statement in the script.

2 Click the Paste SQL button in the PainterBar.
or
Press CTRL+Q.
or
Select Edit➤Paste SQL from the menu bar.

The SQL Statement Type dialog box displays.



3 Select the type of statement you want to insert by double-clicking the appropriate button.

The appropriate dialog box displays so you can paint the SQL statement.

4 Paint the statement, then return to the PowerScript painter.

The statement displays at the insertion point in the workspace.

For more about embedding SQL in scripts, see *PowerScript Language*.

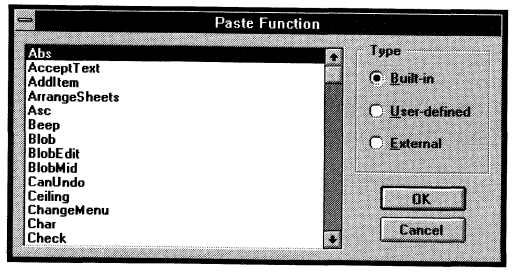
Pasting functions

You can paste the name of any function into a script.

❖ To paste a function name into a script:

1 Select Edit➤Paste Function from the menu bar.

The Paste Function dialog box displays.



2 Choose the type of function you want to paste: built-in, user-defined, or external.

3 Double-click the function you want.

PowerBuilder pastes the name of the function into the script and places the cursor within the parentheses for you to define any needed arguments.

Pasting contents of files

You can import the contents of an external text file into the PowerScript painter or export the contents of the script to a text file.

Importing from a file

If you have code that is common across different scripts, you can keep that code in a text file, then read it into new scripts you write.

❖ To import the contents of a file into the PowerScript painter:

1 Place the insertion point where you want the file contents pasted.

2 Select File➤Import from the menu bar.

The File Import dialog box displays, listing all files with the extension SCR.

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 PowerScript painter at the insertion point.

Exporting a script to a file

You might want to save all or part of a script to an external text file for use later. This is useful when:

- You want to use the code in another script
- You are about to make a major change to a script and want to make sure you have a backup file with the current script in case you need it

❖ To save the contents of a script to a text file:

1 If you want to save the entire script, make sure no text is selected. If you want to save only part of the script, select the part you want to save.

2 Select File➢Export from the menu bar.

The File Export dialog box displays.

3 Name the file and click OK.

The default file extension is SCR.

Compiling the script

Before you can execute a script, you must compile it. There are two ways to do it.

❖ To compile the script and remain in the PowerScript painter:

- Select Compile➤Script from the menu bar.
  or
  Press CTRL+L.

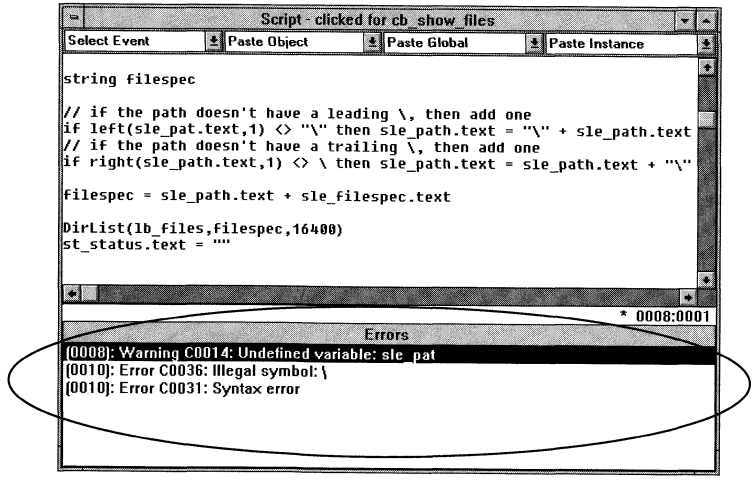
  PowerBuilder compiles the script and reports any problems it finds, as described below in "Handling problems."

❖ To compile the script and immediately return to the painter you came from:

- Click the Return button, as described in "Leaving the PowerScript painter" on page 96.

Handling problems

If problems occur when a script is compiled, PowerBuilder displays messages in a window below the script.



There are two kinds of messages:

- Errors
- Warnings

Understanding errors

Errors indicate serious problems that you must fix before a script will compile and before you can leave the PowerScript painter. 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. Warnings are shown in the Message window as:

line number: Warning *warning number* : *message*

There are two 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 and return to the painter from which you opened the PowerScript painter. But you must fix the problem in the script before you can save the object, such as the window or menu.

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 (perhaps you are accessing two databases in the script, but you can only be connected to one at a time in PowerBuilder).

PowerBuilder lets you compile scripts with database warnings and also lets you save the associated object. It does this because it can't know whether the problem will apply during execution (because 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 during execution.

Displaying warnings

Use the Display Compiler Warnings and Display Database Warnings items on the Compile menu to specify whether compiler and/or database warning messages display when you compile. The default is to display both compiler and database warning messages. (Error messages always display.)

Fixing problems

To fix a problem, click the message. The PowerScript painter scrolls the script window to display the statement that caused the message. After you fix all the problems, compile the script again.

To save a script with errors
Comment out the lines containing errors.

Leaving the PowerScript painter

❖ To leave the PowerScript painter:

- Click the last button in the PainterBar (the Return button).

  The Return button pictures the painter that you came from. For example, if you are writing a script for a control in a window, a window is pictured in the Return button.

  When you click the Return button, PowerBuilder compiles the script. If there are problems, PowerBuilder reports them, as described on page 93.

  If there are no problems, the PowerScript painter closes, the script is saved temporarily in a buffer, and you return to the painter from which you opened the PowerScript painter.

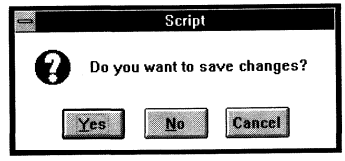
When you save the object (such as the window containing a control you wrote a script for), PowerBuilder recompiles all scripts in the object to make sure they are still valid (for example, that all objects that were referenced when you wrote the script still exist).

Leaving without saving your work

❖ To leave the PowerScript painter without saving your changes:

1 Select File➢Close from the menu bar.

You are asked whether you want to save your changes.



2 Click No to discard your changes.

PowerBuilder returns you to the painter from which you opened the PowerScript painter, without compiling or saving the modified script.

CHAPTER 4

Working with User-Defined Functions

About this chapter

This chapter describes how to build and use user-defined functions.

Contents

What are user-defined functions?

The PowerScript language has many built-in functions. But you may find that you need to code the same process over and over again. For example, you might need to perform a certain calculation in several places in an application or in different applications. In these situations, you will want to create a **user-defined function** to perform the processing.

A user-defined function is a collection of PowerScript statements that perform some processing. You use the Function painter to define user-defined functions. After you define a user-defined function and save it in a library, any application accessing that library can use the function.

Two kinds

There are two kinds of user-defined functions:

- **Global functions,** which are not associated with any object in your application and are always accessible anywhere in the application.

  These correspond to the PowerBuilder built-in functions that are not associated with an object, such as the mathematical and string-handling functions.

- **Object-level functions**, which are defined for a particular type of window, menu, or user object, or for the application object. These functions are part of the object's definition and can always be used in scripts for the object itself. You can also choose to make these functions accessible to other scripts as well.

  These functions correspond to built-in functions that are defined for windows, menus, user objects, or the application object.

Deciding which kind you want

When you design your application, 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; it is just that the function acts on a particular object type).

For example, say 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. 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, which performs processing unique to it.

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.

Defining user-defined functions

❖ To define a user-defined function:

1 Open the Function painter.

2 Name the function.

3 Define a return type.

4 For object-level functions, define an access level.

5 Define arguments for the function.

6 Code the function.

Each of these steps is described below.

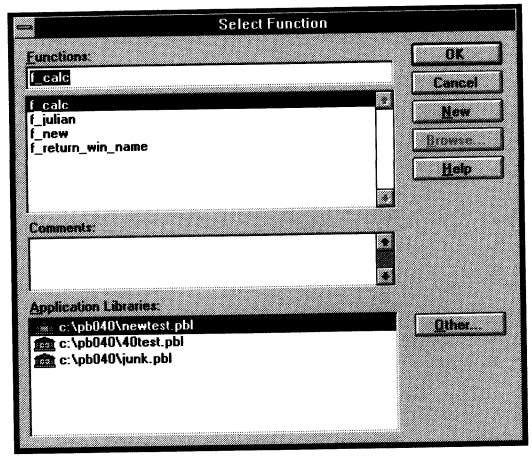
Opening the Function painter

How you open the Function painter depends on whether you are defining a global function or an object-level function.

❖ To open the Function painter for a global function:

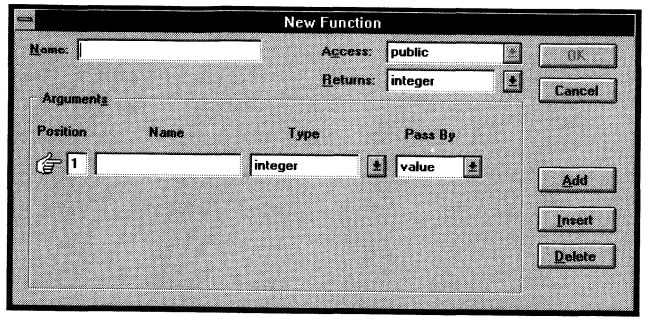
1 Click the Function painter button in the PowerBar or PowerPanel.

The Select Function dialog box displays.



2 Click the New button to declare a new function.

The New Function dialog box displays.



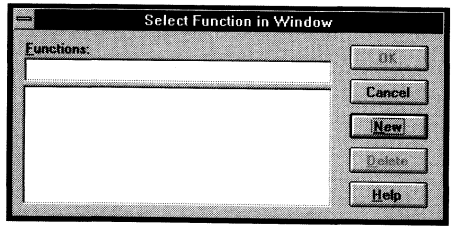
❖ To open the Function painter for an object-level function:

1 Open the painter for the object you want to define a function for.

For example, if you want to define a function for the window w_emp, open the Window painter and select w_emp.

2 From the Declare menu, select the appropriate menu item: Window Functions, Menu Functions, User Object Functions, or Application Functions. You can use the Declare menu in the Window, Menu, User Object, or PowerScript painter.

A Select Function dialog box displays.



Here you can select a function for the current object or define a new function for that object.

3 To define a new function, click the New button.

The New Function dialog box displays.

Naming the function

Function names can have up to 40 characters.

For valid characters, see *PowerScript Language*.

It is a good idea to develop a naming convention for user-defined functions so that you can recognize them easily and distinguish them from built-in PowerScript functions.

For example, you could preface all function names with f_ and use the underscore character to delimit parts of the name, such as:

```
f_calc
f_get_result
```

Since built-in functions do not have underscores in their names, this convention makes it easy for you to identify functions as user-defined.

Naming object-level functions

You might also want to adopt a naming convention to distinguish global functions from object-level functions. Doing this makes it easier for you to identify functions. Here is one suggestion:

Type of function	Name prefix
Global	f_
Window level	wf_
Menu level	mf_
User-object level	uf_
Application-object level	af_

Providing online Help for functions

If you want to provide context-sensitive online Help for your user-defined functions to other PowerBuilder developers, you should always use the same prefix when naming functions.

For more information, see *Building Applications*.

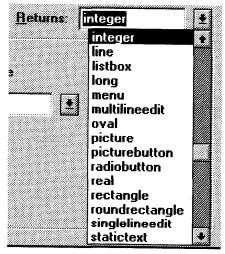
Defining a return type

Many functions perform some processing, 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 data type of the returned value.

Later, you will code a RETURN statement in the function that specifies the value to return (see "Returning a value" on page 108). 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 Returns dropdown listbox in the New Function dialog box.



You can specify any PowerBuilder data type, 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 as the return type any object types 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 Returns listbox. (You cannot select w_calculator from the list, since the list only shows built-in data types.)

❖ To indicate that a function does not return a value:

- Select (None) from the Returns list.

  This tells PowerBuilder that the function does not return a value (this is similar to defining a procedure in some programming languages).

Examples of functions returning values

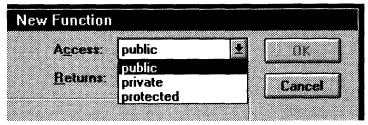
Say you are defining a mathematical function that does some processing and returns a real number. Define the function's return type as real.

Say you are defining a function that takes a string as an argument and returns the string in reverse order. Define the function's return type as string.

Say you are defining 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. Define the return type as w_calculator.

Defining the access level

The New Function dialog box has a listbox named Access.



This box specifies the **access level** of the function—that is, it specifies 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**. So when you are defining a global function, you cannot modify Access; the box is read-only.

For object-level functions

You can restrict access to an object-level function by setting its access level, as follows:

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 its descendants.

If a function is only to be used internally within an object, you should define its access as private or protected. That way, you ensure 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.)

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.

> **If the function takes no arguments**
> Leave blank the initial argument shown in the New Function dialog box and click OK to go to the Function painter workspace.

❖ To define arguments:

1 Name the argument.

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 data type, including:

- Built-in data types, such as integer and real
- Object types such as window or specific objects such as w_emp
- User objects
- Controls such as CommandButtons

> **Array arguments**
> You can also declare that an argument is an array. When you pass an array, you must include the square brackets in the array definition (for example, price[ ] or price[50]), and the data type of the array must be the data type of the argument.
>
> For information on arrays, see *PowerScript Language*.

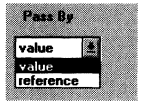
3 Declare how you want the argument passed (see "Passing arguments" next).

4 If you want to add another argument, click the Add button and repeat steps 1–3.

5 Click OK.

Passing arguments

In user-defined functions, you can pass arguments by reference or by value. You specify this property for each argument in the Pass By listbox.

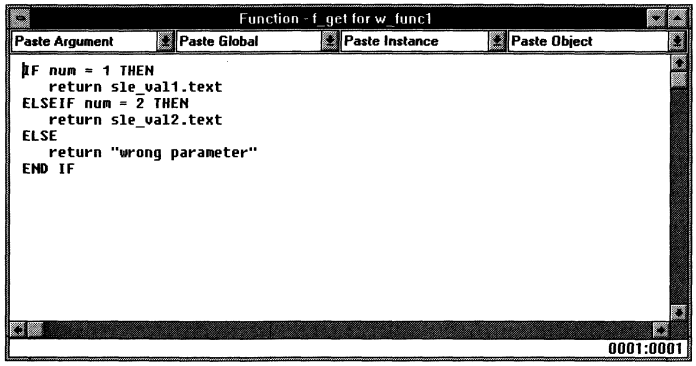


By reference When you pass an argument by reference, the function has access to the original argument and can change it directly.

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.

Coding the function

The Function painter workspace displays after you finish declaring the function's name, return type, access level, and arguments.



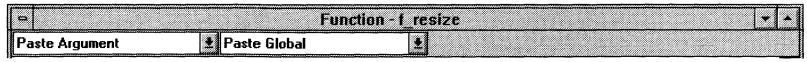
Using the workspace

The Function painter workspace is functionally the same as the PowerScript painter. In it you specify the code for the function, just as you specify the script for an event in the PowerScript painter.

For information about using the workspace, see Chapter 3, "Writing Scripts."

Workspace for global functions

When you declare a global function, the Function painter workspace has a title and two dropdown listboxes:



- The title is the name of the function that you are building or modifying.
- The Paste Argument dropdown listbox lists the arguments defined for the function.
- The Paste Global dropdown listbox lists the global variables defined in the application.

You can paste arguments and variables from the lists into the statements in the function you are building.

Workspace for object-level functions

When you declare an object-level function, the workspace displays the name of the current object in the title and four listboxes.

The first two listboxes are the same as in the workspace for global functions. The other two are Paste Instance and Paste Object. You can use these listboxes to paste names of instance variables, objects, and controls in the current object.

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 workspace or use the buttons in the PainterBar or items on the Edit menu to insert them into the function.

Returning a value

If you defined a return type for your function, you must return a value in the body of the function.

Syntax

To return a value in a function, use the RETURN statement:

RETURN *expression*

where *expression* is the value that you want returned by the function. The data type of the expression must be the data type that 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
```

Using text files

Different functions often use statements that are very similar or even identical. In these cases, you can save time and typing by using all or part of the existing function to build the new function. One way to do this is to save the function in a text file.

❖ To save a function in a text file:

1. If you want to save the entire function, make sure no text is selected. If you want to save only part of the function, select the part you want to save.
2. Select File➤Export from the menu bar.

   The File Export dialog box displays.
3. Name the file. By default, the file is given the extension FUN (for function).

❖ To read a text file into the workspace:

1. Place the insertion point where you want the contents of the file to appear.
2. Select File➤Import from the menu bar.

The File Import dialog box displays.

3 Choose the file and click OK.

PowerBuilder copies the contents of the file into the workspace at the cursor location.

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 can either compile the function without leaving the painter or compile the function in the process of saving it.

Compiling without leaving the painter

You can compile your function anytime while remaining in the Function painter to make sure what you have so far is syntactically correct.

❖ To compile without leaving the Function painter:

- Select Compile➤Script from the menu bar.
  or
  Press CTRL+L.

  PowerBuilder compiles the function but does not save the function or close the Function painter. Any problems are reported in a second window, just as with the PowerScript painter.

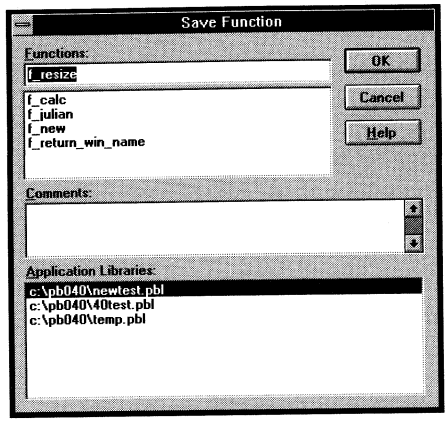
Compiling and saving

How you save the function depends on whether it is a global function or an object-level function.

❖ To save a global function:

1 Click the Return button.

PowerBuilder compiles the function. If there are no problems and the function has never been saved before, the Save Function dialog box displays.



2 If you want, you can rename the function, add comments about the function, or change the library you want to save the function in.

3 Click OK.

PowerBuilder saves the function in the specified library. You can view the function as an independent entry in the Library painter.

❖ To save an object-level function:

1 Click the last button in the PainterBar. This button represents the type of object you are defining a function for. For example, if you are defining a window function, a window displays in the button.

PowerBuilder compiles the function and returns you to the appropriate painter.

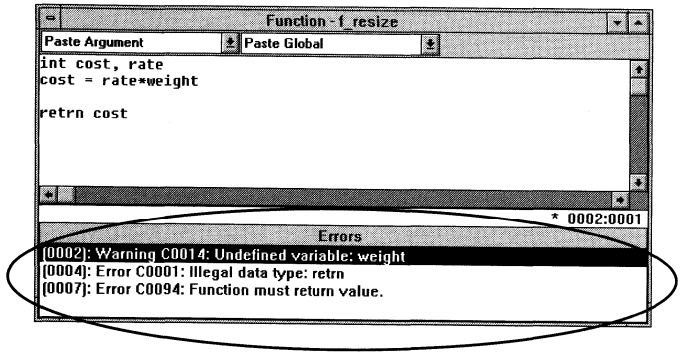
2 Save the object.

PowerBuilder saves the function as part of the object in the object's library. The object-level function does *not* appear as an independent entry in the Library painter, because it is actually part of the object.

Correcting compiler errors

If problems occur during a compile, PowerBuilder displays messages in a separate window below the text of the function. Just as in the PowerScript painter, there are two kinds of messages:

- Error messages
- Warning messages



You handle problems in a function the same way you handle problems in a script.

For information, see "Handling problems" in Chapter 3, "Writing Scripts."

After you correct all the errors, compile again. You cannot save the function until all the errors are fixed. Warnings do not prevent you from saving but may cause errors during execution.

To save a function with unresolved errors
Comment out the statements that have errors.

Modifying user-defined functions

You can change the definition of a user-defined function anytime. You change the processing performed by the function by simply modifying the statements in the workspace. 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:

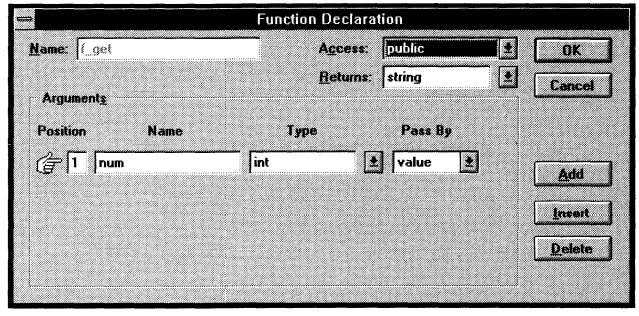
1 Open the Function painter.

2 Select the function you want to modify from the Select Function dialog box.

The Function painter workspace displays.

3 Select Edit➤Function Declaration from the menu bar.

The Function Declaration dialog box appears. The Name box is protected (you cannot change the name of a function here).



4 Make the changes you want, then click OK.

You return to the workspace.

5 Save the modified function.

To retain its current name, select File➤Save from the menu bar.

To save the modified function under a new name, select File➤Save As from the menu bar and enter a name.

Changing the arguments

You can change a function's arguments anytime:

- **Add** an argument by clicking the Add button. The next position number and 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 clicking the Insert button.

  The position number adjusts, and 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 number 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.

The next section describes an easy way to find all places a function is used.

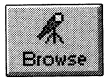
Seeing where a function is used

PowerBuilder provides browsing facilities to help you find where you have referenced your functions.

❖ **To determine which functions and scripts call a user-defined function:**

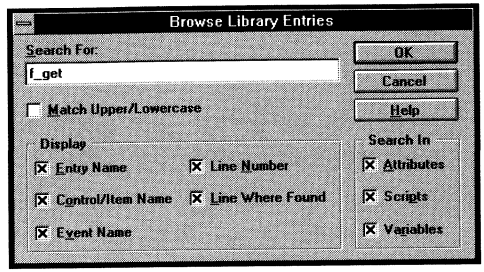
1 Open the Library painter.

2 Select all the entries you want to search for references to the user-defined function.



3 Click the Browse button in the PainterBar.

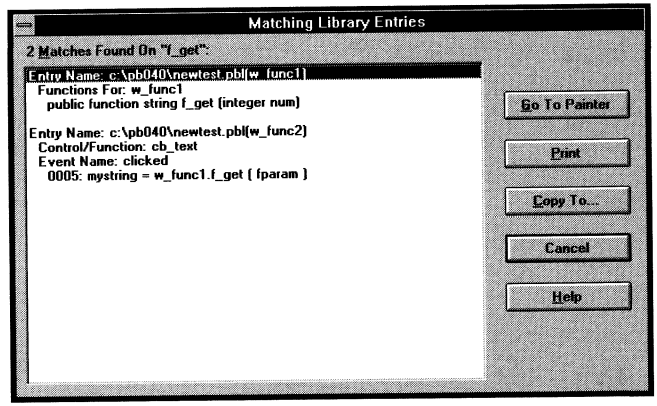
The Browse Library Entries dialog box displays.



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.



You can double-click a listed component to open the appropriate painter.

For more about browsing library entities, see Chapter 23, "Managing Libraries."

Using your functions

You use user-defined functions the same way you use built-in functions. You can call them in scripts or in other user-defined functions.

Calling the function

When calling a user-defined function, enclose the arguments in parentheses immediately after the function name. The arguments must agree in position and type with those defined for the function. If there are no arguments, you must enter parentheses after the function name to identify it as a function.

For example, if the user-defined function f_calculate takes two integer arguments and returns an integer, call the function like this:

```
integer  result
result = f_calculate(12, 45)
```

Calling object-level functions

As with built-in object-level functions, you use dot notation to call user-defined object-level functions:

object.function (*arguments*)

For example, if you defined the function wf_get_value for the window w_emp, call the function like this:

```
w_emp.wf_get_value()
```

Pasting user-defined functions

When you build a script in the PowerScript painter, you can type the call to the user-defined function or select it from the list of available functions in the Object browser. When you paste a function into a script, prototype values display to identify the arguments.

❖ To paste a user-defined function into a script or function:

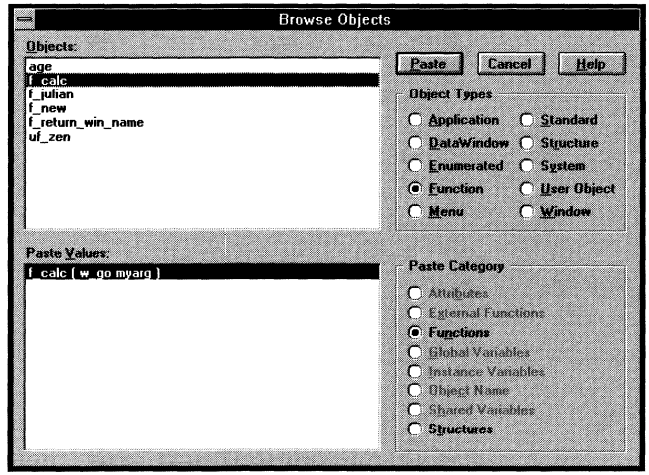
1 Move the insertion point to where you want to paste the function.

2 Click the Browse Objects button.
or
Press CTRL+B.
or
Select Edit➤Browse Objects from the menu bar.

The Object browser displays.

3 To paste a global function, choose Function as the Object Type. To paste an object-level function, choose the corresponding object (such as the name of a window) as the Object Type.

4 Click Functions in the Paste Category box.



The functions appear in the Paste Values box. If you are looking at object-level functions, PowerBuilder displays all built-in functions for that object as well as the functions you have defined.

5 Scroll to the function you want to paste.

> **Shortcut**
> If you click in the Paste Values box and type a letter, PowerBuilder scrolls to the first function with a name starting with that letter.

6 Double-click the function.
or
Select the function and click Paste.

The function and its prototype parameters display at the insertion point in your script.

7 Specify the required arguments.

Pasting global functions
You can also paste global user-defined functions in a script or function using Edit➤Paste Function on the menu bar.

CHAPTER 5

Working with Structures

About this chapter

This chapter describes how to build and use structures.

Contents

What are 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 user_struct, you can then refer to this collection of variables as user_struct.

Two kinds

There are two kinds of structures:

- **Global structures**, which are not associated with any object in your application. You can directly reference these structures anywhere 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

❖ To define a structure:

1 Open the Structure painter.

2 Define the variables that comprise the structure.

3 Save the structure.

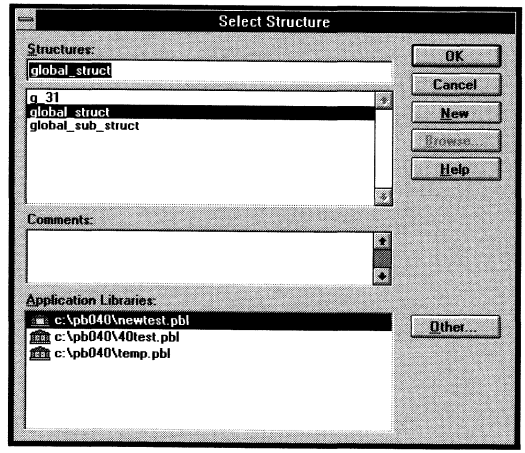
Opening the Structure painter

How you open the Structure painter depends on whether you are defining a global structure or an object-level structure.

❖ To open the Structure painter for a global structure:

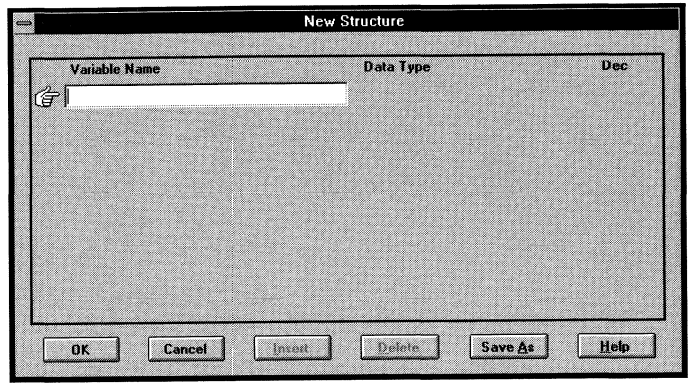
1 Click the Structure painter button in the PainterBar or PowerPanel.

The Select Structure dialog box displays.



2 Click the New button.

The New Structure dialog box displays.



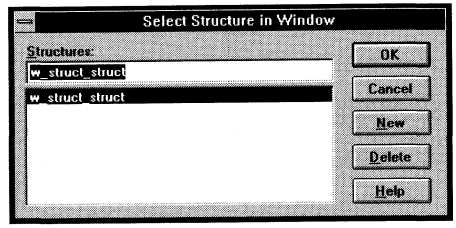
❖ To open the Structure painter for an object-level structure:

1 Open the painter for the object you want to define the structure for.

For example, if you want to define a structure for the window w_emp, open the Window painter and select w_emp.

2 From the Declare menu, select the appropriate menu item: Window Structures, Menu Structures, User Object Structures, or Application Structures. You can use the Declare menu in the Window, Menu, User Object, or PowerScript painter.

A Select Structure dialog box displays.



Here you can select a structure for the current object or declare a new structure for the object.

3 To define a new structure, click the New button.

The New Structure dialog box displays.

Defining the variables

❖ To define the variables that comprise the structure:

1 Enter the name of a variable that you want to include in the structure.

2 Enter the data type of the variable. The default for the first variable is string; the default for subsequent variables is the data type of the previous variable.

You can specify any PowerBuilder data type, 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 defined and want the structure to include the window, type **w_calculator** as the data type. (You cannot select w_calculator from the list, since the list only shows 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 data type.

3 Enter the number of decimal places in the Dec box if the data type is decimal. The default is 2.

4 Repeat until you have entered all the variables.

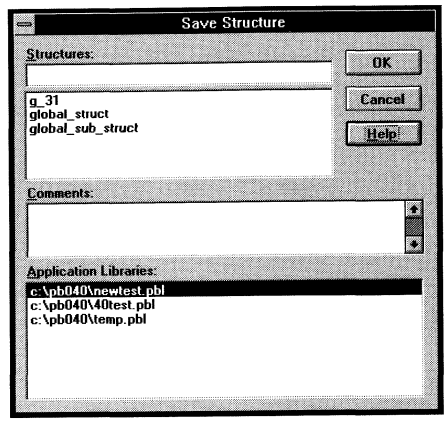
Saving the structure

How you save the structure depends on whether it is a global structure or an object-level structure.

❖ To save a global structure:

1 Click OK in the New Structure dialog box.

The Save Structure dialog box displays.



2 Name the structure. Names can have up to 40 characters.

For information about valid characters, see *PowerScript Language*.

3 Add comments to describe your structure.

4 Choose the library to save the structure in.

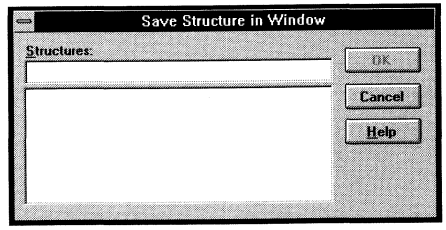
5 Click OK.

PowerBuilder stores the structure in the specified library. You can view the structure as an independent entry in the Library painter.

❖ To save an object-level structure:

1 Click OK in the New Structure dialog box.

The Save Structure in *ObjectType* dialog box displays.



2 Name the structure and click OK.

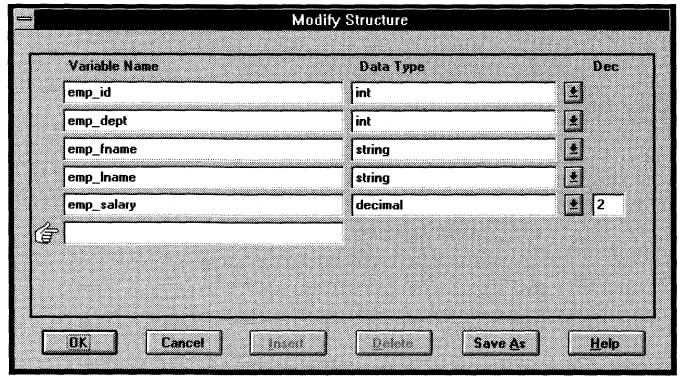
PowerBuilder saves the structure as part of the object in the object's library.

Modifying structures

❖ To modify a structure:

1 Select the structure you want to modify in the Select Structure dialog box.

The Modify Structure dialog box displays.



2 Review the variable information displayed in the window and then modify the structure as necessary.

To insert a variable between existing variables, move the hand pointer to the variable that you want to follow the new variable and click the Insert button.

To delete a variable, click the Delete button.

3 Save the modified structure by doing one of the following:

- Click OK to save the structure with its current name.
- Click Save As to save the structure with a new name.

Building a similar structure
To build a structure that is similar to an existing structure, modify the existing structure and then click Save As to save the structure under another name or in another library.

Using structures

After you define the structure, you can:

- Reference the structure in scripts and functions
- Pass the structure to functions
- Display and paste information about structures by using the Object browser

Referencing structures

When you define a structure in the Structure painter, you are defining a new data type. You can use this new data type 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 variables in the structure.

Referencing global structures

The variables in a structure are similar to the attributes of a PowerBuilder object. To reference a global structure's variable, use dot notation:

structure.variable

Example

Assume that emp_data 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 emp_data and use dot notation to reference the structure's variables, as shown in the following script:

```
emp_data     emp1, emp2        // Declare 2 variables
                               // of type emp_data.

emp1.emp_id = 100              // Assign values to the
emp1.emp_dept = 200            // structure variables.
emp1.emp_fname = "John"
emp1.emp_lname = "Paul-Jones"
```

```
emp1.emp_salary = 19908.23

// Retrieve the value of a structure variable.
emp2.emp_salary = emp1.emp_salary * 1.05

// Use a structure variable in a
// PowerScript function.
MessageBox ("New Salary",  &
   String(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 as:

structure.variable

Example

Assume that the structure cust_data 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:

```
cust_data   cust1
cust1.name = "Joe"
```

Allowing access to object-level structures outside the object

You can also choose to make object-level structures accessible outside the object.

❖ To reference object-level structures outside the object:

1 In the object that defines the structure, declare an instance variable of the structure type.

2 In any script or user-defined function in the application, reference a variable in the structure using the following syntax:

object.instance_variable.variable

Example

Assume you have defined a structure for the window w_history named cust_data and you want to be able to use it anywhere in your application. Define an instance variable of type cust_data for w_history:

```
cust_data   winstruct
```

In other scripts, reference the window structure through the instance variable, such as:

```
w_history.winstruct.name
```

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 emp2 to emp1:

```
emp_data emp1, emp2
.
.
.
emp1 = emp2
```

Using structures with functions

You can pass structures as arguments in user-defined functions. Simply name the structure as the data type when defining the argument.

Similarly, user-defined functions can return structures. Name the structure as the return type for the function.

In PowerBuilder, you can define external functions that take structures as arguments.

Example

Assume the following:

- Revise is an external function that expects a structure as its argument.

- Emp_data is a declared variable of a structure data type.

You can call the function as follows:

```
Revise(emp_data)
```

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 *Building Applications.*

Displaying and pasting structure information

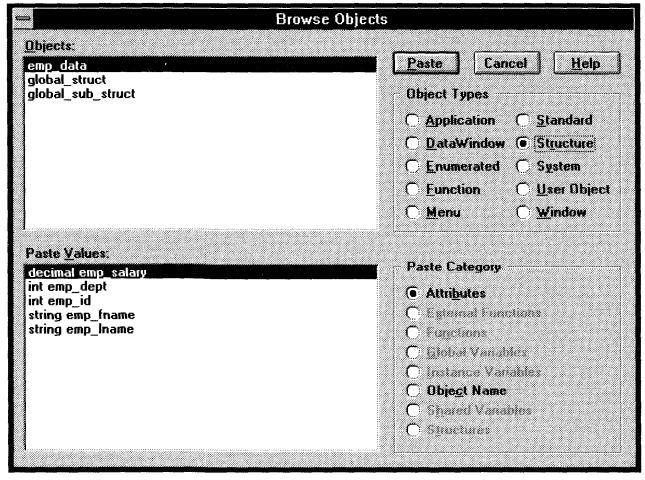
You can display the names and variables of defined structures in the Object browser. You can also paste these entries into a script.

❖ To display structure information:

1 To display a global structure, choose Structure as the Object Type.

To display an object-level structure, choose the corresponding object (such as the name of the window) as the Object Type.

2 When displaying a global structure, choose Attributes as the Paste Category to display the names of the variables in the structure or select Object Name to display the name of the structure.



When displaying an object-level structure, choose Structures as the Paste Category. PowerBuilder displays both the structure name and the variables.

❖ To paste the information into a script:

- Double-click the item in the Paste Values box.
  or
  Select the item and click Paste.

  The item displays at the insertion point in your script.

PART THREE

Working with Windows

This part describes how to create windows for your application. It covers attributes of windows, the controls you can place in windows, how to use inheritance to save time and effort, how to define menus, how to use user objects, and how to use user events.

CHAPTER 6

Defining Windows

About this chapter

This chapter describes how to build windows in the Window painter.

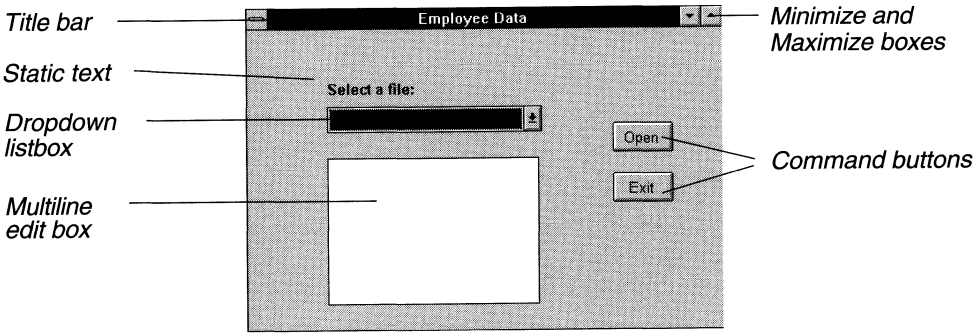
Contents

Overview of 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:

- **Attributes**, which 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 attributes in the Window painter (you can also dynamically change window attributes in scripts during execution). You also 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 CheckBoxes, CommandButtons, or MultiLineEdits) 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

Each operating environment (such as Microsoft Windows or the Macintosh) 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 of these issues is beyond the scope of this manual. You should acquire a book that specifically addresses design guidelines for applications on your target platform and apply the rules when you use PowerBuilder to create your application. You might want to look at the following:

- For Microsoft Windows: *The Windows Interface: An Application Design Guide*, from Microsoft Press
- For Macintosh: *Human Interface Guidelines: The Apple Desktop Interface,* produced by Apple Computer and published by Addison-Wesley Publishing Company
- For Motif: *Motif Style Guide,* produced by the Open Software Foundation and published by Prentice Hall

Building windows

When you build a window, you:

- Specify the appearance and behavior of the window by setting its attributes.
- 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 aren't 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 142.

For information on using inheritance to build a window, see "Using inheritance to build a window" on page 164.

Types of windows

PowerBuilder provides the following types of windows:

- Main
- Popup
- Child
- Response
- Multi Document Interface (MDI) frame
- 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 *Building Applications*.

Using main windows

Define your independent windows as main windows.

For example, assume your application contains a calculator or scratch-pad window that you want always available to the user. Make it a main window, which can be displayed anytime, anywhere on the screen. As a main window, it can overlap other windows on the screen.

Popup windows

Popup windows are typically opened from another window, which in most cases becomes the popup window's parent.

> **Using the application's Open event**
> If you open a popup window from the application's Open event, the popup window doesn't have a parent and works the same as a main window.

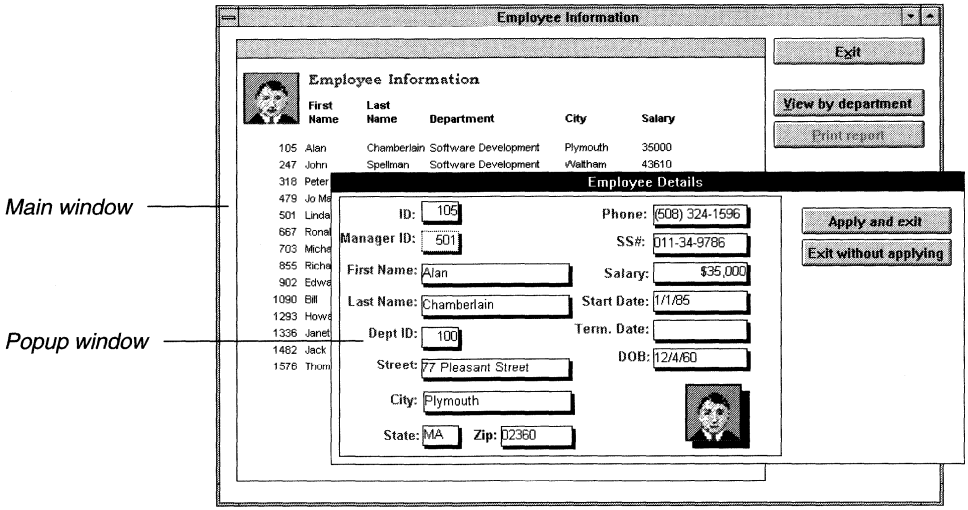
Popup windows can display outside their parent window. They cannot be overlaid by their parent.

A popup window is hidden when its parent is minimized. A popup window is closed when its parent is closed.

When you minimize a popup window, the button for the window displays at the bottom of the screen.

Using popup windows

Popup windows are often used as supporting windows. For example, say you have a window containing master information, such as employee information. You can use a popup window to allow a user to see details of a particular employee.



Explicitly naming a parent

In most cases, the window that opens a popup window becomes that window's parent. For example, say a script in w_go has this statement:

```
Open(w_popup)
```

That statement defines w_go as the parent of w_popup.

You can also explicitly name a popup window's parent when you use Open, as follows:

Open (*popupwindow, parentwindow*)

For example, the following statement:

```
Open(w_popup, w_parent)
```

opens w_popup and makes w_parent its parent.

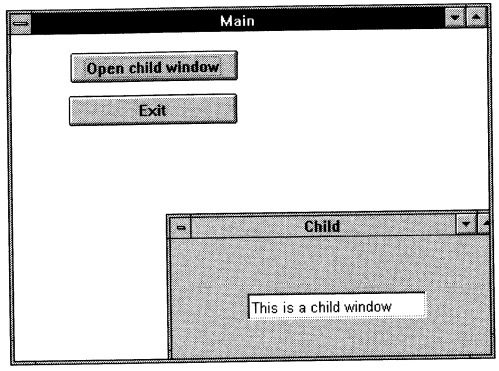
However, there are also other considerations regarding which window becomes the parent of an opened window.

For more information, see the description of the Open function in the *Function Reference*.

Child windows

Child windows are always opened from within a main or popup 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 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 button displays at the bottom of their parent.

The initial position of the child is relative to the parent and not the entire screen.

A child window closes when you close its parent.

Using child windows

You will probably not use child windows very often. 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 building MDI applications, see *Building Applications*.

Response windows

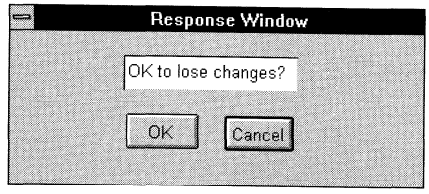
Response windows request information from the user. They are always opened from within another window (its parent). Typically, a response window is opened after some event occurs in the parent window.

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. However, the user *can* go to other Windows applications. But when the user returns to the application, the response window is still active.

Response windows act like modal popup 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 through the PowerScript function MessageBox.

For more information, see the description of MessageBox in the *Function 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 *Building Applications*.

Building a new window

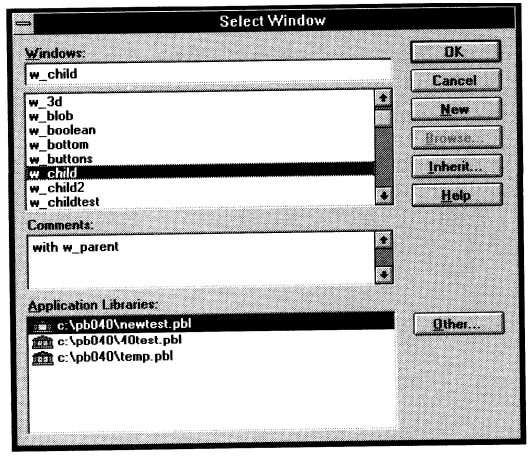
This section describes how to build windows from scratch. You will use this technique to create windows that are not based on existing windows.

Opening the Window painter

❖ To open the Window painter:

1 Click the Window painter button in the PowerBar or PowerPanel.

The Select Window dialog box lists the windows in the current library.

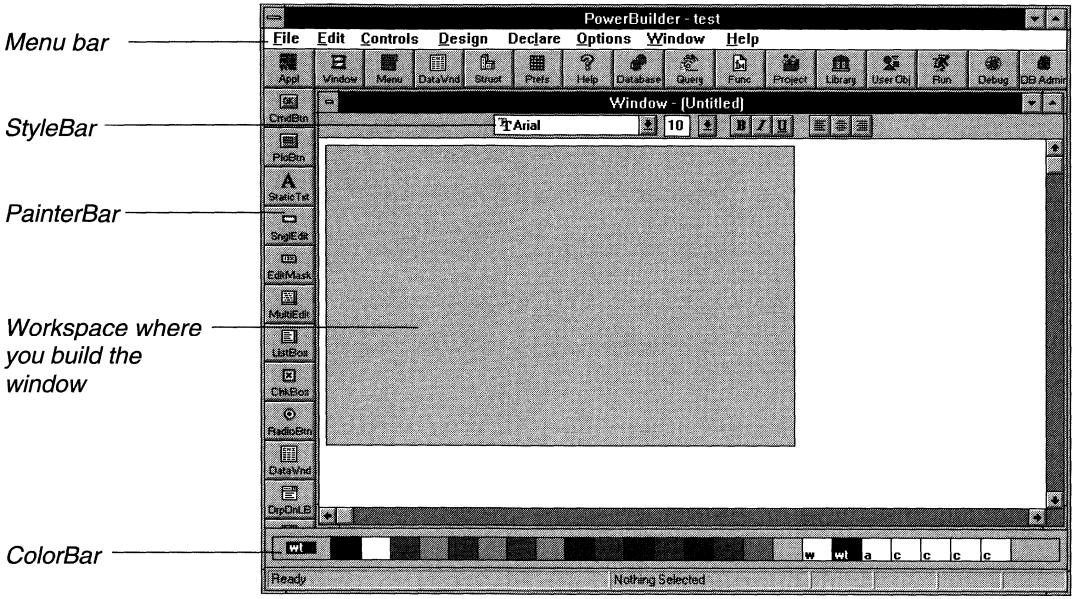


2 Click the New button to build a new window.

The Window painter workspace displays.

About the painter

You design windows in the Window painter. The painter has a menu bar, a StyleBar, a customizable PainterBar, a workspace, and a ColorBar.



About the PainterBar

The PainterBar in the Window painter works the same as in the other painters. You can move it around and customize it.

For complete information about manipulating the PainterBar, see Chapter 1, "The World of PowerBuilder."

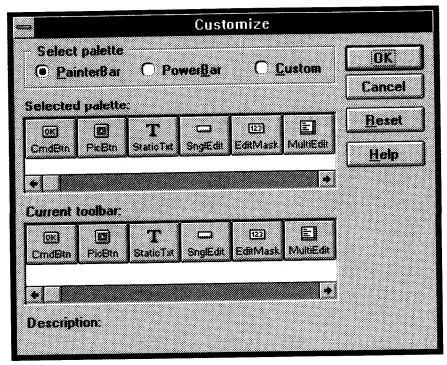
One customization you might want to do specifically in the Window painter is add particular user objects to the PainterBar, so you can place a frequently used user object in a window by simply clicking a button in the PainterBar.

❖ To add a user object to the PainterBar:

1 Click the right mouse button in the PainterBar.

2 Select Customize from the PainterBar's popup menu.

The Customize dialog box displays.



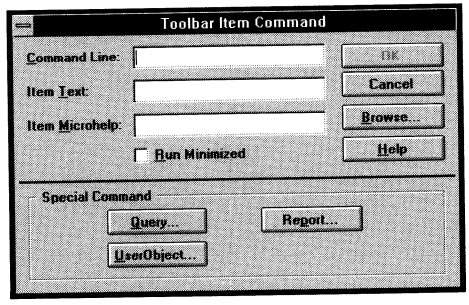
3 Click the Custom button in the Select Palette group.

PowerBuilder displays a set of custom buttons you can place in the PainterBar.

4 Drag one of the unassigned custom buttons into the Current Toolbar box.

5 Release the mouse button.

PowerBuilder places the custom button in the Current Toolbar box and displays the Toolbar Item Command window.



6 Click the UserObject button in the Special Command box.

The Select User Object dialog box displays.

7 Select the user object you want and click OK.

PowerBuilder fills in the Command Line box.

8 Type the text you want to appear in the button and as the button's MicroHelp.

9 Click OK.

PowerBuilder adds the button to the PainterBar. To place the user object in the window, simply click the button.

About the StyleBar and ColorBar

In addition to the PainterBar, the Window painter has two additional toolbars, the StyleBar and the ColorBar. You use the StyleBar to assign attributes to text, and you use the ColorBar to assign colors to window elements.

You can display these toolbars in the Window painter window or in the PowerBuilder frame, which encompasses all windows. You can also hide the ColorBar.

❖ To specify the position of the StyleBar or ColorBar:

1 Select Options➤Color Toolbar or Options➤Text Style Toolbar from the menu bar.

2 Choose the position of the toolbar from the cascading menu.

Working in the Window painter

The Window painter is a flexible environment. You can use:

- Popup menus to specify a style for a control or the window
- Buttons in the PainterBar or the items on the Controls menu to place controls in the window
- Items on the Design menu to specify the appearance and behavior of the window
- The mouse to size and position the window and its controls
- The PowerScript button to build scripts for the window and its controls
- Items on the Declare menu to declare variables, functions, structures, and events for the window and its controls

You can also use keyboard shortcuts to accomplish certain tasks, as described next.

Shortcut keys

The following table lists the keyboard shortcuts available in the Window painter:

Action	Key combination	Comments
Boldface text	CTRL+B	Toggles boldface on and off
Center text	CTRL+N	
Close painter	CTRL+F4	
Copy control to private clipboard	CTRL+C	See "Copying controls" in Chapter 7, "Working with Controls"
Cut control to private clipboard	CTRL+X	
Debug	CTRL+D	Go to Debug painter
Delete	DEL	Deletes all selected controls
File editor	SHIFT+F6	Available everywhere in PowerBuilder
Duplicate	CTRL+T	
Edit text	CTRL+E	Activates the Text box in the StyleBar
Exit PowerBuilder	ALT+F4	Available everywhere in PowerBuilder
Font family	CTRL+F	Activates the Font box in the StyleBar
Italicize text	CTRL+I	Toggles italics on and off
Left justify text	CTRL+L	
Next child window	CTRL+F6	
Paste control from private clipboard	CTRL+V	See "Copying controls" in Chapter 7, "Working with Controls"
PowerPanel	CTRL+P	Available everywhere in PowerBuilder
Return focus to control	CTRL+O	Returns focus to the control from the StyleBar
Right justify text	CTRL+G	

Action	Key combination	Comments
Run	CTRL+R	Available everywhere in PowerBuilder
Script	CTRL+S	
Select all	CTRL+A	
Switch to	CTRL+ESC, ALT+ESC	Available everywhere in PowerBuilder
Tab	TAB	Moves to next control in the window
Tab backwards	SHIFT+TAB	Moves to previous control in the window
Undo/Redo	CTRL+Z	Undoes the most recent change (including the most recent undo)
Underline text	CTRL+U	Toggles underline on and off

Defining the window's style

Every window and control has a **style** that determines how it appears to the user.

A window's style encompasses its:

- Type
- Basic appearance
- Initial position on the screen
- Button
- Pointer

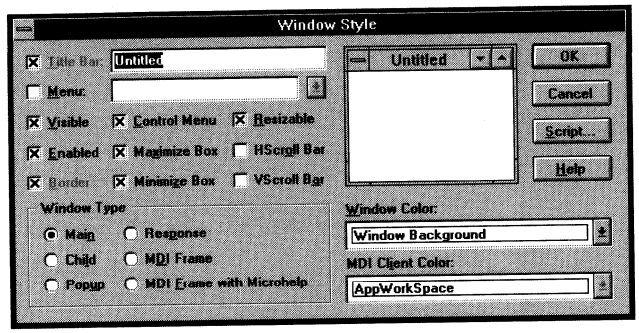
About defining a window's style
When you define a window's style in the Window painter, you are actually assigning values to the attributes for the window. You can programmatically change a window's style during execution by setting its attributes in scripts.

For a complete list of window attributes, see *Objects and Controls*.

❖ To define a window's style:

1 Double-click the window's background.
or
Select Design➤Window Style from the menu bar.

The Window Style dialog box displays.



The window representation at the right of the Window Style dialog box shows how the window will display during execution with the current attribute values.

2 Specify the window's type.

3 Specify other window properties.

4 Associate a menu with the window (if appropriate).

5 Choose a color for the window background.

6 Click OK to return to the Window painter.

You can modify a window's style anytime by displaying the Window Style dialog box and changing the properties.

Using the popup menu

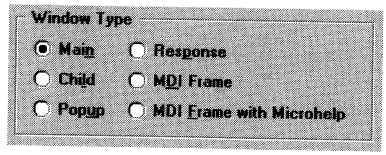
You can also modify a window's attributes by clicking the right mouse button on the window background and choosing the attribute from the window's popup menu.

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:

- Click the appropriate radio button in the Window Type group box in the Window Style window.



Depending on the type of window, PowerBuilder enables or disables certain checkboxes that specify other properties of the window. For example, if you are creating a main window, the Title Bar checkbox is disabled: main windows *always* have title bars, so you cannot deselect the Title Bar checkbox.

Specifying other basic window properties

By selecting and deselecting checkboxes in the Window Style window, you can specify whether the window is sizable or minimizable, whether it has scroll bars, and so on.

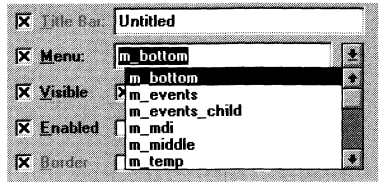
Note the following:

- 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

❖ To associate a menu with the window:

- Choose the menu from the Menu listbox. PowerBuilder lists all menus in the application's library search path.



Changing the menu
You can change a menu associated with a window during execution using the ChangeMenu function.

For more information, see the *Function Reference*.

Choosing a window color

❖ To specify the color of a window:

- Specify the color of the window from the Window Color listbox.

  For main, child, popup, and response windows, the default color is Window Background, which is the color that the user has specified in the Windows Control Panel for Window Background (unless you are defining a 3D window, in which case the default color is gray). You can keep this default or choose another color.

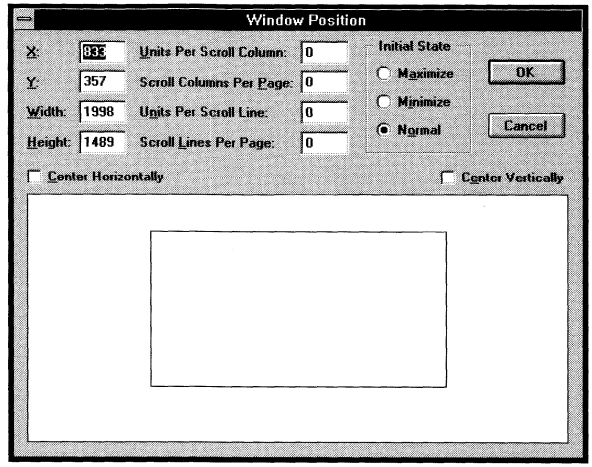
For more about using colors in windows, including how to define your own custom colors, see Chapter 7, "Working with Controls."

Choosing the window's size and position

❖ To move a window:

1 Select Position from the window's popup menu.
or
Select Design➤Window Position from the menu bar.

The Window Position dialog box displays.



2 Move the window as appropriate. You can drag the representation of the window with the mouse, center the window in the screen, or enter values for X and Y location (in PowerBuilder units—see below).

3 Click OK.

About X and Y values
For main, popup, 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.

❖ To resize a window:

- Either use the mouse in the Window painter workspace (drag an edge with the mouse) or use the Window Position dialog box: you can resize the window by dragging one of its corners in the dialog box, or change the window's Width and Height attributes (in PowerBuilder units—see below).

About PowerBuilder units

All window measurements are in PowerBuilder units. Using these units, you can build applications that look similar on different resolution screens. A horizontal PowerBuilder unit is 1/32 of the width of an average character in the system font. A vertical PowerBuilder unit is 1/64 of the system font height.

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 the grid size in the Window and DataWindow painters, which is in pixels.)

For more about PowerBuilder units, see *PowerScript Language*.

Specifying window scrolling

If your window is resizable, it is possible that not all the window's contents are visible during execution. In such cases, you should make the window scrollable by providing scrollbars. You specify that a window has scrollbars in the Window Style window.

By default, PowerBuilder controls scrolling when scrollbars are present. If you want, you can control the amount of scrolling by specifying values for attributes in the Window Position dialog box.

❖ To specify window scrolling:

1 Select Design➤Window Position from the menu bar.

The Window Position dialog box displays.

2 Specify scrolling characteristics, as follows:

Option	Meaning
Units Per Scroll Column	The number of PowerBuilder units to scroll right or left when user clicks the right or left arrow in the horizontal scrollbar. When value is 0 (the default), scrolls 1/100 the width of the window.
Scroll Columns Per Page	The number of columns to scroll when user clicks the horizontal scrollbar itself. When value is 0 (the default), scrolls 10 columns.

Option	Meaning
Units Per Scroll Line	The number of PowerBuilder units to scroll up or down when user clicks the up or down arrow in the vertical scrollbar. When value is 0 (the default), scrolls 1/100 the height of the window.
Scroll Lines Per Page	The number of lines to scroll when user clicks the vertical scrollbar itself. When value is 0 (the default), scrolls 10 lines.

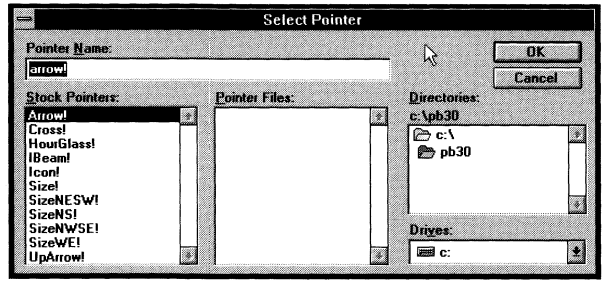
Choosing the window's pointer

You can specify the default pointer used whenever the mouse is over the window.

❖ To choose the window pointer:

1 Select Pointer from the window's popup menu.

The Select Pointer dialog box displays.



2 Choose the pointer either from the Stock Pointers list, or, if you have files containing pointer definitions (CUR files), choose one from the Pointer Files list.

The chosen pointer displays next to the Pointer Name box.

3 Click OK.

Specifying the pointer for a control
You can specify the pointer that displays when the mouse is over an individual control by using the control's popup menu in the Window painter.

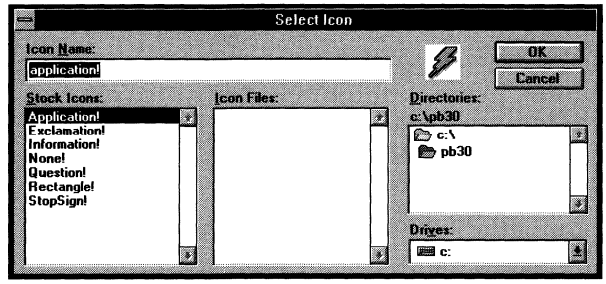
Choosing the window's icon

If the window is minimizable, you can specify an icon to represent the minimized window. If you don't choose an icon, PowerBuilder uses the application icon for the minimized window.

❖ To choose the window icon:

1 Select Icon from the window's popup menu.

The Select Icon dialog box displays.



2 Choose the icon either from the Stock Icons list or from the Icon Files list if you have ICO files containing icon definitions.

The chosen icon is shown to the right of the Icon Name box.

3 Click OK.

Changing the icon during execution
You can change the window icon during execution by assigning the name of the icon file to the window's Icon attribute.

Adding controls

When you build a window, you place controls (such as CheckBoxes, CommandButtons, and MultiLineEdits) 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 7, "Working with Controls."

Saving the window

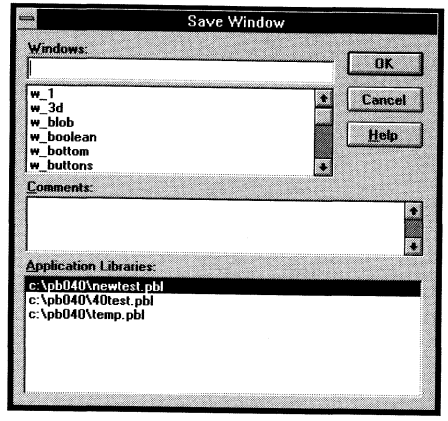
You can save the window you are working on anytime.

❖ 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 box (see below).

3 Write comments 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 to save the window in.

5 Click OK.

Naming the window

The window name can be any valid PowerBuilder identifier with up to 40 characters.

For information about PowerBuilder identifiers, see *PowerScript Language*.

A recommendation

When you name windows, you should use a two-part name: a standard prefix that identifies the object as a window (such as w_) and 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 your window, you can preview its appearance from within the Window painter. By previewing the window, you get a good idea of how it will feel during execution.

❖ To preview a window:

- Select Design➤Preview from the menu bar.
  or
  Press CTRL+W.

  The window displays with the attributes you have defined, such as a 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 the window is resizable)
- Maximize and restore it (if these properties were enabled)
- Tab around the window
- Select controls

What you cannot do

You cannot:

- Change attributes 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:

- Select Design➤Preview from the menu bar.
  or
  Press CTRL+W.

  You return to the Window painter.

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 that is sent to the printer depends on variables specified in the [Library] section of the PB.INI file.

> **Print settings**
> You can view and change the print settings in the Library painter (select Entry➤Print from the menu bar) or in the Preferences painter.

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 themselves 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 box change.

Defining your own events

You can also define your own events (called **user events**) for a window or control, then use the TriggerEvent function to trigger your user event.

For example, assume you offer your 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 as well after asking for confirmation. So here you want the same type of processing to happen after different events.

You could define a user event, such as UpdateDB, for the window, write a script for that event, then everywhere you want that event triggered, call the TriggerEvent function.

To learn how to use user events, see Chapter 11, "Working with User Events."

About functions for windows and controls

PowerBuilder provides built-in functions that act on windows and built-in functions that act on types of controls. You can use these functions in scripts to manipulate your windows and controls.

For example, to open a window, you use the built-in window-level function Open.

Passing parameters
You can pass parameters between windows by opening them with the function OpenWithParm and by closing them with CloseWithReturn.

For more information, see the *Function Reference*.

You can define your own window-level functions to make it easier to manipulate your windows.

For more information, see Chapter 4, "Working with User-Defined Functions."

About attributes of windows and controls

In scripts, you can assign values to the attributes of objects and controls to change their appearance or behavior. You can also test the values of attributes 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 the information that is retrieved and displayed in a DataWindow control.

To refer to attributes of an object or control, use dot notation to identify the object and the attribute:

object.attribute
control.attribute

Unless you identify the object or control when you refer to an attribute, PowerBuilder assumes you are referring to an attribute of the object or control the script is written for.

The reserved word Parent

In the script for a 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)
```

Using Parent instead of typing the name of the window makes the script more general and allows you to copy it easily to other scripts and use in user objects.

For more information

Objects and Controls lists all attributes, events, and built-in functions for all PowerBuilder objects (including windows) and each type of control.

Declaring instance variables

Often you have data that 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; each of them 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 for 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.

The best approach

When declaring variables, you need to consider what the scope of the variable is. If the variable is only meaningful 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 more information

For a complete description of the types of variables and how to declare them, see *PowerScript Language*.

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.

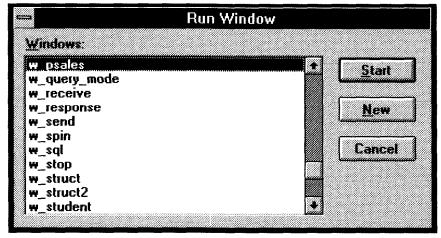
❖ To run a window:

1 Press CTRL+SHIFT+W.

You must save your work before running a window. If you have not saved your work, PowerBuilder prompts you to do so.

2 If necessary, save your work.

The Run Window dialog box displays.



3 Choose the window you want to run, then click Start.

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, except that it does not have access to global variables that you have defined for the application (though it *does* have access to built-in globals, such as SQLCA). Also, the SystemError event is not triggered if there is an error, because SystemError is an application-object event.

Adding Run Window to a toolbar

You can add a button that runs a window to a toolbar. The button is in the PowerBar palette.

For more information about customizing toolbars, see Chapter 1, "The World of PowerBuilder."

Using inheritance to build a window

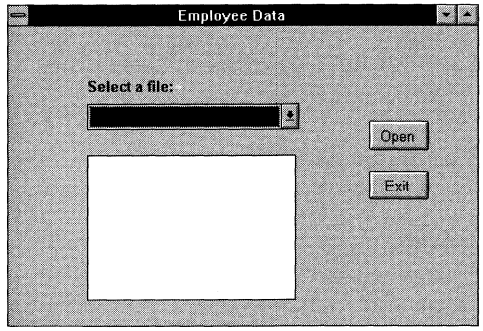
When you build a window that inherits its definition (style, events, functions, structures, variables, controls, and scripts) from an existing window, you save coding time. All you have to do is modify the inherited definition to meet the requirements of the current situation.

Example

Assume your application has a window w_employee that has:

- A title (Employee Data)
- Text that says *Select a File:*
- A dropdown listbox 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:



Now assume you need to build another window that performs similar processing. The only difference is that the dropdown listbox displays customer files instead of employee files and there is a Delete button so the user can delete files.

Your choices

To build this window, you have three choices:

- Build a new window from scratch as described above
- Modify the existing window (w_employee), then save it under another name
- Use inheritance to build a window that inherits the definition from the existing window (that is, build a descendent window)

If you build a new window from scratch, the amount of work is the same as building the first window. If you use either of the two other methods, all you have to do is change the title and the files that display in the DropDownListBox and add a Delete button. Both methods save work.

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 manually make 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 get consistency in the code and in the application windows.

Using our example

Continuing with our example, the best way to build the window to display customer information instead of employee information is to build a window that is inherited from w_employee. You change the title and modify the scripts, then save the descendent window using a new name (such as w_customer).

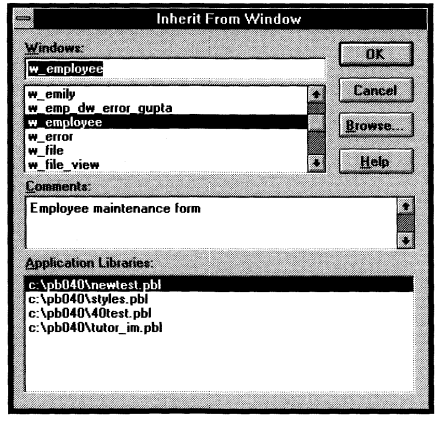
❖ To use inheritance to build a descendent window:

1 Open the Window painter.

The Select Window dialog box displays.

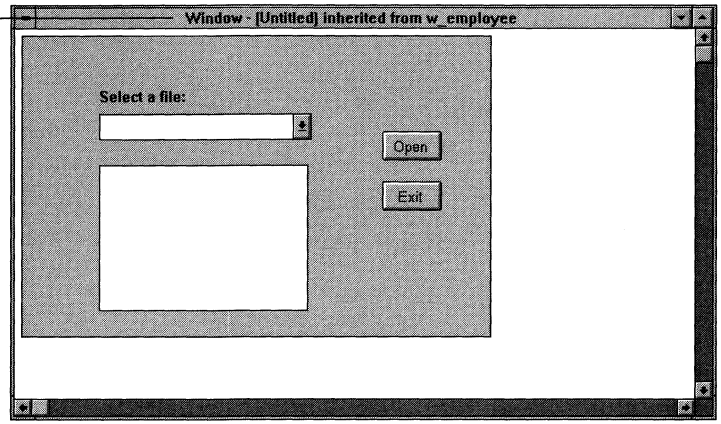
2 Click the Inherit button in the Select Window dialog box.

The Inherit From Window dialog box displays.



3 Double-click the window you want to use to build the new descendant window.
or
Select the window and click OK.

New window is inherited



The selected window displays in the Window painter workspace. The title of the workspace indicates that the window is a descendant and identifies the window from which it inherits its definition (the ancestor window).

4 Make the changes to the descendent window.

5 Save the window under a new name.

What inherited objects contain

When you use inheritance to build an object, everything in the ancestor object is inherited in all its descendants.

What you can do in the descendent window

You can:

- Change the attributes 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, events, and structures
- Extend or override inherited scripts
- Declare functions, structures, and variables for the window
- Declare user events for the window and its controls

What you cannot do

The only thing you cannot do is delete inherited controls.

Unneeded inherited controls
If you don't need an inherited control, you can make it invisible in the descendent window.

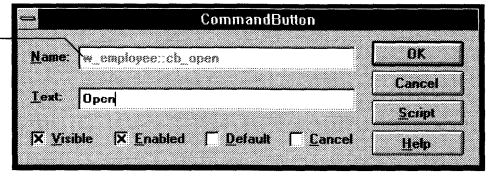
About control names

PowerBuilder uses this syntax to show names of inherited controls:

ancestorwindow :: *control*

For example, in w_customer, which is inherited from w_employee, you see the following for the Open button, which is defined in w_employee:

Inherited control



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.

So you should develop a naming convention for controls in windows that you plan to use as ancestors.

For more information

The issues concerning inheritance with windows are the same as the issues concerning inheritance with user objects and menus. Chapter 8, "Understanding Inheritance," describes the following issues in detail:

- The basics of inheritance
- How to view the inheritance hierarchy
- Considerations when using inherited objects
- How to use inherited scripts
- How to call an ancestor script
- How to call an ancestor function

Creating window instances

When you build an application, you might want to display several windows that are identical in structure, but have different data values.

For example, you might have a w_employee window and want to display information for two or more employees at the same time by opening multiple copies (instances) of the w_employee window.

You can do that, but first you need to understand how PowerBuilder stores window definitions.

How PowerBuilder stores window definitions

When you save a window, PowerBuilder actually generates *two* entities in the library:

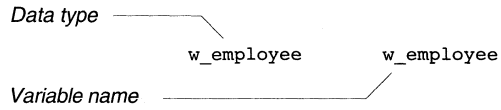
- **A new data type** The name of the data type is the same as the name of the window.

  For example, when you save a window named w_employee, PowerBuilder internally creates a data type named w_employee.

- **A new global variable of the new data type** The name of the global variable is the same as the name of the window.

  For example, when you save the w_employee window, you are implicitly also defining a global variable named w_employee of type w_employee.

  It is as if you had made the following declaration:



By duplicating the name of the data type and variable, new PowerBuilder users can access windows easily through their variables while ignoring the concept of data type.

What happens when you open a window

To open a window, you use the Open function, such as:

```
Open(w_employee)
```

What this actually does is create an instance of the data type w_employee and assign it a reference to the global variable, also named w_employee.

As you have probably noticed, when you open a window that is already open, PowerBuilder simply activates the existing window; it does not open a new window. For example, consider this script for a CommandButton's Clicked event:

```
Open(w_employee)
```

No matter how many times this button is clicked, there is still only one window w_employee. It is pointed to by the global variable w_employee.

To open multiple instances of a window, you declare variables of the window's type.

Declaring instances of windows

Because a window is actually a data type, you can declare variables of that data type, just as you can declare integers, strings, and so on. You can then refer to those variables in code.

For example:

```
w_employee  mywin
```

declares a variable named mywin of type w_employee.

Opening an instance

To open a window instance, you refer to the window variable in the Open function:

```
w_employee  mywin
Open(mywin)
```

Here the Open function determines that the data type of the variable mywin is w_employee. It then creates an instance of w_employee and assigns a reference to the mywin variable.

If you code the above script for the Clicked event for a CommandButton, each time the button is clicked a new instance of w_employee is created. In other words, a new window is opened each time the button is clicked.

So by creating variables whose data type is the name of a window, you can open multiple instances of a window. This is easy and straightforward. PowerBuilder manages the windows for you—for example, freeing memory when you close the windows.

Closing an instance

A common way to close the instances of a window is to put a CommandButton in the window with this script for the clicked event:

```
Close(Parent)
```

This script closes the parent of the button (the window in which the button displays). Continuing the example above, if you put a CommandButton in w_employee, the script closes the current instance of w_employee.

Limitation of using variables

But there is one problem: using the above technique, you cannot reference a particular instance of the window from another window. For example, if there are three windows open, you cannot explicitly refer to the second one from another window. There is no global handle for windows opened using reference variables.

But there is a way to get referenceable window instances: use arrays of windows.

Using window arrays

To create an array of windows, declare an array of the data type of the window.

For example, the following statement declares an array named myarray, which contains five instances of the window w_employee:

```
w_employee  myarray[5]
```

Using unbounded arrays
You can also create unbounded arrays of windows if the number of windows to be opened is not known at compile time.

Opening an instance using an array

To open an instance of a window in an array, use the Open function and pass it the array index. Continuing the example above, the following statements open the first and second instances of the window w_employee:

```
Open(myarray[1])  // Opens the first instance
                  // of the window w_employee
Open(myarray[2])  // Opens the second instance
                  // of the window w_employee
```

Moving first instance opened
The statements in this example open the second instance of the window at the same screen location as the first instance. Therefore, you would want to call the Move function in the script to relocate the first instance before the second Open function call.

Manipulating arrays

Using arrays of windows, you can manipulate particular instances by using the array index.

For example, the following statement hides the second window in the array:

```
myarray[2].Hide()
```

You can also reference controls in windows by using the array index, such as:

```
myarray[2].st_count.text = "2"
```

Opening many windows
When you open or close a large number of instances of a window, you might want to use a FOR...NEXT control structure in the main window to open or close the instances. For example:

```
w_employeemyarray[5]
for i = 1 to 5
   Open(myarray[i])
next
```

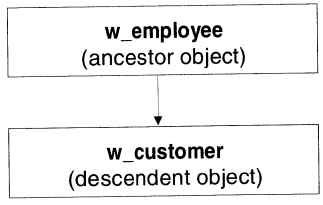
Creating mixed arrays

In the previous example, all windows in the array are the same type. You can also create arrays of mixed type. Before you understand this technique, you need to know one more thing about window inheritance.

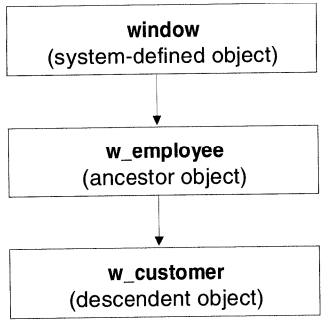
The built-in window data type

All windows you define are actually descendants of the built-in data type window.

For example, the discussion of window inheritance starting on page 164 described the window w_employee, which is defined from scratch, and w_customer, which inherits from w_employee. This hierarchy was presented:



In fact, the complete inheritance hierarchy is the following:



The system-defined object named window is the ancestor of *all* windows you define in PowerBuilder. The built-in object named window defines attributes that are used in all windows (such as X, Y, and Title).

If you declare a variable of type window, you can reference any type of window in the application. This is because all user-defined windows are a kind of window.

The following code creates an array of three windows. The array is named newarray. The array can reference any type of window, because all user-defined windows are derived from the window data type:

```
window   newarray[3]
string   win[3]
int      i

win[1] = "w_employee"
win[2] = "w_customer"
win[3] = "w_sales"

for i = 1 to 3
   Open(newarray[i], win[i])
next
```

The code uses a new form of the Open function:

Open (*windowVariable, windowType*)

where *windowVariable* is a variable of type window (or a descendant of window) and *windowType* is a string that specifies the type of window.

The above code opens three windows: an instance of w_employee, an instance of w_customer, and an instance of w_sales.

For more about this form of the Open function, see the *Function Reference*.

Using arrays or reference variables

How do you know whether to use reference variables or arrays to manipulate window instances?

Reference variables are easy, and PowerBuilder manages them automatically, just as it manages other data types. But you cannot manipulate particular instances of windows created using reference variables.

With arrays, you can refer to particular instances, but arrays are more difficult to use. You must manage them yourself. For example, say the user closed the second window in the array, then wants to open a new window. You need to code whether to add a window to the end of the array (thereby using more memory than needed) or find an empty slot in the existing array for the new window.

So if you need to refer to instances, use arrays. Otherwise, use reference variables.

Referencing entities in descendants

When you declare a variable whose data type is a kind of object, such as a window, you can use the variable to reference any entity defined in the object, but not in one of its descendants. Consider the following code:

```
w_customer     mycust

Open(mycust)
// The following statement is legal if
// w_customer window has a st_name control.
mycust.st_name.text = "Joe"
```

Mycust is declared as a variable of type w_customer (that is, mycust is a w_customer window). If w_customer contains a StaticText control named st_name, then the last statement shown above is legal.

However, consider the following case:

```
window      newwin
string      winname = "w_customer"

Open(newwin, winname)
// Illegal because objects of type Window
// do not have a StaticText control st_name
newwin.st_name.text = "Joe"
```

Here, newwin is defined as a variable of type window. PowerBuilder rejects the above code because the compiler uses what is called **strong type checking**: the PowerBuilder compiler does not allow you to reference any entity for an object that is not explicitly part of the variable's *compile-time* data type.

Because objects of type window do not contain a st_name control, the statement is not allowed.

You would either need to change the declaration of newwin to be a w_customer (or an ancestor window that also contains a st_name control), such as:

```
w_customer  newwin
string      winname = "w_customer"

Open(newwin, winname)
// Legal now
newwin.st_name.text = "Joe"
```

Or define another variable, of type w_customer, and assign it to newwin, such as:

```
window          newwin
w_customer      custwin
string          winname = "w_customer"

Open(newwin, winname)
custwin = newwin
// Legal now
custwin.st_name.text = "Joe"
```

CHAPTER 7

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

Overview of controls

There are two types of controls:

- Controls that have events.
- Controls that do not have events. These are group boxes, which you can use to group buttons, and the **drawing objects**.

About controls with events

Users can act on controls that have events. You write scripts that determine the processing that takes place when an event occurs in the control.

The controls that have events are:

CheckBox	OLE 2.0
CommandButton	Picture
DataWindow	PictureButton
DropDownListBox	RadioButton
EditMask	SingleLineEdit
Graph	StaticText
HScrollBar	User Object
ListBox	VScrollBar
MultiLineEdit	

Creating user objects
If you often use a control or set of controls with certain properties (such as a group of related radio buttons), you might want to create a user object, which is one or more controls with predefined behavior that can be placed in a window.

For more about user objects, see Chapter 10, "Working with User Objects."

About the drawing objects

You use the drawing objects to enhance the design of your window. The drawing objects are:

Line	Rectangle
Oval	RoundRectangle

Placing controls in a window

❖ To place a control in a window:

1 Select the control from the Window painter PainterBar or from the Controls menu.

If you select a user object control, the Select User Object dialog box displays listing all user objects defined for the application. Select the user object, then continue.

2 Click where you want the control.

The control displays at the location.

After you place 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 Edit➤Duplicate from the menu bar once for each duplicate control you want to place in the window.
>
> The controls are placed one under another. You can drag them to another location if you want.

Placing DataWindow controls, Pictures, and PictureButtons

When you place a DataWindow control, a picture, or a PictureButton in a window, you are placing a control in the window. No DataWindow object or picture has yet been specified for the control.

If the control is a	You see
DataWindow control	An empty box to indicate no DataWindow object has been specified
Picture	The dotted outline of a box to indicate no picture has been specified
PictureButton	A large button (resembling a CommandButton) with space at the top to indicate no picture has been specified

For more about using DataWindow objects and controls, see Part Four, "Working with Databases."

Pictures and PictureButtons are discussed later in this chapter.

Placing OLE 2.0 controls

You can place objects from applications that support OLE 2.0, such as Excel worksheets and Visio drawings, in your windows.

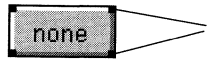
For information about using OLE 2.0 with PowerBuilder, see *Building Applications*.

Selecting controls

❖ To select a control:

- Click it.

  The control displays with handles on it. Previously selected controls are no longer selected.



Handles indicate that control is selected

❖ To select neighboring multiple controls:

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.

   PowerBuilder displays a bounding box.
3. Release the mouse button.

   All the controls in the region are selected and have handles.

❖ To select non-neighboring multiple controls:

1. Click the first control.
2. Press and hold the CTRL key and click additional controls.

   All the controls are selected and have handles.

Acting on multiple controls

You can act on all 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.

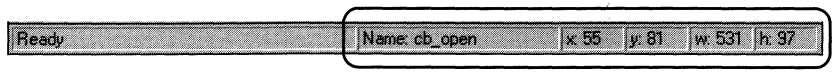
Tips

You can also display a list of controls and select one from the list by selecting Edit➤Control List from the menu bar.

You can select all controls by selecting Edit➤Select All from the menu bar.

Information displayed about the selected control

The name, X and Y coordinates, width, and height of the selected control is displayed in the MicroHelp bar:



If multiple objects are selected, *Group Selected* displays in the Name area and the coordinates and size do not display.

Defining a control's attributes

Just like the window object, each control has attributes that determine how the control looks and behaves during execution (the control's **style**).

The easiest way to define a control's attributes is by using the control's popup menu.

❖ To define a control's attributes using a popup menu:

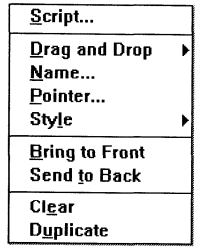
1 Display the popup menu as usual.
or
Select Design➤Control Style from the menu bar.

The popup menu for the selected control displays.

2 Use the menu to change the control's attributes.

About the popup menu

Each type of control has its own set of attributes and thus its own popup menu. Here is the menu for a CheckBox:



You select items on the popup menu to change the control's definition.

Each control is described later in this chapter.

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. The default prefixes are set in the Preferences painter and saved in the PB.INI file.

The initial default prefix for each control is listed below (note that there is no prefix for a window):

Control	Prefix
CheckBox	cbx_
CommandButton	cb_
DataWindow	dw_
DropDownListBox	ddlb_
EditMask	em_
Graph	g_
GroupBox	gb_
HScrollBar	hsb_
Line	ln_
ListBox	lb_
MultiLineEdit	mle_
OLE 2.0	ole_
Oval	oval_

Control	Prefix
Picture	p_
PictureButton	pb_
RadioButton	rb_
Rectangle	r_
RoundRectangle	rr_
SingleLineEdit	sle_
StaticText	st_
User Object	uo_
VScrollBar	vsb_

Changing the default prefixes

You can change the default prefixes for controls in the Window section of the Preferences painter.

For more about the Preferences painter, see Chapter 24, "Customizing PowerBuilder."

Changing the name

You should change the default suffix to a suffix that is meaningful in your application and develop a naming convention for control names.

For example:

Instead of	You could use
cb_6	cb_retrieve
cbx_1	cbx_enabled
dw_1	dw_EmployeeData
sle_2	sle_LName

Using these conventions makes it much easier for you to understand scripts you write to manipulate 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:

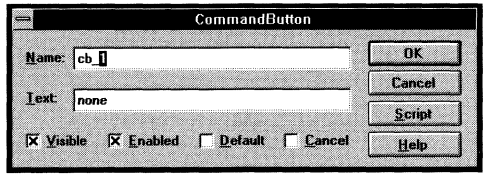
1 Display the control's popup menu, as described above.

2 Select Name from the menu.

The control's Style dialog box displays with the default suffix selected.

> **Shortcut**
> You can also display a control's Style dialog box by double-clicking the control.

Here is the Style dialog box for a CommandButton:



3 Type the application-specific suffix. Your typing replaces the selected default suffix.

You can use any valid PowerBuilder identifier with up to 40 characters.

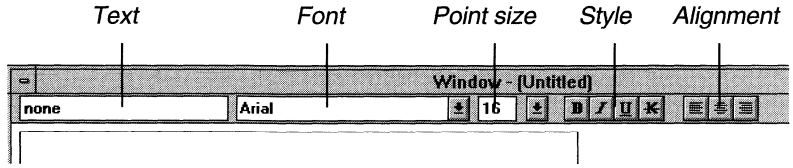
For information about PowerBuilder identifiers, see *PowerScript Language.*

4 Click OK.

Changing text

The Window painter has a StyleBar that you can use to change the following properties of selected controls' text:

- The text itself
- The font, point size, and characteristics such as bolding
- The alignment of text within the control (except that text in CommandButtons is always center aligned)



❖ To change text attributes of controls:

1 Select each control whose attributes you want to change.

2 Specify the new attributes using the StyleBar.

Choosing fonts
Make sure that fonts you pick will be available to your users when you distribute your application. If you pick a font that is on your computer but not on your user's computer, the operating environment will use the font it considers closest to the one you specified. It might not look very good.

How text size is stored

A control's text size is specified in the control's TextSize attribute. PowerBuilder saves the text size in points, using negative numbers. For example, if you define the text for the StaticText control st_prompt to be 12 points, PowerBuilder sets the value of st_prompt's TextSize attribute to –12. (PowerBuilder uses negative numbers to record point size for compatibility with previous releases, which saved text size in pixels as positive numbers.)

So if you want to change the point size of text during execution in a script, remember to use a negative value. For example, to change the point size for st_prompt to be 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

You can move or resize a control using the mouse or the keyboard.

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 it with the mouse.

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, 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

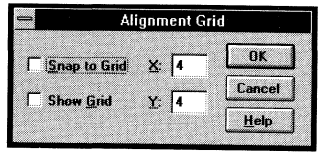
Using the grid

The Window painter provides a grid to help you align controls.

❖ To use the grid:

1 Select Design➤Grid from the menu bar.

The Alignment Grid dialog box displays.



2 Use thc 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

The grid options are:

Option	Meaning
Snap to Grid	If selected, controls snap to the grid when you place or move them
Show Grid	If selected, the grid is displayed in the workspace
X	The width of each cell in the grid in pixels
Y	The height of each cell in the grid in pixels

Tips

You can specify a default grid size in the Preferences painter.

Window painting is slower when the grid is displayed, so you might want to display the grid only when necessary.

Aligning controls

It is easy to align two or more controls.

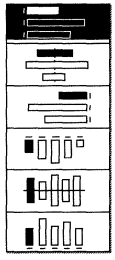
❖ To align controls:

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 Edit➤Align Controls 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.

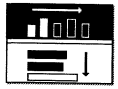
   PowerBuilder aligns all the selected controls with the first one.

Equalizing the space between controls

You can manually move controls by dragging them with the mouse. You can also easily equalize the spacing around specified controls.

❖ To equalize the space between controls:

1. Select the two controls whose spacing is correct (select one control, then press and hold CTRL and click the second control).
2. Select the other controls whose spacing you want to be the same as the first two controls by pressing CTRL and clicking.



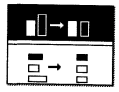
3. Select Edit➤Space Controls from the menu bar.
4. From the cascading menu, select the dimension whose spacing you want to equalize.

Equalizing the size of controls

Assume you have several CommandButtons and want them to be the same size. You can accomplish this manually or by using the Edit menu.

❖ To equalize the size of controls:

1. Select the control whose size is correct.
2. Select the other controls whose size you want to match the first control by pressing and holding CTRL and clicking.



3. Select Edit➤Size Controls from the menu bar.
4. From the cascading menu, select the dimension whose size you want to equalize.

Copying controls

You can copy controls within a window or to other windows. All attributes of the control, as well as all of its scripts, are copied. You can use this technique to easily make a copy of an existing control and change whatever you want in the copy.

❖ To copy a control:

1 Select the control in the Window painter workspace.

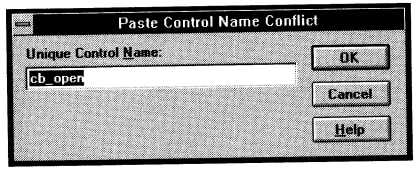
2 Select Edit➤Copy from the menu bar.
or
Press CTRL+C.

The control is copied to a private PowerBuilder clipboard.

3 To copy the control within the same window, select Edit➤Paste from the menu bar or press CTRL+V.

To copy the control to another window, open another instance of the Window painter and open the desired window in it. Make that window active and select Edit➤Paste 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, you should move the pasted control so it doesn't 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.

> **About 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 the control is from the top of the window (Y)
- The distance the control is 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 these controls.

> **Default tab values**
> The default tab value for drawing objects and RadioButtons in a GroupBox is 0 (skip the 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.

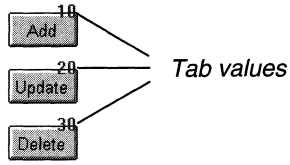
For example, if the tab values for controls A, B, and C are 30, 10, and 20 respectively and you add control D between control A and B, PowerBuilder assigns control D the tab value 40.

Changing the window's tab order

❖ **To change the tab order:**

1 Select Design➤Tab Order from the menu bar.

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 (0-9999); 0 removes the control from the tab order (skips the control). It doesn't 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 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.

> **Tips**
> To prohibit modifications in an edit box or selection of a control, set its tab value to 0.
>
> To permit tabbing in a group box, change the tab value of the GroupBox to 0, then assign nonzero tab values to the controls in the GroupBox.

4 Repeat the procedure until you have the tab order you want.

5 Select Design➤Tab Order from the menu bar again.

PowerBuilder saves the tab order.

Each time you select Tab Order, PowerBuilder renumbers the tabs to include any controls that have been added to the window and to allow space to insert new controls in the tab order.

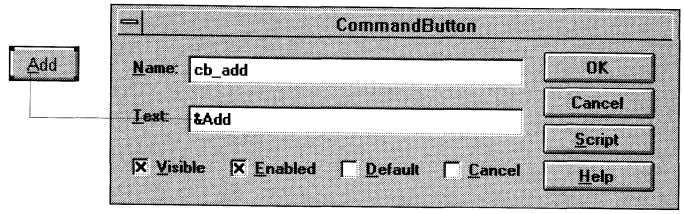
Defining accelerator keys

You can define accelerator keys for your controls to allow users to change focus to the control by pressing ALT+*AcceleratorKey*. How you do it depends on whether the type of control has displayed text associated with it.

❖ To define an accelerator key for a CommandButton, CheckBox, or RadioButton:

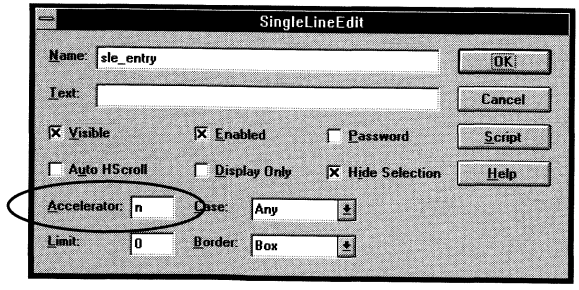
- When defining the text that displays for the control, precede the accelerator key with an ampersand character (&).

  PowerBuilder displays an underline to indicate the accelerator key.



❖ To define an accelerator key for a SingleLineEdit, MultiLineEdit, ListBox, or DropDownListBox:

1. Double-click the control to display the Style dialog box.
2. Type the letter of the accelerator key in the Accelerator box. For example, to make ALT+N the accelerator for the control, type **n** in the Accelerator box.



3. Click OK.

At this point you have defined the accelerator key, but the user has no way of knowing it. So next you need to label the control.

4 Place a StaticText control next to the control you just assigned the accelerator key. When defining the text for the StaticText control, precede the accelerator key with an ampersand character (&).

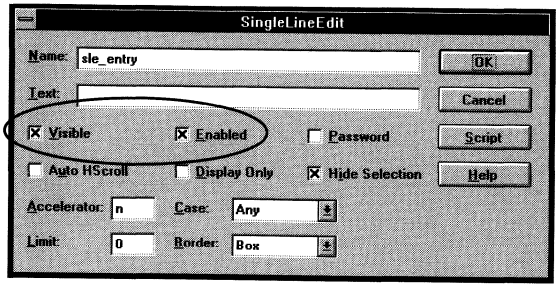
PowerBuilder displays an underline to indicate the accelerator key.

Now your user knows that there is an accelerator key associated with the control.

Specifying accessibility of controls

Controls have two boolean attributes that affect accessibility of the control:

- Visible
- Enabled



Using the Visible attribute

If the Visible attribute is selected, the control displays in the window. If you want a control to be initially invisible, do not select this attribute in the Window painter.

Hidden controls do not display by default in the Window painter. To display hidden controls in the painter, select Design➤Show Invisibles from the menu bar.

To display a control during execution, assign the TRUE value to the Visible attribute, such as:

controlname.Visible = TRUE

Using the Enabled attribute

If the Enabled attribute is selected, the control is active. For example, an enabled CommandButton can be clicked.

If you want a control to display but be inactive, do not select the Enabled attribute. For example, a CommandButton might be active only after the user has selected an option. In this case, display the CommandButton initially disabled (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 a ColorBar, which displays colors that you can use for components of the window.

Initially, the ColorBar displays these color selections:

- 16 predefined colors
- Window color (labeled w)
- Window Text color (labeled wt)
- Application Workspace color (labeled a)
- Four custom colors (labeled c)

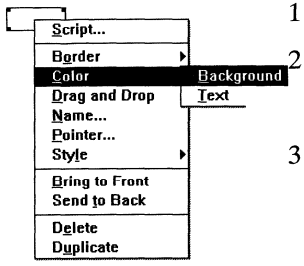
The Window, Window Text, and Application Workspace colors are those defined by the user in the Windows Control Panel. So if you use these colors in your window, the window colors will change to match the user's settings during execution.

Selecting colors

There are two ways to assign colors to controls.

> **About CommandButtons**
> You cannot change the colors for CommandButtons.

❖ To assign a color using the popup menu:



1. Place the mouse pointer over the control and display the popup menu.
2. Select Color from the popup menu.

   A cascading menu displays.
3. Choose the part of the control you want to specify a color for and select the color.

❖ To assign a color using the ColorBar:

1 Select the control.

2 Click either the left or right mouse button on the appropriate color in the ColorBar, as described next.

Selecting colors from the ColorBar

You can select two colors from the ColorBar for a window component: one by clicking the left mouse button and the other by clicking the right mouse button.

When you select a color, the color displays in the first space in the ColorBar. The color you select by clicking the left mouse button displays in the center. The color you select with the right mouse button displays in the border area.

The meaning of the colors depends on the type of control:

Type of control	Left mouse color	Right mouse color
Control with event	Text color	Background color
Drawing object	Fill color (the color within the object); if the object has a pattern, the background color for the pattern	The outline color; if the object has a pattern, the color of the pattern

Letting users change colors
You can also let *users* change colors in an application by providing a ColorBar in a window. The easiest way to do that is with a user object.

For more information, see Chapter 10, "Working with User Objects."

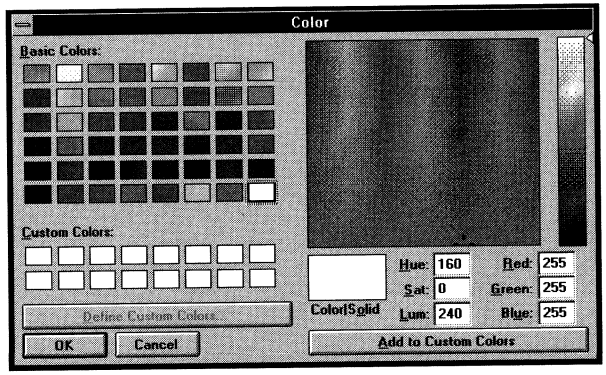
Defining your own colors

You can define your own custom colors for use in windows, user objects, and DataWindow objects.

❖ To maintain your custom colors:

1 Double-click in a custom color area in the ColorBar.

The Color dialog box displays.



2 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.

3 When you have the color you want, click Add to Custom Colors.

The new color displays in the list of custom colors.

4 Select the new color in the list of custom colors.

5 Click OK.

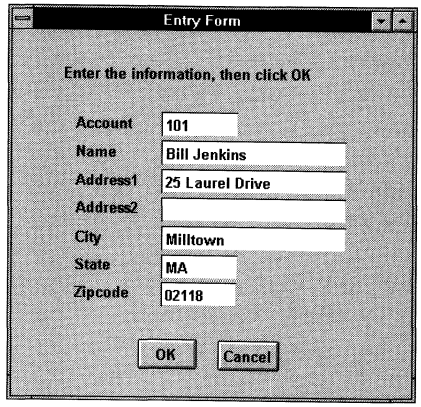
The new color displays in the ColorBar and is available in all windows, user objects, and DataWindow objects you create.

PowerBuilder saves custom colors in the [Colors] section of the PB.INI file, so they are available across sessions.

Using the 3D look

Applications today commonly have a three-dimensional look and feel. By selecting a 3D border for your SingleLineEdit boxes and other controls, and by making windows gray, your applications can too.

Edit boxes are using the 3D Lowered border



❖ To have the 3D look used by default:

- Select Options➤Default to 3D from the menu bar.

  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 PB.INI, so the preference is maintained across sessions.

Using the individual controls

There are four basic types of controls.

Controls that invoke actions These controls include CommandButtons and Picture Buttons.

Controls that display data These controls include ListBoxes, DropDownListBoxes, DataWindow controls, StaticText, Graphs, Pictures, SingleLineEdits, MultiLineEdits, EditMasks, and OLE 2.0 controls.

Controls that indicate choices These controls include RadioButtons and CheckBoxes. You can group these controls in a GroupBox.

Controls that are decorative only These controls are the drawing objects (Line, Rectangle, RoundRectangle, and Oval).

How to use the controls

You should use the controls as described above. For example, users expect radio buttons to only select an option. Don't use a radio button to also invoke an action, such as opening a window or printing. Use a command button for that.

There is one exception: listboxes are often used both to display data and to invoke actions. For example, double-clicking a listbox item often causes some action to occur.

The following sections describe some features that are unique to the individual controls.

Not covered here
This chapter does not cover the following:

DataWindow controls and objects—described in Part Four, "Working with Databases."

User objects—described in Chapter 10, "Working with User Objects."

Graph controls—described in Chapter 18, "Working with Graphs."

OLE 2.0 controls—described in *Building Applications*.

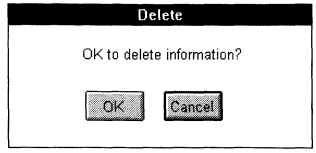
For more information

To learn more about the proper use of controls, consult a style guide that presents guidelines for your target environment.

For a list of events, attributes, and functions for each control, see *Objects and Controls*.

Using CommandButtons

CommandButtons are used to carry out an action. For example, you can use an OK button to confirm a deletion or a Cancel button to cancel a requested deletion.



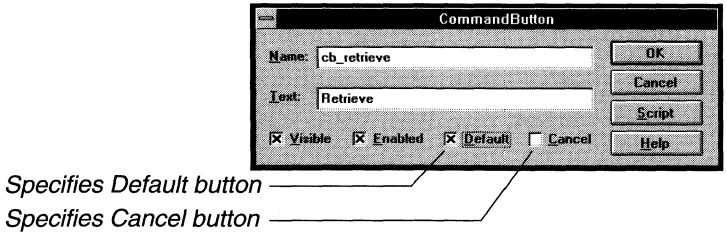
Place related CommandButtons along the bottom of the window if there are not too many. If there are a lot, place them along the right side of the window.

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 define a CommandButton as being the default button in a window: select Default from the Style cascading menu on the control's popup menu or select the Default box in the control's Style dialog box.



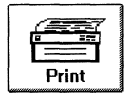
If you define a default CommandButton, the user's pressing the ENTER key when the focus is not on another CommandButton is the same as clicking the default button (that is, the default button's Clicked event is triggered). If the focus is on another CommandButton, pressing ENTER is the same as clicking the button that has focus.

A bold border is placed around the default CommandButton (or the button with focus if the user explicitly tabs to a CommandButton).

Similarly, you can define a CommandButton as being the cancel button by selecting Cancel from the Style cascading menu on the control's popup menu or by selecting the Cancel box in the Style dialog box. If you define a cancel CommandButton, the user's pressing the ESC key is the same as clicking the cancel CommandButton (that is, its Clicked event is triggered).

Using PictureButtons

PictureButtons are PowerBuilder-specific controls that are identical to CommandButtons in their functionality. The only difference is that you can specify a bitmap (BMP) file, runlength-encoded (RLE) file, or Aldus-style Windows metafile (WMF) to display in the button.



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 using a picture instead of just text.

❖ To specify a picture:

1 Place a PictureButton control in the window.

2 Select Change Enabled or Change Disabled from the control's popup menu, depending on which picture you want to specify.

The Select Picture dialog box displays.

3 Choose the file you want to display. Initially, BMP files are shown. Change to another file type by changing the file type at the bottom of the window.

4 Click OK.

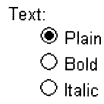
If the PictureButton is defined as initially enabled, the enabled picture displays in the Window painter. If the PictureButton is defined as initially disabled, the disabled picture displays in the painter.

Using RadioButtons

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:

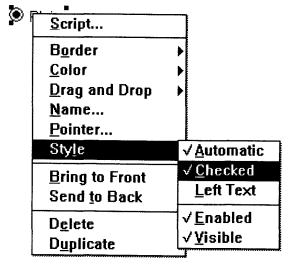


When the user clicks a RadioButton, it becomes selected and the previously selected RadioButton in the group becomes deselected.

Only 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 doing one of the following:

- Selecting Checked from the Style cascading menu in the control's popup menu

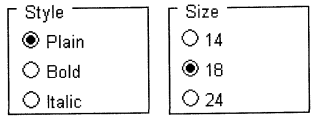


- Selecting Checked in the control's Style dialog box

Grouping RadioButtons

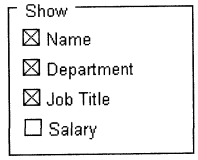
By default, all RadioButtons in a window are in one group, no matter what their location in the window. Only one RadioButton can be selected at a time.

You use a GroupBox control to group related RadioButtons. All RadioButtons inside a GroupBox are considered to be in one group. For example, in the following example, there are two groups of RadioButtons that are independent of each other. One button is selected in each group.



Using CheckBoxes

CheckBoxes are square boxes used to set independent options. When they are selected, they contain an X; 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

Typically, CheckBoxes have two states, on and off. But sometimes you want to represent a third state, such as Unknown. The third state displays as a gray box.

For example, say you have a CheckBox that represents whether the selected customer is active. If there are several selected customers, and only some are active, you would use the third state for the CheckBox. That state indicates that the selection is neither active nor inactive, but a mixture.

❖ **To enable the third state:**

- Choose Three State from the Style cascading menu of the control's popup menu or from the control's Style dialog box.

  This specifies that the CheckBox has three states. When the user clicks the box in the application, it will cycle through the three states.

To specify that the button's current state is the third state (that is, the gray box), select Third State from the menu or in the Style dialog box.

Using StaticText

You use a StaticText control to display text to the user or to describe a control that doesn't have text associated with it, such as a listbox or edit box.

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 attribute).

StaticText controls have events associated with them, but you will probably seldom write scripts for them because users don't expect to interact with static text.

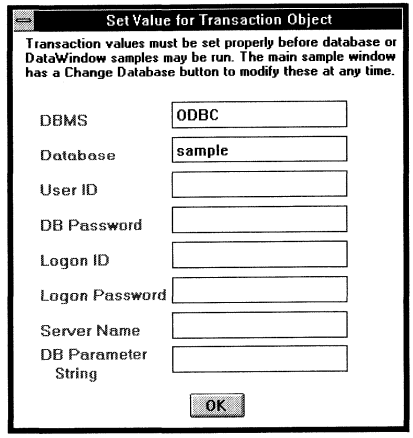
Indicating accelerator keys

One use of a StaticText control is to label a listbox or edit box. If you assign an accelerator key to a listbox or edit box, 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 box). This technique is described in "Defining accelerator keys" on page 195.

Using SingleLineEdits and MultiLineEdits

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 attributes, including:

- Whether the box has a border (the Border attribute)
- Whether the box automatically scrolls as needed (AutoHScroll and, for MultiLineEdits, AutoVScroll)
- For SingleLineEdits, whether the box is a Password box—whether to display entry using asterisks instead of the actual entry (Password)
- In which case to accept and display entry (TextCase)
- Whether the selection displays when the control does not have focus (Hide Selection)

For more information about attributes of these controls, click the Help button in the control's Style dialog box.

Using EditMasks

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:

(###) ###-####

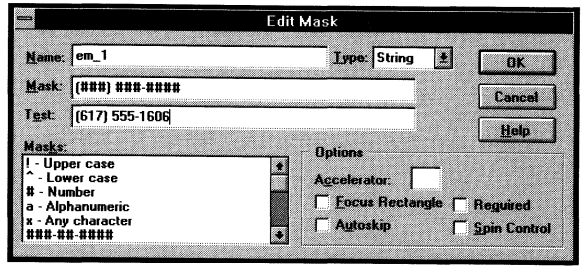
During execution, the punctuation characters (the parentheses and dash) display in the box and the cursor jumps over them as the user types.

❖ To use an EditMask control:

1 Click the EditMask button in the PainterBar and click where you want the control to be.

2 Double-click the control or select Name from its popup menu.

The EditMask Style dialog box displays.



3 Name the control.

4 In the Type dropdown listbox, specify the type of data that users will be entering into the control.

5 Specify the mask. The special characters used in the mask for the specified data type display in the Masks box. The preceding dialog box shows an edit mask for phone numbers, which are strings. As you can see in the Masks box, ! represents an uppercase letter, ^ represents a lowercase letter, # represents a number, and so on.

About masks
Masks in EditMask controls in windows work the same as masks in display formats and in the EditMask edit style in DataWindows.

For more information about specifying masks, see the discussion of display formats in Chapter 15, "Displaying and Validating Data."

6 (Optional) Test the mask by typing a value in the Test box.

7 Specify other attributes for the EditMask control. For information on the other attributes, click the Help button.

8 Click OK.

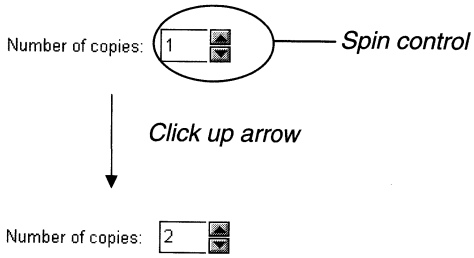
Keyboard behavior

Note the following about how certain keystrokes behave in edit masks.

- Pressing BACKSPACE always deletes the previous character, even if SHIFT is pressed.
- Pressing DELETE deletes everything that is selected.
- Non-numeric edit masks treat any characters that don't match the mask pattern as delimiters.
- With Date, Datetime, and Time edit masks, if the user types a non-zero number in the first position of dd, mm, hh, ss, or mm (minutes) and then types the delimiter character, the typed digit is entered as the second position and the first position is set to 0. For example, typing **1/1/92** results in **01/01/92**.
- With Date, Datetime, and Time edit masks, if the user types a number in the first position that is greater than the maximum day, month, hour, and so on, the typed number is entered as the second position and the first position is set to 0. The cursor is positioned in front of the second position.

Using spin controls

You can define an EditMask as a **spin control**, which is an edit box 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. The control would look like this:

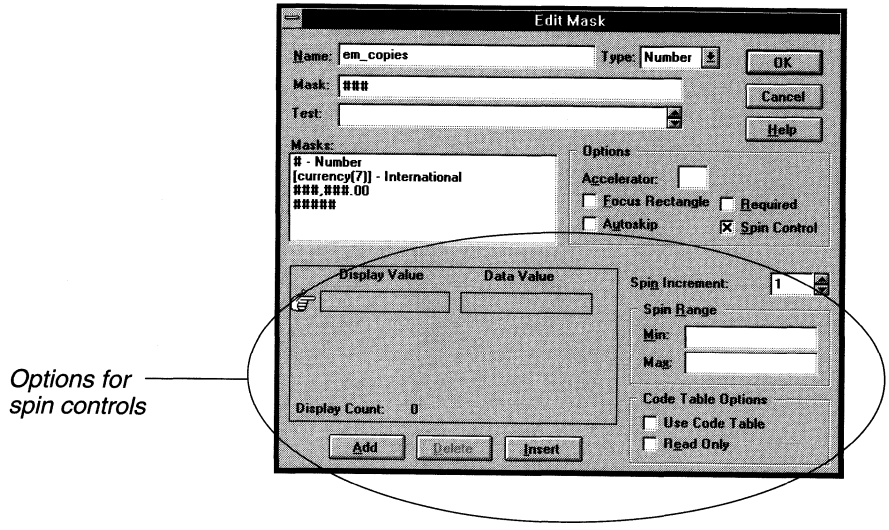


❖ To define an EditMask as a spin control:

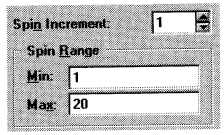
1 Name the EditMask and provide the data type and mask, as described above.

2 Select the Spin Control box in the Options group.

Options for spin controls display.



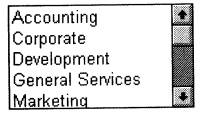
3 Specify the needed information. For example, to allow users to select a number from 1 to 20 in increments of 1, specify a spin range of 1 to 20 and a spin increment of 1.



For more information on the options for spin controls, click the Help button in the EditMask Style dialog box.

Using ListBoxes

A ListBox displays available choices. You can specify that ListBoxes have scrollbars 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 File Open box, clicking a filename in a listbox selects the file. Double-clicking a name opens the file.

PowerBuilder automatically selects (highlights) an item when a user selects it during execution. If you want something to happen when users *double-click* an item, you must code a script for the control's DoubleClicked event (note that the Clicked event is always triggered before the DoubleClicked event).

Populating the list

In the Window painter, you type the values for the list in the control's Style dialog box. Press CTRL+ENTER to go to the next line.

Changing the list during execution
To change the items in the list during execution, use the functions AddItem, DeleteItem, and InsertItem.

For more information, see the *Function Reference*.

Setting tab stops

You can set tab stops for text in ListBoxes (and in MultiLineEdits) by setting the TabStop attribute. You can define up to 16 tab stops (the default is a tab stop every eight characters). 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 (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 attributes

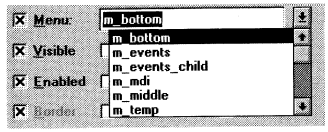
For ListBoxes, you can specify whether:

- Items in the ListBox are displayed in sorted order
- The ListBox allows the user to select multiple items
- The ListBox displays scrollbars if needed

For more information, click the Help button in the ListBox Style dialog box.

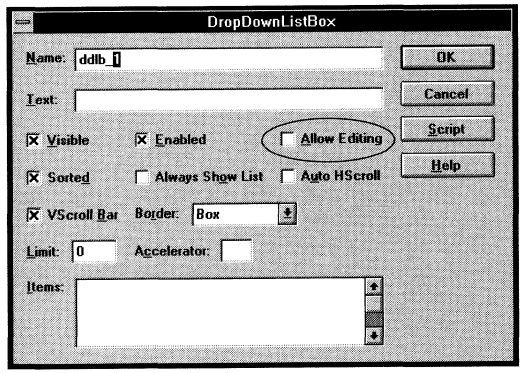
Using DropDownListBoxes

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.

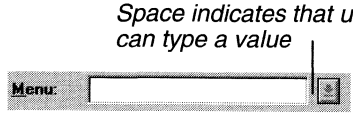
There are several ways for users to pick an item from a noneditable DropDownListBox:

- 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 beginning with the character.
- Click the down arrow to the right of the edit box to display the list, then select the one you want.

Editable boxes

If you want to give your user the option of specifying a value that is not in the list, make the DropDownListBox editable.

In editable DropDownListBoxes, there is a space displayed between the edit box and the down arrow.



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, described above.

Specifying the size of the dropdown 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 dropdown box—is shown.

Other attributes

As with ListBoxes, you can specify whether the list is sorted and whether the edit box is scrollable.

For more information, click the Help button in the DropDownListBox Style dialog box.

Using pictures

Pictures are PowerBuilder-specific controls that display a bitmap (BMP) file, runlength-encoded (RLE) file, or Aldus-Style Windows metafile (WMF).

❖ To display a picture:

1 Place a picture control in the window.

2 Select Change Picture from the control's popup menu.

The Select Picture dialog box displays.

3 Choose the file you want to display. Initially, BMP files are shown. Change to another file type by changing the file type at the bottom of the dialog box.

4 Click OK.

The picture displays.

You can choose to resize or invert the image.

Be careful how you use picture controls. They can be used for almost any purpose. They have events, so users can click on them. Or they can be used simply to display an image. Be consistent in their use so users know what they can do with them.

Using drawing objects

PowerBuilder provides the following drawing objects: line, oval, rectangle, and RoundRectangle.

These objects have no events associated with them. They are for display purposes only. Use them to make your window more attractive or to group controls.

You can use the following functions to manipulate drawing objects during execution:

Hide
Move
Resize
Show

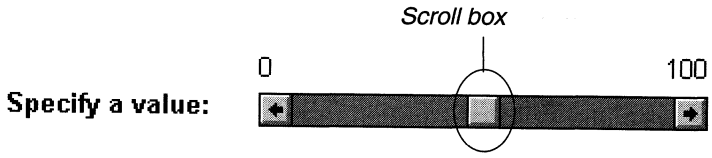
In addition, each drawing object has a set of attributes that define its appearance. You can assign values to the attributes in a script to change the appearance of a drawing object.

Using HScrollBars and VScrollBars

You can place freestanding scrollbar controls within a window. Typically, you use these controls to do one of the following:

- Act as a slider control for users to be able to specify a continuous value
- Graphically display information to the user

For example:



You can set the position of the scroll box by specifying the value for the control's Position attribute. When the user drags the scroll box, the value of Position is automatically updated.

CHAPTER 8

Understanding Inheritance

About this chapter

This chapter describes how to use inheritance to build PowerBuilder objects.

Contents

Overview of inheritance

One of the most powerful features of PowerBuilder is **inheritance**. It enables you to build windows, user objects, and menus that are derived 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 manually make 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 get consistency in the code and objects in your applications.

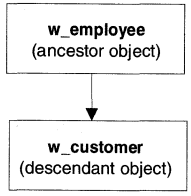
This chapter describes how inheritance works in PowerBuilder and how to use it to maximize your productivity.

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 descendant objects. Chapter 6, "Defining Windows," used the example of creating a window w_customer that inherits its properties from the existing window w_employee. In this example, w_employee is the ancestor and w_customer is the descendant.

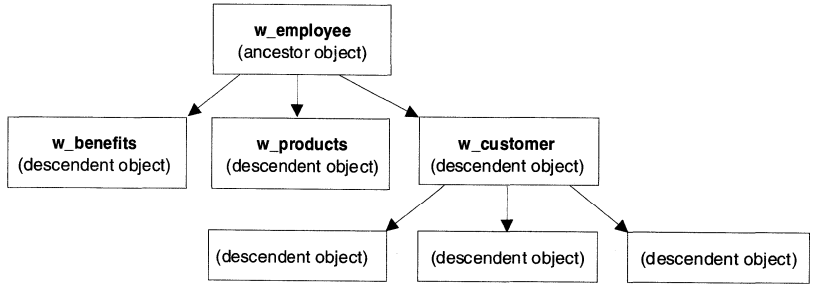
The object at the top of the hierarchy is a base class object, and the other objects are descendants of this object. Each descendant inherits information from its ancestor. The base class object typically performs generalized processing, and each descendant modifies the inherited processing as needed.

In the example, the w_employee window is the base class window. The hierarchy looks like this:



Multiple descendants

An object can have an unlimited number of descendants and each descendant can also be an ancestor. For example, if you build several windows that are direct descendants of the w_employee window and three windows that are direct descendants of the w_customer window, the hierarchy looks like this:



Viewing the hierarchy

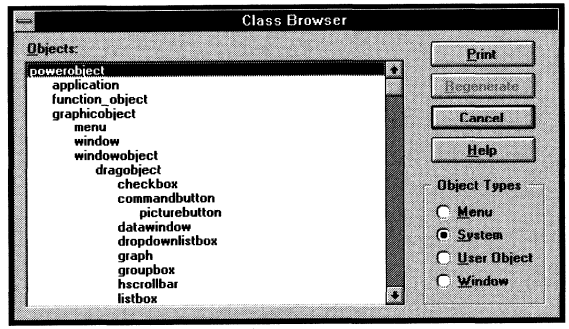
PowerBuilder provides a Class browser that shows the hierarchy of windows, menus, and user objects.

❖ To view the hierarchy:

1 Open the Library painter.

2 Select Utilities➤Browse Class Hierarchy from the menu bar.

The Class browser displays the system hierarchy.



3 Click a radio button to see the hierarchy for the corresponding object type.

PowerBuilder shows all objects of the chosen type in the current application. Inherited objects are shown indented under their parents.

Working with inherited objects

This section describes the following:

- Working in a descendent object
- Working in an ancestor object
- Resetting attributes in a descendant

Working in a descendant

You can change descendent objects to meet their specialized needs. For example, you can:

- Change attributes of the descendent object
- Change attributes of inherited controls in the object
- Add controls to a descendent window or user object
- Add MenuItems to a menu

For specifics about what you can do in inherited windows, user objects, and menus, see Chapter 6, "Defining Windows," Chapter 10, "Working with User Objects," and Chapter 9, "Working with Menus."

Working in an ancestor

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 be careful when you change the definition of an ancestor object. You might 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. The easiest way to regenerate descendants of an object is to use the Class browser.

For information about regenerating objects, see Chapter 23, "Managing Libraries."

About local changes
If you change a property in an ancestor object, the property will also change in all descendants—*if* you haven't already changed that property in a descendant, in which case the property in the descendant stays the same. That is, local changes always override inherited properties.

Resetting a descendent object's attributes

When you use inheritance to build a window, menu, or user object, you can change the values of inherited attributes (such as the placement of controls in a window). If you change your mind, you can easily reset the attributes to the values they had in the ancestor.

❖ To reset the attributes to their values in the ancestor:

- Select Edit➤Reset Attributes from the menu bar in the Window, Menu, or User Object painter.

  The attributes are reset to the values they had in the ancestor.

Using inherited scripts

In the hierarchy formed by ancestors and descendants, each descendant inherits its scripts from its immediate ancestor. If the inherited event does not have a script, you can write a script for the event for the descendant. If the inherited event does have a script, the ancestor script will execute in the descendant by default. You can:

- Override the ancestor script—execute a descendent script *instead of* the ancestor script
- Extend the ancestor script—build a script that executes *after* 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 228.

You cannot delete or modify an ancestor script from within a descendant.

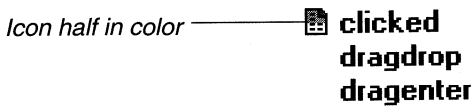
Viewing inherited scripts

When you open the PowerScript painter for an inherited object or control that has a script defined only in an ancestor, the workspace is blank: you do not see the ancestor script.

The PowerScript painter indicates which events have scripts written for an ancestor as follows:

- If the event has a script in an ancestor only, the script icon in the Select Event dropdown listbox is displayed in color (shown here in gray).

- If the event has a script in an ancestor as well as in the object you are working with, the icon is displayed half in color.

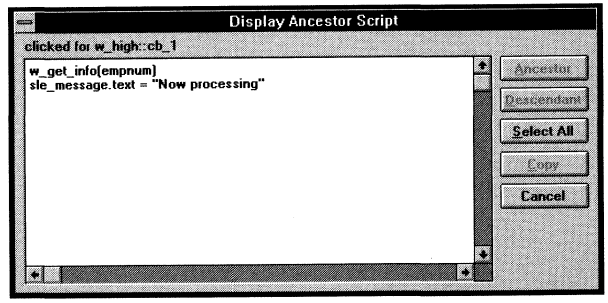


❖ To view the ancestor script:

1. Select the object or control whose ancestor script you want to view. To view an inherited script for the window, make sure no control is selected.
2. Open the PowerScript painter.
3. Select the event whose script you want to see.

   The workspace is blank; the script for the ancestor does not display.
4. Select Compile➤Display Ancestor Script from the menu bar.

   PowerBuilder displays in a dialog box the script defined in the closest parent (the object immediately above the current object in the hierarchy).



5. To climb the inheritance hierarchy, click the Ancestor button.

   PowerBuilder shows the script for the grandparent of the current object. By clicking Ancestor, you can traverse the entire inheritance hierarchy.
6. To copy the script to the clipboard, select the part you want, then click the Copy button. You can then paste the contents into another script.

Overriding a script

❖ **To override an ancestor script:**

1 Select the object or control whose ancestor script you want to override. To override a window-level script, make sure no control is selected.

2 Open the PowerScript painter.

3 Select the event for which you want to override the script.

4 Select Compile➤Override Ancestor Script from the menu bar.

5 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 228 or "Calling an ancestor function" on page 230.

Tip
To override a script for the ancestor, but not execute a script in the descendant, enter only a comment in the workspace.

During execution, 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.

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 Select the object or control whose ancestor script you want to extend. To extend a script for a window, make sure no control is selected.

2 Open the PowerScript painter.

The PowerScript painter displays and the workspace is blank; the script for the ancestor does not display.

3 Select the event for which you want to extend the script.

4 Make sure Extend Ancestor Script is selected on the Compile menu (it is the default).

5 Enter the appropriate statements in the workspace.

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 228 or "Calling an ancestor function" on page 230.

Example of extending a script

If the ancestor window script for the Clicked event in a button beeps when the user clicks the button without selecting an item in a listbox, you might extend the script in the descendant to display a message box in addition to beeping.

Calling an ancestor script

When you write a script for a descendent object or control, you can call scripts written for any ancestor.

In the following syntax, *ancestorwindow, ancestoruserobject,* and *ancestormenu* represent any ancestor of the descendent object; it is not restricted to the immediate ancestor (parent).

To reference the parent
To reference the immediate ancestor (parent), you can use the Super reserved word, such as:

```
call super::clicked
```

For more about Super, see *PowerScript Language*.

For window scripts

If the script you want to call is for an event in an ancestor window, use this syntax:

CALL *ancestorwindow* :: *event*

If the script you want to call is for an event in a control or user object in an ancestor window, use this syntax:

CALL *ancestorwindow* `*control* :: *event*
CALL *ancestorwindow* `*object* :: *event*

For user object scripts

If the script you want to call is for an event in an ancestor user object, use this syntax:

CALL *ancestoruserobject* :: *event*

If the script you want to call is for an event in a control or user object in an ancestor custom user object, use this syntax:

CALL *ancestoruserobject* `*control* :: *event*
CALL *ancestoruserobject* `*object* :: *event*

For menu scripts

If the script you want to call is for an event in a MenuItem, use this syntax:

CALL *ancestormenu*`*menuitem* :: *event*

When referring to a MenuItem in a dropdown or cascading menu, you must specify each MenuItem on the path to the MenuItem you are referencing, separating each name with a period.

Example of calling an ancestor script

The following script in w_customer (a window inherited from w_employee) checks the value in the SingleLineEdit sle_status and executes the ancestor script only if the value is "Active":

```
if sle_status.Text = "Active" then
   call w_employee`rb_review::clicked
else
   MessageBox("Incorrect Status",  &
      "Employee status must be Active",  &
      StopSign!, Cancel!)
end if
```

Continuing the example, assume that you want to change the script for the Exit button to display a message before closing the window. The message asks whether the user wants to update the database. If the answer is yes, the script should update the DataWindow control, then call the ancestor script to exit the application. The script would look like this:

```
int   Update

Update = MessageBox("Update?",  &
   "Update the data before exiting?",  &
   Question!, YesNo!)

if Update = 1 then Update(dw_data)
call w_employee`cb_exit::clicked
```

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 don't want to call the first one up, you need to specify the name of the object defining the function you want:

ancestorobject :: *function* (*arguments*)

Limitation

The second syntax, in which you directly specify the ancestor object, works only in scripts for the descendent object itself, not in scripts for *controls or user objects* in the descendent object or in MenuItem scripts.

To call a specific version of an ancestor user-defined function in a script for a control, user object, or MenuItem in a descendent object, do the following:

1 Define an object-level user-defined function in the descendant object that calls the ancestor function.

2 Call the function you just defined in the descendent script.

CHAPTER 9

Working with Menus

About this chapter

You add menus to windows to give your users an easy, intuitive way to select commands and options in your applications. This chapter describes how to define and use menus.

Contents

Overview of menus

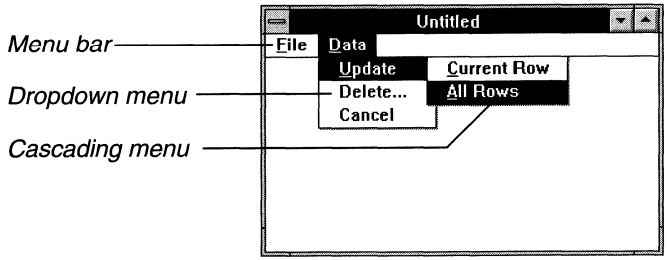
Menus are lists of commands or options (menu items) that a user can select in the currently active window. The items on the menus under the menu bar are usually related and provide the user with commands (for example, Open and Save As on the PowerBuilder File menu) or alternate ways of performing a task (for example, items on the Controls menu in the Window painter correspond to buttons in the PainterBar).

All windows in an application except child and response windows should have menus.

Menus you define in PowerBuilder work exactly the same as standard menus in your operating environment. For example, you can select items with the mouse or with the keyboard, and you can define accelerator keys for items.

About menus and MenuItems

Each choice in a menu is defined as a MenuItem in PowerBuilder. MenuItems display in a menu bar or in dropdown or cascading menus. A dropdown menu is a menu under an item in the menu bar. A cascading menu is a menu to the side of an item in a dropdown menu.



The preceding window shows two MenuItems on the menu bar (File and Data), three MenuItems on the Data menu (Update, Delete, and Cancel) and two MenuItems on the cascading menu under Update (Current Row and All Rows).

Using ellipsis points
By convention, ellipsis points (...) following a menu item indicate that clicking the item will display a dialog box.

Using menus

You can use menus you build in PowerBuilder in two ways:

- In the menu bar of windows, as shown above

  Window menus are associated with a window in the Window painter and display whenever the window is opened.

- As popup menus

  Popup menus display only when a script executes the PopMenu function.

Both uses are described in this chapter.

Designing menus

PowerBuilder gives you complete freedom as to how you design your menus. But you should follow conventions for your operating environment in order to make it easy to use your applications. For example, you should keep menus simple and consistent. You should group related items in a dropdown menu. You should use cascading menus sparingly and restrict them to one level.

This chapter describes some guidelines you should follow when designing menus, but a full discussion of these issues is beyond the scope of this manual. 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 MenuItems by setting their attributes.
- Build scripts that determine how to respond to events in the MenuItems. To support these scripts, you can declare functions, structures, and variables for the menu.

Two ways

There are two ways to build a menu. You can:

- Build a new menu from scratch

  The next section describes how to build menus from scratch.

- 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 you time and coding.

  "Using inheritance to build a menu" on page 256 describes how to use inheritance to build a menu.

Building a new menu

This section describes how to build menus from scratch. You will use this technique to create menus that aren't based on existing menus.

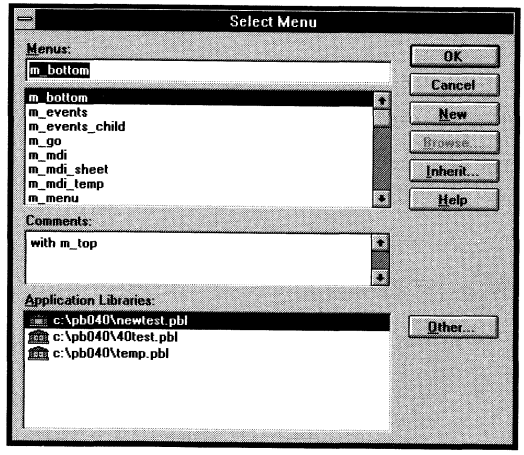
You use the Menu painter to build menus.

Opening the Menu painter

❖ To open the Menu painter:

1 Click the Menu painter button in the PowerBar or PowerPanel.

The Select Menu dialog box lists the menus in the current library.

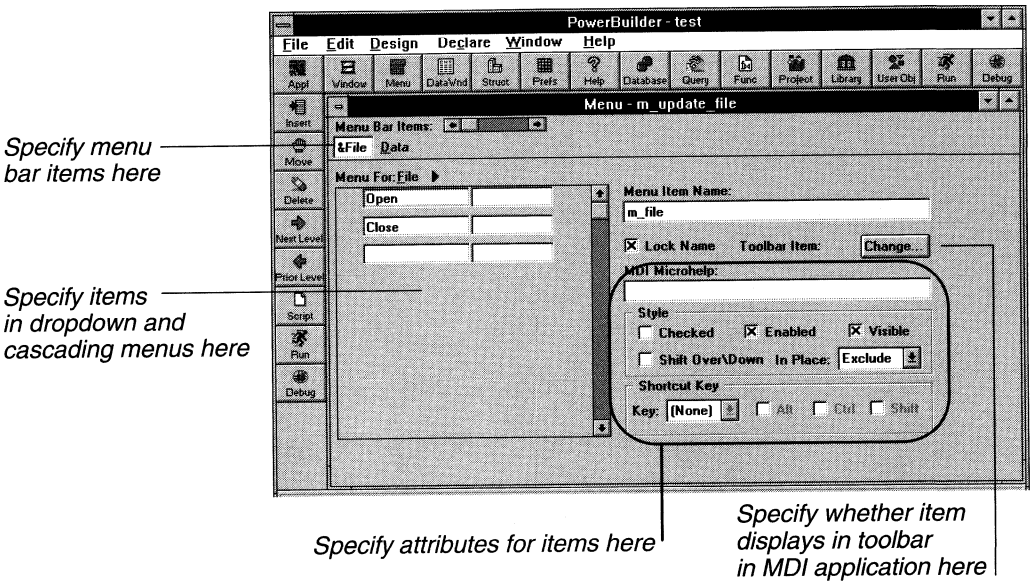


2 Click the New button to build a new menu.

The Menu painter workspace displays.

About the Menu painter

The Menu painter has several work areas.



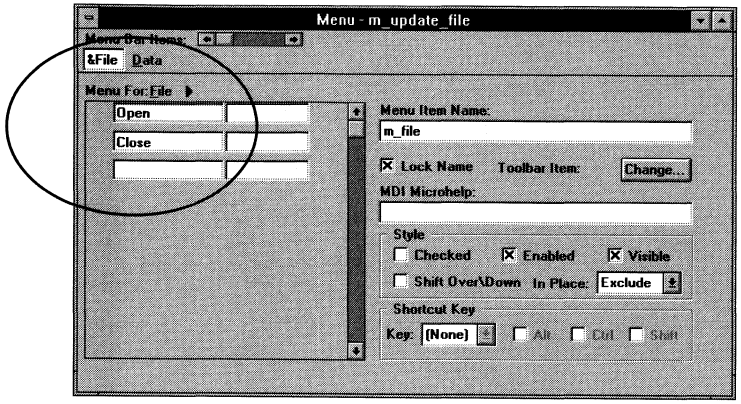
Working in the Menu painter

You can specify the following in the Menu painter:

- The MenuItems that display in the menu bar (the bar at the top of the window)
- The MenuItems that display under each item in the menu bar
- Attributes of the MenuItems (how they display)
- Accelerator and shortcut keys
- Scripts for MenuItem events

Adding MenuItems

Each menu consists of at least one MenuItem on the menu bar and MenuItems in a dropdown menu. The menu shown in the following painter screen has two items in the menu bar (File and Data). The dropdown menu under File has two items (Open and Close).



You can add MenuItems in three places:

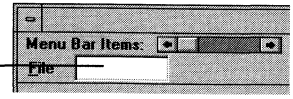
- To the menu bar
- To a dropdown menu
- To a cascading menu

❖ To add MenuItems to the menu bar:

1 Click the empty space to the right of the last defined MenuItem on the menu bar at the top of the painter workspace.

PowerBuilder displays an empty box.

Add item to menu bar here



2 Type the text that will display for the MenuItem.

3 To add another MenuItem to the menu bar, click to the right of the MenuItem you just defined.

PowerBuilder displays another empty box.

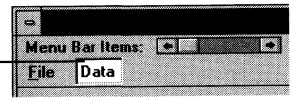
4 Type the text for the new MenuItem.

5 Repeat steps 3 and 4 to add additional items to the menu bar.

❖ To add MenuItems to a dropdown menu:

1 Click in the box for the item in the menu bar.

Click here to add items to dropdown menu for Data



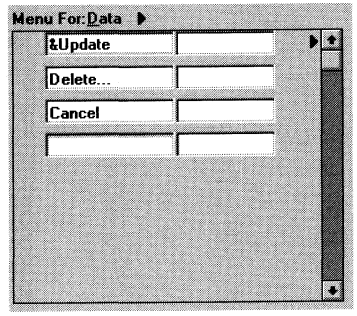
PowerBuilder displays the currently defined dropdown menu for the selected MenuItem.

2 Press TAB to go to the first empty box under the Menu For heading.

3 Type the text that will display for the MenuItem.

4 Repeat steps 2 and 3 to add additional items in the dropdown menu.

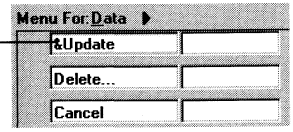
Dropdown menu has three items



❖ To add MenuItems to a cascading menu:

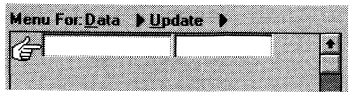
1 Click in the box for the item in a dropdown menu that you want to attach a cascading menu to.

Click here to add items to cascading menu under Update



2 Click the Next Level button.
or
Select Edit➤Next Level from the menu bar.

An empty box displays under the Menu For heading and the pointer moves to the box so you can build the cascading menu.



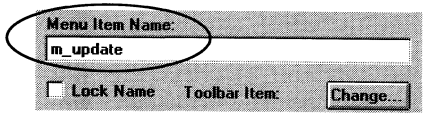
The Menu For heading displays the name of the MenuItem, an arrowhead, the MenuItem you selected, and another arrowhead to remind you that you are entering items in a cascading menu.

3 Enter items in the menu the same way you enter items in the dropdown menu.

4 To return to the previous menu level, click the Prior Level button or select Edit➤Previous Level from the menu bar.

How MenuItems are named

When you add a MenuItem, PowerBuilder gives it a default name, which displays in the Menu Item Name box.



This is the name by which you refer to a MenuItem in a script (for example, it corresponds to cb_*name* for a CommandButton).

About the default MenuItem names

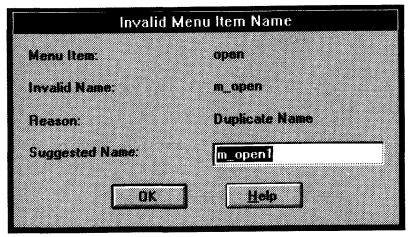
The default name is a concatenation of the text specified for the Prefix variable in the Menu section of the Preferences painter (initially the default prefix is m_) and the valid PowerBuilder characters and symbols in the text you typed for the MenuItem.

If there are no valid characters or symbols in the text you typed for the MenuItem, PowerBuilder creates a unique name *prefix_n*, where *n* is lowest number that can be combined with the prefix to create a unique name.

The complete MenuItem 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).

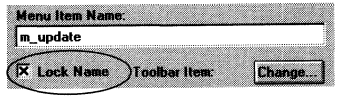
Duplicate MenuItem names

If you add a MenuItem to the menu and the name that PowerBuilder assigns has already been used in the menu, PowerBuilder displays a window and suggests a unique name for the MenuItem.



MenuItem names are locked by default

After you add a MenuItem and move to another item, the name PowerBuilder assigns to the MenuItem is locked.



Even if you later change the text that displays for the MenuItem, PowerBuilder will not rename the MenuItem. This is so you can change the text that displays in a menu without having to revise all your scripts that reference the MenuItem (remember, you reference a MenuItem through the name that PowerBuilder assigns to it).

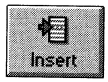
If you want to change the text that displays for a MenuItem and have PowerBuilder rename the MenuItem based on the new text, click the Lock Name checkbox to deselect it, then change the text that displays for the MenuItem.

Inserting MenuItems

You can insert MenuItems between existing MenuItems on the menu bar or in dropdown and cascading menus.

❖ To insert a MenuItem:

1 Select the menu bar item before which you want to insert an item.



2 Click the Insert button in the PainterBar.
or
Press the INS key.
or
Select Edit➤Insert from the menu bar.

An empty box displays.

3 Type the text of the new MenuItem.

Moving MenuItems

To change the order of items in the menu bar or in a dropdown or cascading menu, you can drag the items to the order you want within their current menu. You cannot drag items from one menu to another (for example, you cannot drag an item in the menu bar to a dropdown menu).

❖ To move a MenuItem:

1 Click the Move button in the PainterBar.
or
Select Edit➤Move from the menu bar.

You are now in Move mode.

2 Press and hold the left mouse button on the MenuItem you want to move.

The pointer becomes a hand pointer.

3 Drag the item to a new location in its menu.

4 Release the mouse button.

The MenuItem displays in its location and you leave Move mode.

Deleting MenuItems

❖ **To delete a MenuItem:**

1 Select the MenuItem you want to delete.

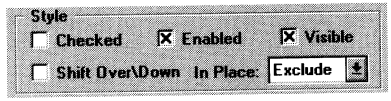
2 Click the Delete button in the PainterBar.
or
Select Edit➤Delete from the menu bar.

The selected item is deleted.

Defining the appearance of MenuItems

By selecting checkboxes in the Menu painter, you can specify how a MenuItem appears during execution. Each of the checkboxes corresponds to an attribute of a MenuItem.

For a list of all MenuItem attributes, see *Objects and Controls.*



Attribute	Meaning
Checked	Whether the MenuItem displays with a checkmark next to it
Enabled	Whether the MenuItem can be selected
Visible	Whether the MenuItem is visible
ShiftToRight	(Displays as Shift Over\Down in Menu painter) Whether the MenuItem shifts to the right or down when you add MenuItems in a menu that is inherited from this menu. Selecting this attribute allows you to insert MenuItems in descendent menus, instead of only being able to add MenuItems to the end of menus in descendent menus. For more information, see "Inserting MenuItems within a menu" on page 258.

The settings you specify here determine how the MenuItems display by default. You can change the values of the attributes in scripts during execution.

The In Place listbox
The In Place listbox in the Style group is used when you are building a window that uses OLE 2.0. It specifies how menus are modified when an OLE 2.0 object is activated.

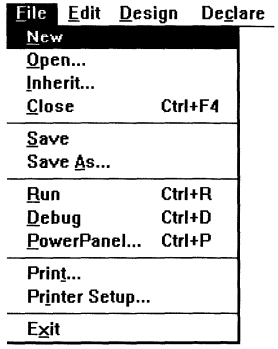
For more information, see the chapter on OLE in *Building Applications*.

Assigning accelerator and shortcut keys

All MenuItems should have **an accelerator key**, which allows a user to select the item from the keyboard by pressing ALT+*key* when the menu is displayed. Accelerator keys display with an underline in the MenuItem's text.

In addition, you can define **shortcut keys**, which are function keys or combinations of keys that a user can press to select a MenuItem anytime.

For example, in the following menu all MenuItems have accelerator keys (New's accelerator key is N, Open's is O, and so on). Close, Run, Debug, and PowerPanel each have shortcut keys (the CTRL key in combination with another key).



You should adopt conventions for using accelerator and shortcut keys in your applications. All items should have accelerator keys, and commonly used items should have shortcut keys.

Assigning accelerator keys

Users can press a MenuItem's accelerator key to select the item whenever the menu containing the item is displayed.

❖ To assign an accelerator key:

- Type an ampersand (&) before the letter in the MenuItem 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.

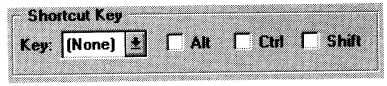
Accelerator keys display as underlined letters in the menu during execution.

Assigning shortcut keys

Users can press a MenuItem's shortcut key to select the item anytime without having to display the menu.

❖ To assign a shortcut key:

1. Select the MenuItem you want to assign a shortcut key to.
2. Select a key from the Key dropdown listbox in the Shortcut Key group.



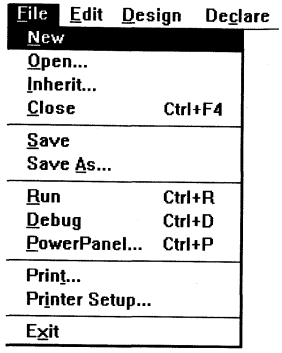
3. If you want, select Alt, Ctrl, and/or Shift to create a key combination.

   PowerBuilder displays the shortcut key in the box next to the MenuItem text.

Shortcut keys display to the right of the MenuItem text during execution.

Creating separation lines in menus

You should separate groups of related MenuItems with lines.



❖ To create a line between items on a menu:

- Type a single dash (-) as the MenuItem text.

Defining MicroHelp text and toolbar items

Two properties of MenuItems are used in Multiple Document Interface (MDI) applications:

- MicroHelp text
- Association of a MenuItem with an button in a toolbar

For information about defining these properties, see the chapter on building MDI applications in *Building Applications*.

Saving menus

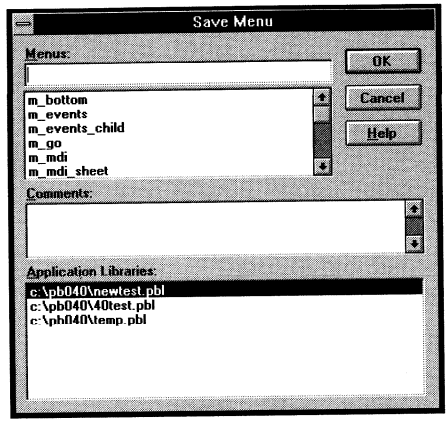
You can save the menu you are working on anytime. When you save a menu, PowerBuilder saves the compiled MenuItems and scripts in the library you specify.

❖ 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 below).

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.

5 Click OK.

Naming the menu

The menu name can be any valid PowerBuilder identifier up to 40 characters.

For information about PowerBuilder identifiers, see *PowerScript Language*.

A recommendation

When you name menus, you should use a two-part name: a standard prefix that identifies the object as a menu (such as m_) and a suffix that helps you identify the particular menu.

For example, you might name a menu used in a sales application m_sales.

Viewing your work

While building a menu, you can preview it and print out its definition.

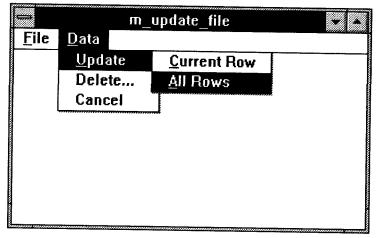
Previewing a menu

You can visually preview a menu anytime to see how it looks.

❖ To preview a menu:

- Select Design➤Preview from the menu bar.
  or
  Press CTRL+W.

  The menu displays in a window.



What you can do

You can navigate through the menu using the mouse or the keyboard.

What you cannot do

You cannot trigger events in the MenuItems. For example, clicking a MenuItem while previewing does not trigger the item's Clicked event.

❖ To return to the Menu painter:

- Select Design➤Preview from the menu bar.
  or
  Select Close from the preview window's Control menu.

  You return to the Menu painter.

Printing a menu's definition

You can print a menu's definition for documentation purposes.

❖ To print information about the current menu:

- Select File➤Print from the menu bar.

  Information about the current menu is sent to the printer specified in Printer Setup. The information that is sent to the printer depends on variables specified in the Library section of the PB.INI file.

> **Print settings**
> You can view and change the print settings in the Library painter (select Entry➤Print from the menu bar) or in the Preferences painter.

Writing scripts for MenuItems

You write scripts that specify what happens when users select a MenuItem.

❖ **To write a script for a MenuItem:**

1 Select the MenuItem.

2 Click the Script button in the PainterBar.

The PowerScript painter opens.

MenuItem events

MenuItems have the following events:

- Clicked
- Selected

About the Clicked event

The Clicked event is triggered whenever:

- The MenuItem is clicked with the mouse
- The MenuItem is selected (highlighted) using the keyboard and then ENTER is pressed
- The shortcut key for the MenuItem is pressed
- The menu containing the MenuItem is displayed and the accelerator key is pressed

A MenuItem responds to a mouse click or the keyboard only if both its Visible and Enabled attributes are TRUE.

If the MenuItem has a dropdown or cascading menu under it, the script for its Clicked event (if any) is executed when the mouse button is pressed, and then the dropdown or cascading menu displays. If the MenuItem does not have a menu under it, the script for the Clicked event is executed when the mouse button is released.

Tip

When the user clicks the menu bar to display a dropdown menu, the Clicked event for the MenuItem on the menu bar is triggered, then the dropdown menu is displayed.

So you can use the menu bar's Clicked event to specify the attributes of the MenuItems in the dropdown menu. For example, if you want to disable items in a dropdown menu, you can disable them in the script for the Clicked event for the MenuItem in the menu bar.

About the Selected event

The Selected event is triggered when the user selects (highlights) a MenuItem.

Using the events

Typically, your application will contain Clicked scripts for each MenuItem in a dropdown or cascading menu. For example, the script for the Clicked event for the Open MenuItem on the File menu will open a file.

You will probably use few Selected scripts since users don't expect things to happen when they simply highlight a MenuItem. (One possible use of Selected scripts, though, is to change MicroHelp displayed in an MDI application as the user scrolls through a menu.)

Using functions and variables

You can use functions and variables in your scripts.

Using functions

PowerBuilder provides built-in functions that act on MenuItems. You can use these functions in scripts to manipulate MenuItems during execution. For example, to hide a menu, you can use the Hide built-in function.

For a complete list of the menu-level built-in functions, see *Objects and Controls*.

Defining menu-level functions
You can define your own menu-level functions to make it easier to manipulate your menus.

For more information, see Chapter 4, "Working with User-Defined Functions."

Using variables

Scripts for MenuItems have access to all global variables defined for the application, and you can define local variables, which are accessible only in the script where they are defined.

You can also define instance variables for the menu when you have data that needs to be accessible to scripts in several MenuItems in a menu. Instance variables are accessible to all MenuItems in the menu.

For a complete description of variables and how to declare them, see *PowerScript Language*.

Defining menu-level structures
If you need to manipulate a collection of related variables, you can define menu-level structures.

For more information, see Chapter 5, "Working with Structures."

Referring to objects in your application

You can refer to any object in the application in scripts for MenuItems. 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:

window

When referring to an attribute in a window, you must always qualify the attribute with the window's name:

window.attribute

Examples

To move the window w_cust from within a MenuItem script, code:

```
w_cust.Move(300, 300)
```

To minimize w_cust, code:

```
w_cust.WindowState = Minimized!
```

Using ParentWindow

You can use the reserved word ParentWindow to refer to the window that the menu is associated with during execution. For example, the following statement closes the window the menu is associated with:

```
Close(ParentWindow)
```

You can also use ParentWindow to refer to attributes of the window a menu is associated with, but not to refer to attributes *of controls or user objects* in the window.

For example, the following statement is valid, because it refers to attributes 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.attribute
window.userobject.attribute

Example

To enable a CommandButton in window w_cust from a MenuItem script, code:

```
w_cust.cb_print.Enabled = TRUE
```

Referring to MenuItems

When referring to a MenuItem, use this syntax:

menu.menuitem
menu.menuitem.attribute

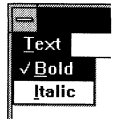
Reference within the same menu
When referring to a MenuItem within the same menu, you don't have to qualify the reference with the menu name.

When referring to a MenuItem in a dropdown or cascading menu, you must specify each MenuItem on the path to the MenuItem you are referencing, separating the names with periods.

Examples

To place a checkmark next to the MenuItem m_bold, which is on a dropdown menu under m_text in the menu saved in the library as m_menu, code:

```
m_menu.m_text.m_bold.Check( )
```



If the above script were for a MenuItem in the same menu (m_menu), you wouldn't need to qualify the MenuItem with the name of the menu. You could simply code:

```
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.

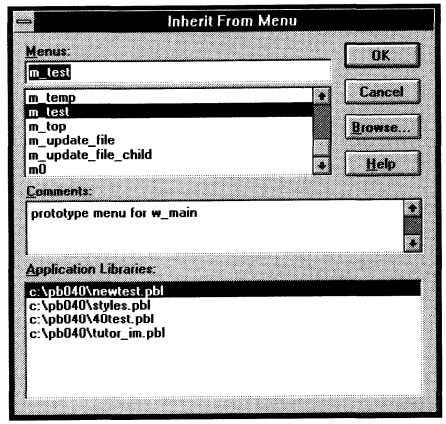
❖ To use inheritance to build a descendent menu:

1 Open the Menu painter.

The Select Menu dialog box displays.

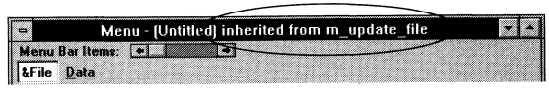
2 Click the Inherit button.

The Inherit From Menu window lists the menus in the current library.



3 Select the menu you want to use to create the descendant.

The selected menu displays in the workspace. The title of the workspace indicates that the menu is a descendant. For example, if the new menu inherits from the existing menu m_update_file, the title bar displays the following:



4 Make the changes you want to the descendent menu as described in the next section.

5 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 MenuItems 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 descendant menu:

- Add MenuItems to the end of a menu
- Insert MenuItems in a menu

  For more information, see "Inserting MenuItems within a menu" on page 258.
- Modify existing MenuItems

  For example, you can change the text displayed for a MenuItem or change its initial appearance, such as making it disabled.
- Build scripts for MenuItems that don't 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 descendant menu:

- Change the order of inherited MenuItems
- Delete an inherited MenuItem

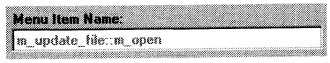
Hiding a MenuItem
If you don't need a MenuItem in a descendant menu, you can hide it using the Hide built-in function.

About MenuItem names

PowerBuilder uses the following syntax to show names of inherited MenuItems:

ancestormenu::menuitem

For example, in a menu derived from m_file_update, you see the following for the m_open MenuItem, which is defined in m_file_update:



Understanding inheritance

The issues concerning inheritance with menus are the same as the issues concerning inheritance with windows and user objects. Chapter 8, "Understanding Inheritance," describes the issues in detail:

- The basics of inheritance
- How to view the inheritance hierarchy
- Considerations when using inherited objects
- How to use inherited scripts
- How to call an ancestor script
- How to call an ancestor function

Inserting MenuItems within a menu

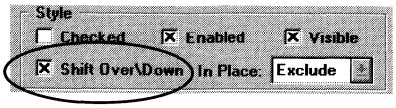
When defining a descendent menu, you might want to insert MenuItems in the middle of a dropdown or cascading menu or in the middle of the menu bar. Here's how to do that.

❖ To insert MenuItems in a menu in a descendent menu:

1 In the parent menu, select Shift Over\Down for all of the following MenuItems:

- MenuItems in the menu bar that need to be shifted right when you insert an item in the menu bar in a descendent menu

- MenuItems in a dropdown or cascading menu that need to be shifted down when you insert an item in a dropdown or cascading menu



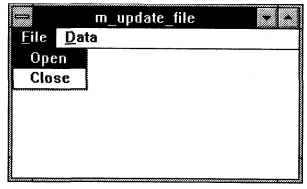
2 In the descendent menu, add MenuItems to the end of the menu bar or at the end of the appropriate dropdown or cascading menus in the Menu painter workspace.

At execution time, PowerBuilder will insert the added MenuItems before all the MenuItems specified as shifting right in the parent.

About the display in the Menu painter
When you insert MenuItems in a descendent menu in the Menu painter, the workspace does not show the insertions; it shows all added MenuItems at the end of the menu. However, when you preview the menu, as described on page 248, the MenuItems are shown in the order in which they will display in your application.

An example of inserting MenuItems

Say you have the following menu, m_update_file.



You want to define a menu that is inherited from this menu, with the following modifications:

- A new MenuItem, File➤New, which displays above File➤Open and File➤Close.
- A new item on the menu bar, Edit, which displays between File and Data.

In other words, you want File➤Open and File➤Close to shift down in the descendant's dropdown menu, and you want the Data menu item to shift right in the descendant's menu bar.

Here is how you would do it:

1 Open m_update_file in the Menu painter.

2 Select Shift Over\Down for each of the following MenuItems:

- File➤Open and File➤Close (you want these items to shift down when you insert File➤New in the descendant)
- Data in the menu bar (you want this item to shift right when you insert Edit in the menu bar in the descendant)

3 Save and close m_update_file

4 Create a new menu inherited from m_update_file.

5 Add a New MenuItem to the File dropdown menu.

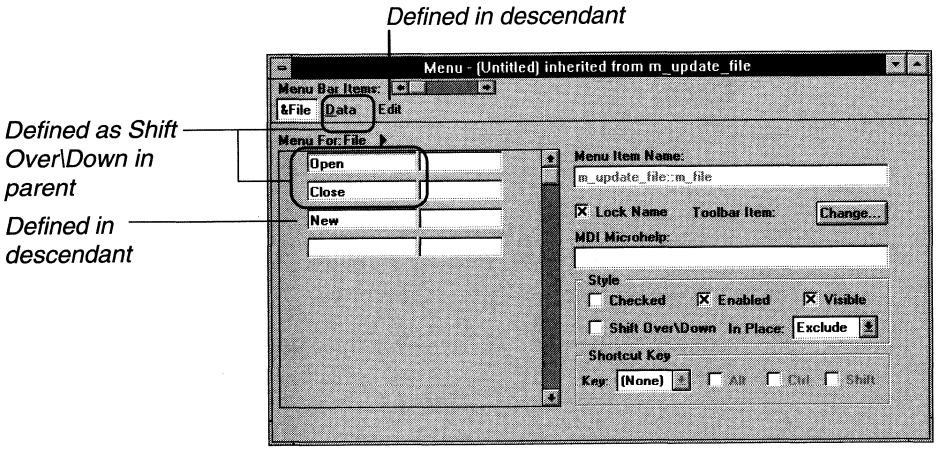
The New item displays at the end of the dropdown menu in the workspace.

6 Add an Edit MenuItem to the menu bar.

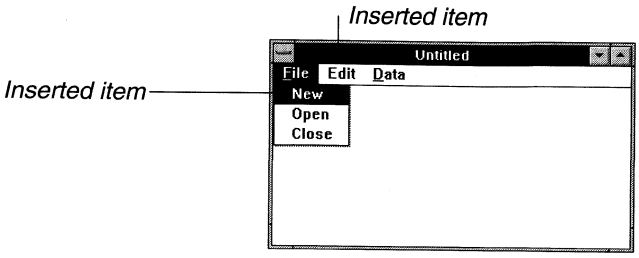
The Edit item displays at the end of the menu bar in the workspace.

Because you defined File➤Open, File➤Close, and Data as Shift Over\Down, when you execute the descendent menu, the new MenuItems are inserted as you want. You can see this when you preview the menu.

Here is the workspace:



During preview and at execution time, the MenuItems defined in the descendant are inserted appropriately:



Using menus

You can use menus in two ways:

- Place them in the menu bar of a window
- Display a menu as a popup menu

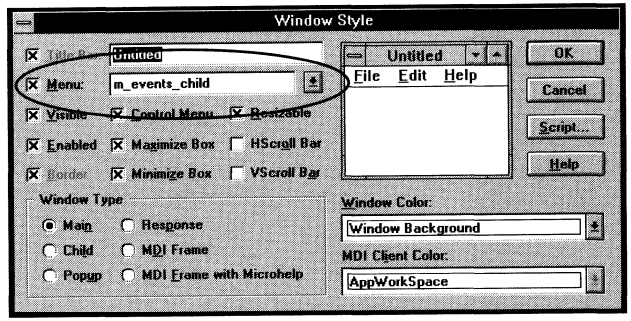
Adding a menu bar to a window

To display a menu bar when the window is opened, you associate a menu with the window in the Window painter.

❖ To associate a menu with a window:

1. Open the Window painter and select the window you want to associate the menu with.
2. Double-click the window's background.
   or
   Select Design➤Window Style from the menu bar.

   The Window Style dialog box displays.
3. Select the Menu checkbox and pick the menu from the dropdown listbox, which lists all menus available to the application.



Identifying MenuItems in window scripts

You can reference MenuItems in scripts in windows and controls using the following syntax:

menu.menuitem

You must always fully qualify the MenuItem with the name of the menu.

When referring to a MenuItem in a dropdown or cascading menu, you must specify each MenuItem on the path to the MenuItem you are referencing, separating the names with periods.

For example, to refer to the Enabled attribute of MenuItem m_open, which is under the menu bar item m_file in the menu saved in the library as m_menu, code:

```
m_menu.m_file.m_open.Enabled
```

Changing a window's menu during execution

You can use the ChangeMenu function in a script to change the menu associated with a window during execution.

For information about ChangeMenu, see the *Function Reference*.

Displaying popup menus

To display a popup 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 popup menu at the current pointer position (assuming the 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 popup menu.

The following statements create an instance of the menu m_new, then pop up the menu m_new.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() )
```

For more information

For complete information about PopMenu, see the *Function Reference*.

CHAPTER 10

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 without any additional work. One of the best ways for you to get reusability in PowerBuilder is with user objects. This chapter describes how to define and use user objects.

Contents

Overview of user objects

Applications often have features in common. For example, you might often use a Close button that performs a certain set of operations, then closes the window. Or a listbox that lists all departments. Or you might want all your DataWindow controls to perform the same type of error checking. Or you might want a predefined file viewer that you can simply plug into windows whenever you need them. Or you might want to provide a certain package of processing—such as calculating commissions or performing statistical analysis—in several applications.

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. You don't need to redefine the object each time you need it.

> **Examples of user objects**
> The PowerBuilder Examples contain many interesting user objects in PBEXAMUO.PBL. Take a look at them to get an appreciation for the power of user objects.

There are two types of user objects:

- Visual
- Class

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 the controls in the user object and write scripts for the controls. You then place the user object in windows you build in your applications. You can use the user object as often as you need.

There are four types of visual user objects:

- Standard
- Custom
- External
- VBX

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. If you frequently use a PowerBuilder control to perform the same processing, you may want to create a standard visual user object to perform the processing.

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.

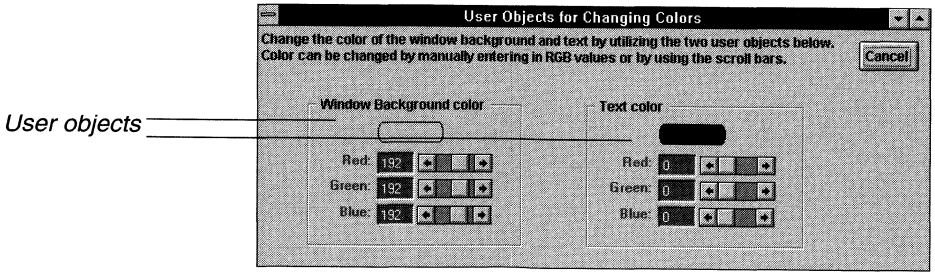
Custom visual user objects

Custom visual user objects are objects that have several controls that function as a unit. If you frequently group controls together in a window and always use the controls to perform the same processing, you may want to build a custom visual user object that contains these controls and their scripts.

> **Tip**
> Think of a custom visual user object as a window that is a single unit and is used as a control.

You might want to build a user object that contains several buttons that perform standard processing. Each button has a script that specifies its processing. When you place the user object in a window, you place all the buttons as a unit in the window.

The following window (from the PowerBuilder Examples) contains two custom visual user objects, which users can manipulate to set colors:

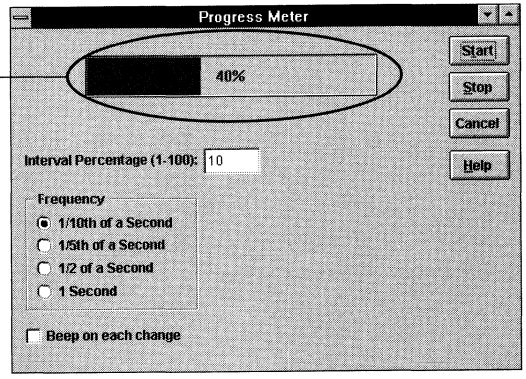


External visual user objects

External visual user objects contain controls from objects in the underlying windowing system that were created outside PowerBuilder. If you have a custom DLL, you can use it in PowerBuilder to create an external user object. To use the DLL, 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.

For example, the following window (from the PowerBuilder Examples) shows a progress meter that is defined in an external DLL:

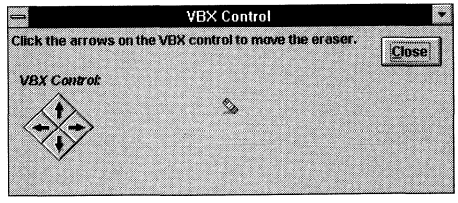
External visual user object



VBX user objects

VBX user objects are objects that are compatible with Visual Basic Version 1.0. You can build or purchase the files that define the VBX controls and use them in your PowerBuilder applications.

The following window (from the PowerBuilder Examples) uses a VBX user object derived from a public domain VBX control:



Class user objects

You might need to implement processing that has no visual component, but which you want to use in many places—such as calculating commissions or performing statistical analysis. In object-oriented terminology, this type of processing is implemented through a **class**. You typically use classes to define business rules and other processing that acts as a unit. You implement them in PowerBuilder using class user objects.

You build class user objects in the User Object painter, specifying instance variables (attributes, in object-oriented terminology) and object-level functions (methods). Then you create an instance of the class user object in your application, thereby making its processing available.

For example, you might want to define a class user object that performs statistical analysis. Whenever you need to do this type of processing, you would instantiate the user object in a script and call its functions.

There are two kinds of class user objects:

- Standard
- Custom

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. You can add instance variables and functions to enhance the behavior of the built-in object. Once you have defined your 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 this type of user object is defining a standard class user object inherited from the built-in transaction object and using it to do database remote procedure calls from within an application.

Custom class user objects

Custom class user objects are objects of your own design that encapsulate attributes and functions but are not visible to the user. They are not derived from PowerBuilder objects. You define custom class user objects to create units of processing that have no visual component.

For example, if you need to calculate commissions in an application, you can define a Calculate_Commission custom class user object that contains attributes and user-defined functions that do the processing needed to calculate commissions.

Whenever you need to provide this processing, you create an instance of the user object in a script, which then has access to the logic defined in the user object.

Building user objects

There are two ways to build a user object. You can:

- Build a user object from scratch

  The next section describes how to build user objects from scratch.

- Build a user object that inherits its style, events, functions, structures, variables, and scripts from an existing user object

  "Using inheritance to build user objects" on page 287 describes how to build user objects descended from another 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 aren't based on existing user objects.

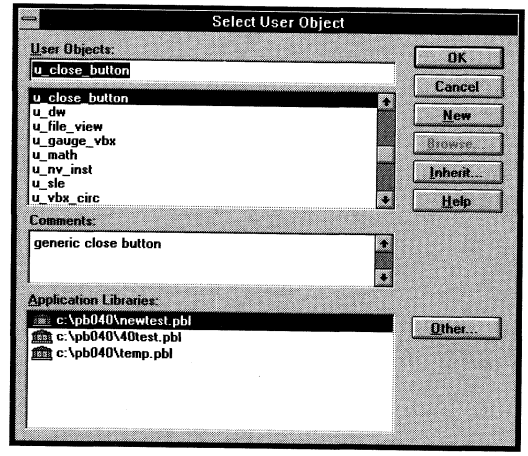
You use the User Object painter to build user objects.

Opening the User Object painter

❖ To open the User Object painter:

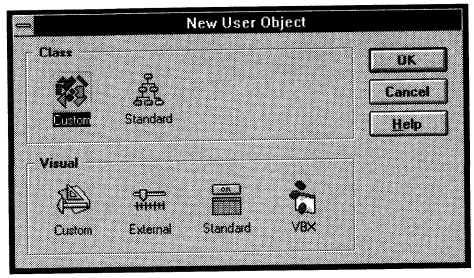
1 Click the User Object painter button in the PowerBar or PowerPanel.

The Select User Object dialog box displays, listing the user objects in the current library.



2 Click the New button to build a new user object.

The New User Object dialog box displays.



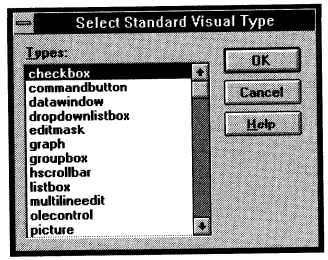
3 Select the type of user object you want to build.

What you do next depends on the user object type you selected.

For information about building	See
Standard visual user objects	Page 272
Custom visual user objects	Page 274
External visual user objects	Page 276
VBX user objects	Page 278
Standard class user objects	Page 282
Custom class user objects	Page 284

Building a standard visual user object

When you select the standard visual object type in the New User Object dialog box, PowerBuilder displays the list of control types.

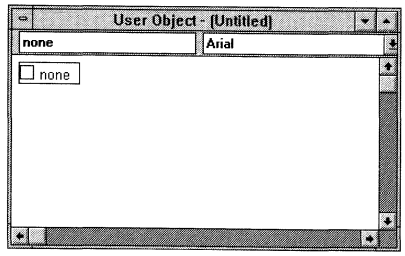


❖ To build a standard visual user object:

1 Select the PowerBuilder control you want to use to build your standard visual user object. Your object will have the attributes and events associated with the PowerBuilder control you are modifying.

2 Click OK.

The selected control displays in the User Object painter workspace. For example, if you chose CheckBox, the workspace looks like this:



3 Work with the control exactly as you do in the Window painter. You can press the right mouse button to display the control's popup menu. You can double-click the control to open the Style dialog box.

4 Review the default attributes and make any necessary changes.

5 Declare any functions, structures, or variables you need for the user object.

> **Where to declare them**
> You can declare functions, structures, and variables for the user object in the User Object painter workspace or PowerScript painter.

6 Declare any needed user events for the user object.

For information about user events, see "Communicating between a window and a user object" on page 296.

7 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.

8 Save the user object.

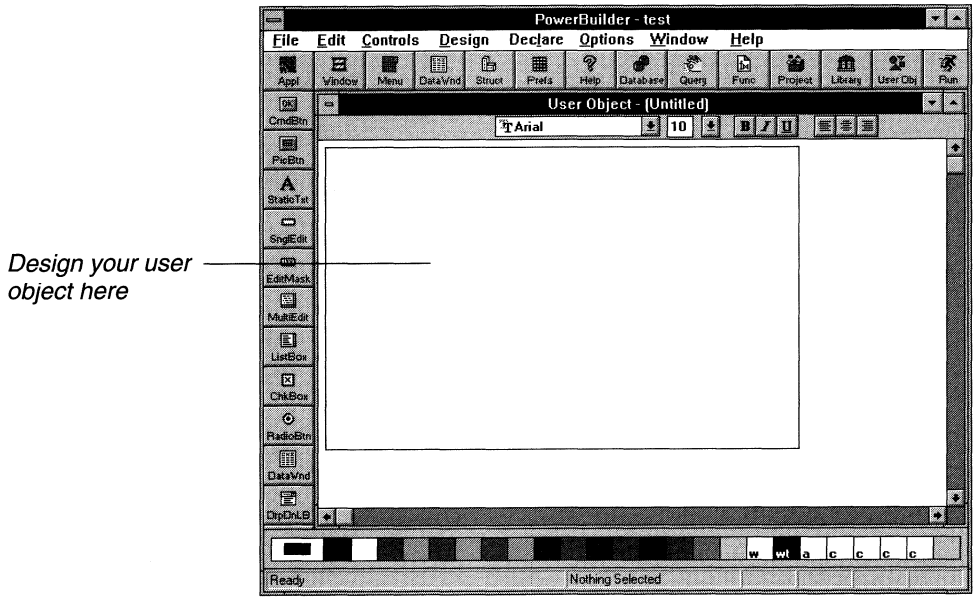
See "Saving a user object" on page 284.

9 Use the user object in your application.

For more information, see "Using user objects" on page 290.

Building a custom visual user object

When you select the custom visual user object type in the New User Object dialog box and click OK, the User Object painter workspace displays. It looks like the Window painter workspace; the empty box in the upper-left corner is the custom visual user object.



The procedure for building a custom visual user object is similar to the process of building a window, described in Chapter 6, "Defining Windows."

❖ To build a custom visual user object:

1 Place the controls in the custom visual user object.

2 Define the attributes of the controls just as you do in the Window painter.

3 Declare any functions, structures, or variables you need for the user object.

> **Where to declare them**
> You can declare functions, structures, and variables for the user object in the User Object painter workspace or PowerScript painter.

4 Declare any needed events for the user object or its controls.

For information about user events, see "Communicating between a window and a user object" on page 296.

5 Build and compile the scripts for the user object or its controls. You can write scripts for each of the controls in a custom visual user event. In addition, custom visual user objects have the following events associated with them:

Event	Occurs
Constructor	Immediately after the window's Open event and when a user object is dynamically placed in a window
Destructor	Immediately after the window's Close event and when a user object is dynamically removed from a window
DragDrop	When a dragged object is dropped on the user object
DragEnter	When a dragged object enters the user object
DragLeave	When a dragged object leaves the user object
DragWithin	When a dragged object is moved within the user object
Other	When a Windows message occurs that is not a PowerBuilder event
RButtonDown	When the right mouse button is pressed

For more about drag and drop, see *Building Applications*.

6 Save the user object.

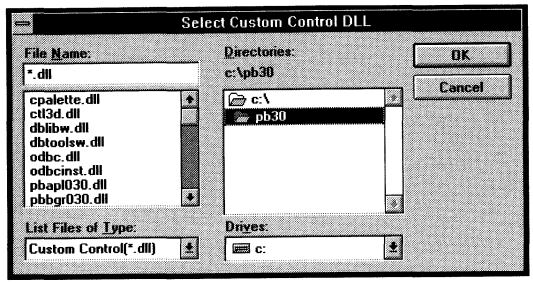
See "Saving a user object" on page 284.

7 Use the user object in your application.

For more information, see "Using user objects" on page 290.

Building an external visual user object

When you select the External visual user object type in the New User Object dialog box and click OK, the Select Custom Control DLL dialog box displays.

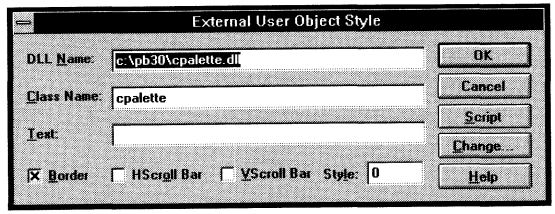


❖ To build an external visual user object:

1 Specify the DLL that defines the user object.

2 Click OK.

The External User Object Style dialog box displays.



3 Enter the name of the DLL.

4 Enter the class name registered in the DLL (information about class name is usually provided by the vendor of the purchased DLL).

5 Enter text in the Text box. Text is only displayed if the object has a text style attribute.

6 Modify the display attributes (border and scrollbars) as required.

7 Enter 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 attributes for the control.

8 Click OK. The selected object displays in the User Object painter workspace.

9 Declare any functions, structures, or variables you need to declare for the user object (information about functions is usually provided by the vendor of the purchased DLL).

> **Where to declare them**
> You can declare functions, structures, and variables for the user object in the User Object painter workspace or in the PowerScript painter.

10 Declare any needed events for the user object.

For information about user events, see "Communicating between a window and a user object" on page 296.

11 Create and compile the scripts for the user object. External visual user objects have the following events associated with them:

Event	Occurs
Constructor	Immediately after the window's Open event and when a user object is dynamically placed in a window
Destructor	Immediately after the window's Close event and when a user object is dynamically removed from a window
DragDrop	When a dragged object is dropped on the user object
DragEnter	When a dragged object enters the user object
DragLeave	When a dragged object leaves the user object
DragWithin	When a dragged object is moved within the user object
Other	When a Windows message occurs that is not a PowerBuilder event
RButtonDown	When the right mouse button is pressed

For more about drag and drop, see *Building Applications*.

12 Save the user object.

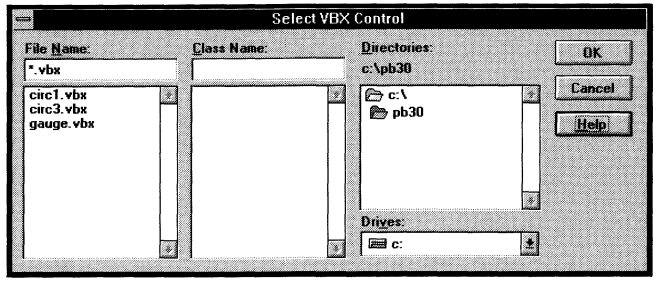
See "Saving a user object" on page 284.

13 Use the user object in your application.

For more information, see "Using user objects" on page 290.

Building a VBX user object

When you select the VBX object type in the New User Object dialog box and click OK, the Select VBX Control dialog box displays.



For VBX control developers

For information about developing VBX controls for use in PowerBuilder, see "Notes for VBX control developers" on page 281.

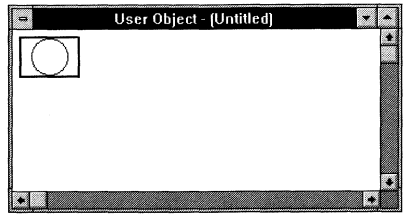
❖ To build a VBX user object:

1 Specify the VBX file that defines the VBX control.

2 Select the class name for the control in the Class Name box (for a description of the classes, see the VBX control vendor's documentation).

3 Click OK.

PowerBuilder displays the user object in the workspace.



4 Specify the attributes for the user object (see the next section).

5 Build and compile the scripts for the user object.

See "Writing scripts for VBX user objects " on page 280.

6 Save the user object.

See "Saving a user object" on page 284.

7 Use the user object in your application.

For more information, see "Using user objects" on page 290.

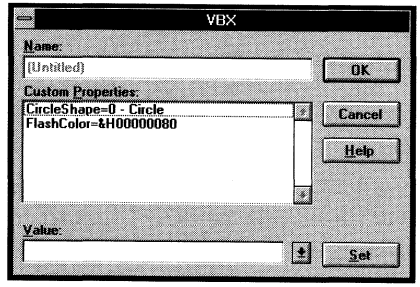
Specifying attributes

Each type of VBX control has its own set of attributes (also called properties). When you define a VBX user object, PowerBuilder reads the definition from the VBX file and builds an attribute list.

❖ To define a VBX user object's attributes:

1 Double-click the object in the Window or User Object painter workspace.

The VBX dialog box displays the attributes unique to the VBX control.

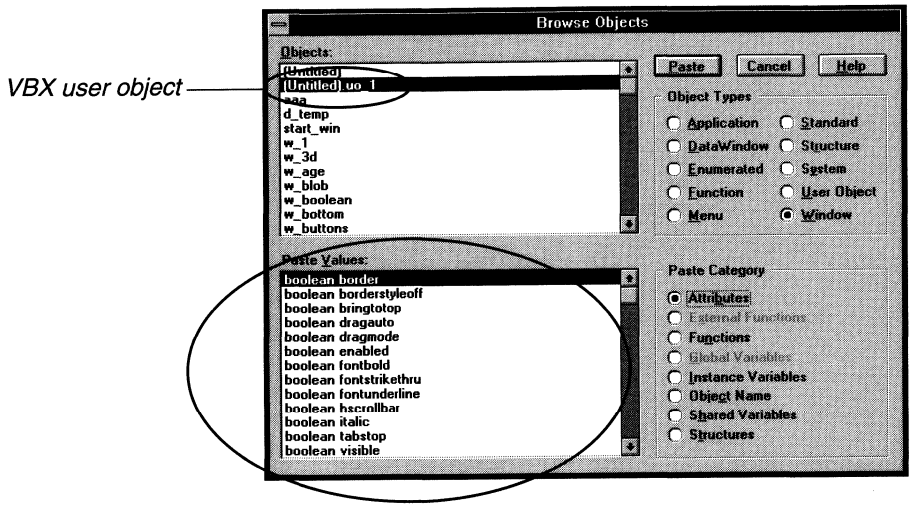


2 Change values for the attributes as needed: select an attribute, enter a value in the Value box, and click Set.

3 Click OK.

For complete information about the attributes for a specific VBX control, see the documentation from the VBX control's vendor.

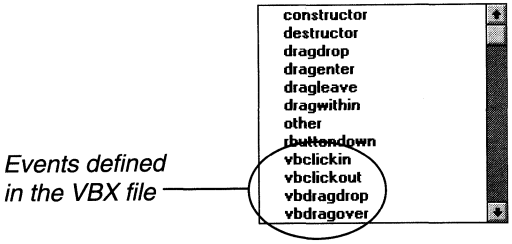
You can also assign values to the attributes in a script. You can see a list of the attributes in the Object browser.



Writing scripts for VBX user objects

VBX user objects have the same events as custom and external visual user objects, plus events specific to the VBX user object.

When you define a VBX user object in the User Object painter, PowerBuilder reads in the events defined in the VBX file. It prepends *VB* to the name of each event and lists them in the Select Event listbox in the PowerScript painter.



For complete information about the events for a specific VBX control, see the documentation from the VBX control's vendor.

You can call DLL functions exported by VBX controls in scripts (for the list of available functions and for the definition of the function, see the documentation provided by your VBX vendor). Declare each function as a PowerBuilder external function.

Passing parameters
If a VBX user object event takes parameters, you can supply the parameters through the PowerScript functions EventParmDouble and EventParmString.

For more information, see the *Function Reference.*

Notes for VBX control developers

You should be aware of the following if you are developing VBX controls for use in PowerBuilder:

- PowerBuilder supports only Version 1.0 VBX controls.
- VBRecreateControlHwnd is supported.
- VBReadBasicFile and VBWriteBasicFile are supported. The file number used is an integer value generated by PowerBuilder when a file is in a script.
- Two functions that control refreshing PowerBuilder attributes are available. Refresh is either on or off for all variables. The default value is on (TRUE):

  ```
  LONG FAR PASCAL VBX_SetControlRefreshFlag(HCTL hCtl,
  BOOL bRefresh)

  LONG FAR PASCAL VBX_SetControlhWndRefreshFlag(HWND
  hCtlWnd, BOOL bRefresh)
  ```

  Set bRefresh to TRUE to refresh, FALSE to ignore refresh.

  To use these functions, declare them as external functions. They are in PBVBX040.DLL. Here is how you would declare the second function:

  ```
  function long VBX_SetControlhWndRefreshFlag(int hWnd,
  boolean bRefresh) library "PBVBX040.DLL"
  ```

Here is a sample script using the function:

```
int hWnd
long lStat

// Get the hWnd by calling the HANDLE function
// on the control in PowerScript.
hWnd = Handle(w_mywin.uo_myvbxcontrol)

// Turn on the refresh flag.
lStat = VBX_SetControlhWndRefreshFlag(hWnd,
TRUE)
```

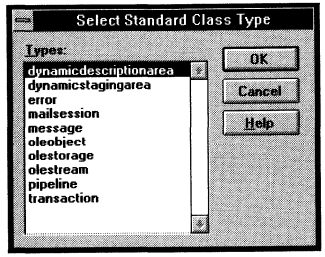
- You can write a new function (PBINITCC) as an entry point for PowerBuilder to replace VBINITCC. The function PBINITCC will be called if found; otherwise, VBINITCC will be called. Using PBINITCC allows you to recognize PowerBuilder as the host environment and react accordingly (you might want to register a different model to reduce the number of custom properties visible to PowerBuilder and therefore copied into PowerBuilder attributes).

  Here is the prototype for PBINITCC:

```
BOOL FAR PASCAL __export PBINITCC(
    USHORT    usPBVersion,        // Current 0x0400
    BOOL      fRunTime,           // See above
    PBPROC    lpfnRefreshFunction // for info about
    );                            // VBX_SetControlRefreshFlag.
```

Building a standard class user object

When you select the standard class user object type in the New User Object dialog box and click OK, the Select Standard Class Type dialog box displays.



❖ To build a standard class user object:

1 Select the built-in system object that you want your user object to inherit from.

2 Click OK.

The User Object painter workspace displays.

3 Declare any functions, structures, or variables you need for the user object.

> **Where to declare them**
> You can declare functions, structures, and variables for the user object in the User Object painter workspace or PowerScript painter.
>
> **For a list of attributes and functions**
> In the PowerScript painter, you can use the Quick browser to see a list of all the built-in attributes and functions for the system object you are inheriting from.

4 Declare any needed events for the user object.

For information about user events, see "Communicating between a window and a user object" on page 296.

5 Build and compile the scripts for the user object. Standard class user objects have the following events associated with them:

Event	Occurs
Constructor	When the user object is created
Destructor	When the user object is destroyed

6 Save the user object.

See "Saving a user object" on page 284.

7 Use the user object in your application.

For more information, see "Using user objects" on page 290.

Building a custom class user object

When you select the custom class visual user object type in the New User Object dialog box and click OK, the User Object painter workspace displays.

❖ To build a custom class user object:

1 Declare any functions, structures, or variables you need for the user object.

> **Where to declare them**
> You can declare functions, structures, and variables for the user object in the User Object painter workspace or in the PowerScript painter.

2 Build and compile the scripts for the user object. Custom class user objects have the following events associated with them:

Event	Occurs
Constructor	When the user object is created
Destructor	When the user object is destroyed

3 Save the user object.

See "Saving a user object" on page 284.

4 Use the user object in your application.

For more information, see "Using user objects" on page 290.

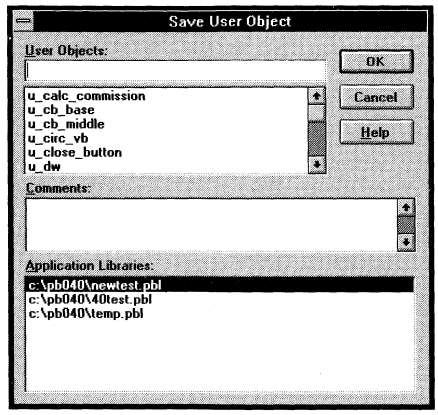
Saving a user object

❖ To save a user object:

1 In the User Object painter, select File➤Save from the menu 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 Name the user object in the User Objects box (see below).

3 Write comments to describe the user object. These comments display in the Select User Object 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 user object later.

4 Specify the library in which to save the user object.

> **Which library?**
> In your applications, you can use any user object that is in the current application library search path. 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.

Naming the user object

The user object name can be any valid PowerBuilder identifier up to 40 characters.

For information about PowerBuilder identifiers, see *PowerScript Language*.

A recommendation

You should adopt naming conventions when you create user objects to make it easy to understand a user object's type and purpose. Here is a convention you could follow:

Use u_ as the prefix for all user objects and apply the prefix for the various types of user objects as follows:

Type of user object	Format	Example
Standard visual	u_*control_purpose*	u_cb_close is a standard visual user object that consists of a CommandButton that closes a window
custom visual	u_*purpose*	u_toolbar is a custom user object that is a toolbar
external visual	u_ex_*purpose*	u_ex_sound is an external user object that outputs sound
VBX	u_vbx_*purpose*	u_vbx_gauge is a VBX user object that is a gauge
standard class	u_*systemobject_purpose*	u_trans_test is a standard class user object derived from the transaction object used for testing
custom class	u_cust_*purpose*	u_cust_commission is a custom class user object that calculates commissions

Using inheritance to build user objects

When you build a user object that inherits its definition (attributes, events, functions, structures, variables, controls, and scripts) from an existing user object, you save coding time. All you have to do is modify the inherited definition to meet the requirements of the current application.

For example, assume your application has a user object u_file_view that has three CommandButtons:

- List—displays a list of files in a ListBox
- Open—opens the file the user selects and displays the selected file in a MultiLineEdit
- Close—displays a message box and then closes the window

Then assume you want to build another user object that is exactly like the existing u_file_view except it has a fourth CommandButton. If you use inheritance to build the new user object, all you have to do is add the fourth CommandButton.

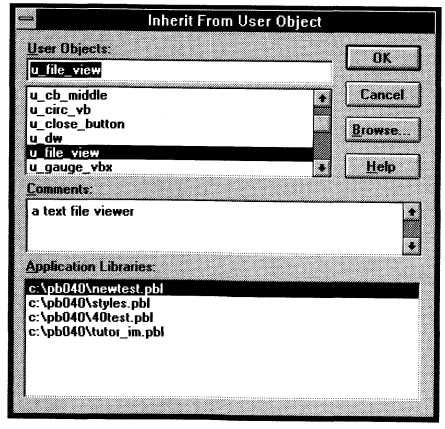
❖ To use inheritance to build a descendant user object:

1 Open the User Object painter.

The Select User Object dialog box displays.

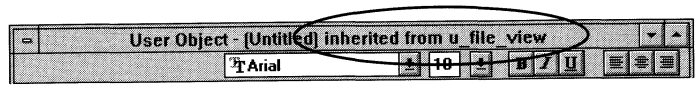
2 Click the Inherit button.

The Inherit From User Object dialog box lists the user objects in the current library.



3 Select the user object you want to use to create the descendant. The selected object displays in the workspace.

The title bar for the workspace indicates that the object is a descendant. For example, if the new user object inherits from the existing object u_file_view, the title bar looks like this:



4 Make any changes you want to the user object.

5 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 attributes, 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.

What you can do

You can do the following in a descendent user object:

- Change the values of the attributes and the variables
- Build scripts for events that do not have scripts in the ancestor
- Extend or override the inherited scripts
- Add controls (if it is a custom visual user object)
- Reference the ancestor's functions, events, and structures
- Declare variables, events, functions, and structures for the descendant

What you cannot do

You cannot delete controls from an inherited custom visual user object.

> **Hiding controls**
> If you don't need a control in a descendant 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. Chapter 8, "Understanding Inheritance," describes the issues in detail:

- The basics of inheritance
- How to view the inheritance hierarchy
- Considerations when using inherited objects
- How to use inherited scripts
- How to call an ancestor script
- How to call an ancestor function

Using user objects

Once you have built your user object, you are ready to use it in your applications. This section describes

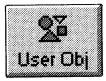
- How to use visual user objects
- How to use class user objects

Using visual user objects

You use visual user objects by placing them in a window or in a custom visual user object.

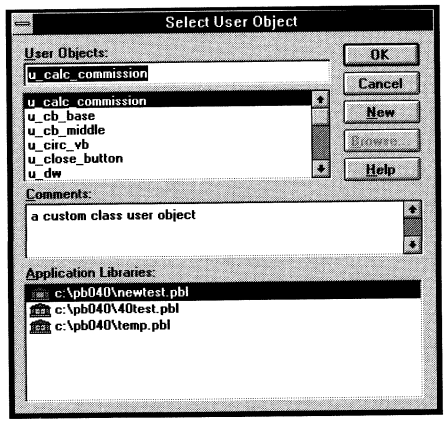
❖ To place a user object:

1 Open the Window painter to place a user object in a window, or open the User Object painter to place a user object in a custom visual user object.



2 Click the User Object button in the PainterBar.
or
Select Controls➤User Object from the menu bar.

The Select User Object dialog box displays.



3 Select the user object you want to use.

4 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.

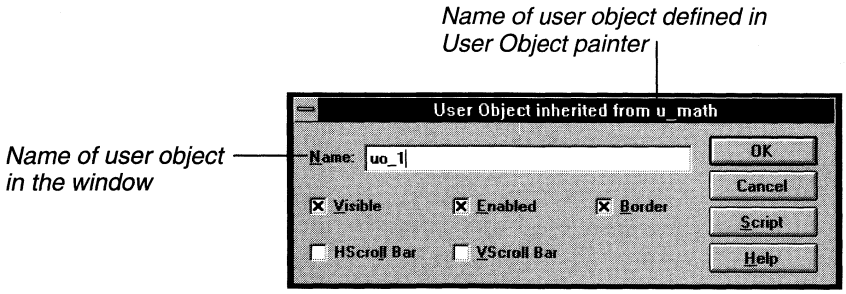
Creating user object buttons
You can place buttons for the user objects you use frequently in the PainterBar in the Window painter and User Object painter. Then you can simply click the button to place a user object.

For more information, see "About the painter" in Chapter 6, "Defining Windows."

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 anything else you can do with a control.

When you place a 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 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" in Chapter 7, "Working with Controls."

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. That means that all scripts defined for the ancestor user object are inherited. You can choose to override or extend those scripts if you want.

For more information, see Chapter 8, "Understanding Inheritance."

Writing scripts for controls in custom user objects

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 during execution

You can add a user object to a window during execution using the PowerScript functions OpenUserObject and OpenUserObjectWithParm in a script. You can remove a user object from a window using the CloseUserObject function.

For information about these functions, see the *Function Reference.*

Using class user objects

Class user objects have no visible component, so they are not placed in windows like visual user objects are. To use a class user object, you create an instance of it in a script as follows.

> **Using standard class user objects**
> If you have built a standard class user object inherited from one of the built-in globals, instead of manually creating an instance of it, you can install it as the default for your application.
>
> For more information, see "Using global standard class user objects" on page 293.

❖ **To use a class user object:**

1 Declare a variable of the user object type and create an instance of it using the CREATE statement.

The attributes and functions defined in the user object are now available within the scope of the variable declaration. You reference them using dot notation, just as you access attributes and functions of other PowerBuilder objects such as windows.

2 Use the user object's properties to do the processing you want.

3 When you have finished using the user object, be sure to destroy it using the DESTROY statement.

This frees up memory for your application.

For example, say you have defined a custom class user object named u_calc_commission that calculates commissions. Whenever you need to calculate a commission in a script, you can use the user object as follows:

```
// First, declare and create the user object.
u_calc_commission   MyClassUO
MyClassUO = CREATE u_calc_commission

// Now have access to the attributes and functions
// encapsulated in the user object.
// Do the processing here.

...
...

// When through, destroy the user object.
DESTROY MyClassUO
```

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

You can use a user object inherited from one of these globals by declaring an instance of it in a script, as described in the preceding section. If you do this, your user object is used *in addition to* the built-in global variable. Typically you will 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.

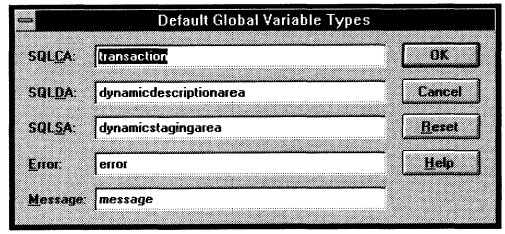
But you might want your standard class user object to *replace* the built-in global. You don't want it used in addition to the built-in global. You do this by telling PowerBuilder to use your user object *instead of* the built-in system object that it inherits from. You will probably use this technique if you have built a user object inheriting from the Error or Message object.

❖ To use a standard class user object as the default global:

1 Open the Application painter.

2 Select Edit➤Default Global Variables from the menu bar.

The Default Global Variable Types dialog box displays.



3 Specify the standard class user object you defined in the corresponding field.

> **Changing your mind**
> Click Reset to reset all listed globals to the defaults that were shipped with PowerBuilder.

4 Click OK.

Once you have installed your user object as the default global, 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 attributes and functions defined in the user object are available anywhere in the application. You reference them using dot notation, just as you access attributes and functions of other PowerBuilder objects such as windows.

For more information

For information about using the Error object, see Chapter 21, "Debugging and Running Applications," in this manual.

For information about creating your own transaction object to support database remote procedure calls, see *Building Applications*.

For information about using the Message object, see *Building Applications*.

For information about DynamicDescriptionArea and DynamicStagingArea, which are used in dynamic SQL, see *PowerScript Language*.

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 should change to that color.

  You need to pass the color from the user object to the window control.

The two best ways of communicating between a visual user object and a window are through:

- Using functions
- Using user events

Directly referencing user objects

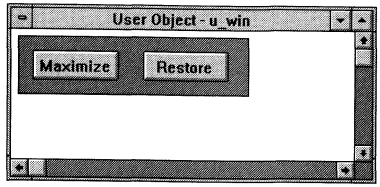
Instead of using functions and user events, you can directly reference attributes of a user object. For example, if you have a user object control called uo_1 that is associated with a custom user object that has a SingleLineEdit named sle_1, you can code the following in a script for the window:

```
uo_1.sle_1.Text = "new text"
```

But 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.

An example to illustrate the technique

This section describes how to use functions and user events to communicate between a user object and a window. It illustrates the techniques using a simple custom visual user object named u_win, which contains two buttons, Maximize and Restore.



If the user clicks the Maximize button, the current window should become maximized. If the user clicks the Restore button, the window should resume its normal size.

The problem is that the user object isn't associated with a window when it is defined in the User Object painter. So how can a script for a user-object button reference the window the user object is in? You need to pass the name of the window to the user object so the buttons can reference the window.

We will show two approaches, one that uses functions and one that uses user events.

Using functions

Exchanging information using functions is straightforward.

❖ To pass information from a window to a user object:

1 Define a public user object–level function that takes as parameters the information needed from the window.

2 Place the user object in the window.

3 When appropriate, call the user object–level function from a script in the window, passing the needed information as parameters.

Now the user object has information about the window.

❖ 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.

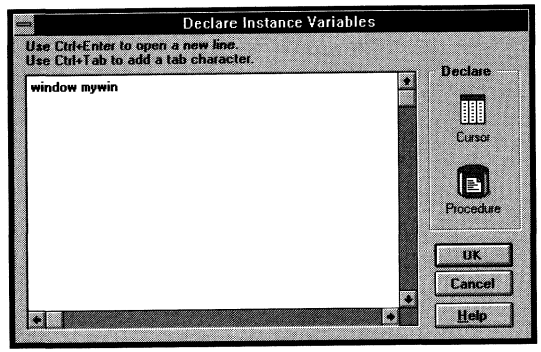
Now the window has information about the user object.

Using functions in our example

We will pass the name of the window to the user object using a function. The user object is named u_win.

In the User Object painter

1 Define an instance variable for u_win, as follows:

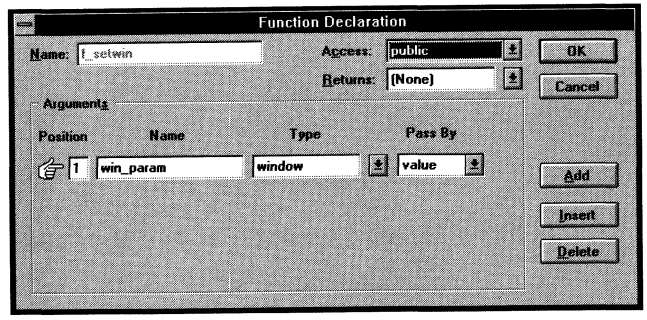


Mywin is a variable of type window.

For information about declaring variables of type window, see Chapter 6, "Defining Windows."

This variable will hold the name of the window the user object is in.

2 Declare a user object–level function, f_setwin, with the following properties:



The function takes one parameter, named win_param. Win_param is of type window (that is, win_param is a window). The function is public, so it can be called anywhere.

3 Define the function body as follows:

```
mywin = win_param
```

When f_setwin is called, the value passed in (win_param) is assigned to mywin, the user object's instance variable.

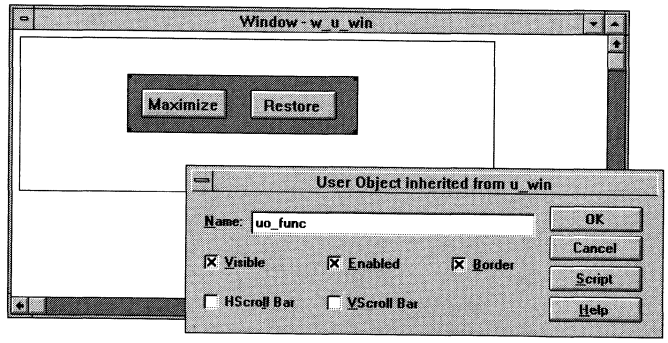
4 Write scripts for the two buttons:

Button	Script
cb_max	mywin.WindowState = Maximized!
cb_restore	mywin.WindowState = Normal!

These scripts reference the instance variable mywin, which holds the name of the window.

In the Window painter

1 Place u_win in the window and name it uo_func.



2 In the window's Open event, call the user object–level function, passing the name of the window:

```
uo_func.f_setwin(This)
```

The above function uses the reserved word This. *This* always evaluates to the name of the current entity. Since this statement is in a script for the window, it evaluates to the window's name. So the above statement passes the name of the current window.

Running the window

When the window opens, the user object–level function is called, passing the name of the window to the user object. The user object stores the name in its instance variable mywin. When a button is clicked in the user object, it knows which window to reference.

Advantages and disadvantages of using functions

On the plus side: using functions to pass information is easy. It supports the use of parameters and return types, so it is not prone to errors. It supports data encapsulation and information hiding.

On the negative side: functions create overhead, which might be unnecessary for simple operations. They operate only synchronously (your application has to wait for the function to return before continuing).

Using user events

You can also define your own events, called **user events**, to communicate between a window and a user object.

You can declare user events for any PowerBuilder object or control. In most cases, they are not necessary; PowerBuilder predefines the events you need.

However, custom visual user objects often require user events, for the following reason:

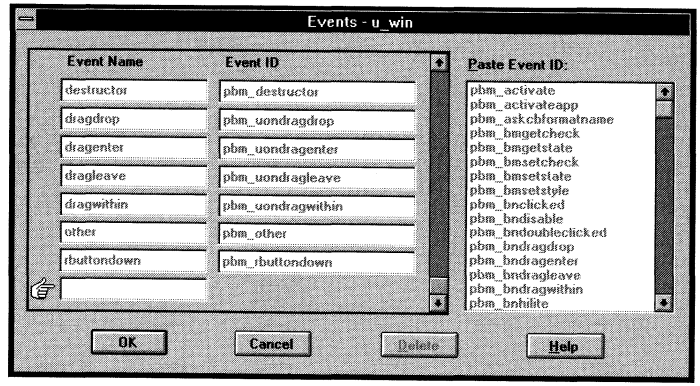
After you place a custom visual user object in a window or in another custom user object, you can write scripts only for events that occur in the user object itself. You cannot write scripts for events in the *controls* in the user object. So what you do is define your own events for the user object and trigger those events in a control in the user object. Then in the Window painter, you write scripts for the user events, referencing components of the window as needed.

❖ To define and trigger a user event in a user object:

1 Open the User Object painter and select the custom visual user object.

2 Select Declare➤User Events from the menu bar. When defining a user event for the user object itself, make sure that no control in the user object is selected; if a control is selected, the user event you define will be associated with that control, not with the user object itself.

The Events dialog box displays, showing all the events defined for the user object: Constructor, Destructor, DragDrop, DragEnter, DragLeave, DragWithin, Other, and RButtonDown.



3 Type the name of the user event in the empty Event Name box.

4 Associate the event with one of the *custom* event IDs (pbm_custom*nn*, such as pbm_custom01) by picking the event ID from the Paste Event ID box.

5 Repeat steps 3 and 4 to define additional user events.

6 Click OK.

You return to the User Object painter workspace.

7 In scripts for a control in the custom visual user object, trigger the user event using the TriggerEvent function:

userobject.**TriggerEvent** ("*eventname*")

For example, the following statement in the Clicked event for a CommandButton in a custom visual user object triggers the max_requested event in the user object:

```
Parent.TriggerEvent("Max_requested")
```

This statement uses the reserved word Parent. In the context of a control in a custom visual user object, Parent refers to the user object itself. So this statement says trigger the Max_requested event in the user object that the control is in.

Now you have defined the user event. You write scripts for these user events in the Window painter, as described next.

❖ To implement the user event in the window:

1 Open the Window painter and select the appropriate window.

2 Place the custom visual user object in the window.

3 Select the user object and open the PowerScript painter.

4 Write scripts for the user events you defined in the User Object painter.

These scripts will be triggered when controls in the user object itself call them with TriggerEvent.

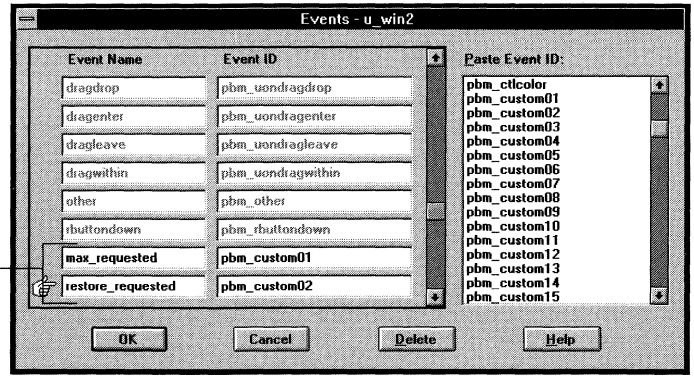
5 Save the window.

Using user events in our example

We will use user events to allow a button in a user object to affect the window it is in. The user object is u_win2.

In the User Object painter

1 Define two user events for the user object, as follows:



Two user events

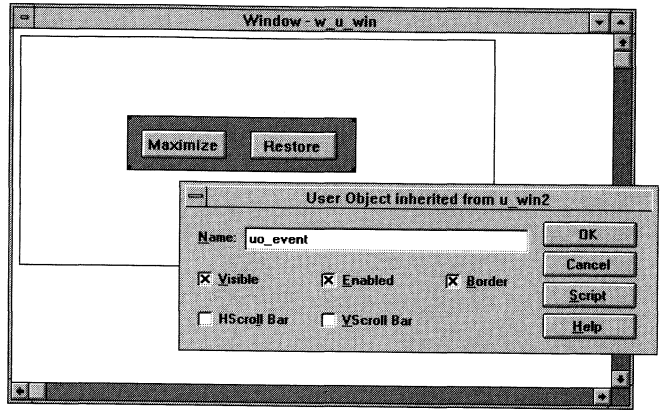
Notice that both event IDs are for the custom events (pbm_custom01 and pbm_custom02).

2 Trigger the events in scripts for the CommandButtons as follows:

Button	Script for the Clicked event
cb_max	Parent.TriggerEvent("Max_requested")
cb_restore	Parent.TriggerEvent("Restore_requested")

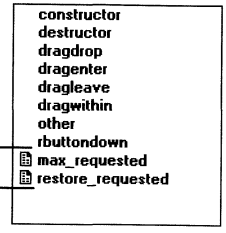
In the Window painter

1 Place the user object in the window and name it uo_event.



2 Select uo_event and open the PowerScript painter.

The two user events defined in the User Object painter display in the Select Event listbox.



Two user events

3 Write scripts for the two user events:

Event	Script
Max_requested	Parent.WindowState = Maximized!
Restore_requested	Parent.WindowState = Normal!

These scripts use the Parent reserved word. In the context of a user object in a window, Parent is the name of the window containing the user object.

Running the window

Here is what happens when the Maximize button is clicked in this scenario:

1 The Max_requested event is triggered in the user object uo_event because the Clicked event script for the Maximize button contains this statement:

```
Parent.TriggerEvent("Max_requested")
```

2 The script for the Max_requested event executes, maximizing the window due to this statement:

```
Parent.WindowState = Maximized!
```

Advantages and disadvantages of using user events

On the plus side: the technique is very flexible and powerful. Also, this type of communication can be synchronous (using the TriggerEvent function) or asynchronous (using the PostEvent function).

On the minus side: because there is no type checking, it is prone to error.

CHAPTER 11

Working with User Events

About this chapter

This chapter describes how to define and use user events.

Contents

Overview

Windows, user objects, and controls each have a predefined set of events. In nearly all cases, the predefined events are all you need.

But there are times when you want to declare your own event for a window, user object, or control. These events are called **user events**. Here are some possible uses of user events:

- You have placed a custom visual user object in a window and need to communicate between the user object and the window. This technique is described in Chapter 10, "Working with User Objects."
- You want to modify the way keystrokes are processed in your application. For example, you want the user to be able to press ENTER to move from SingleLineEdit to SingleLineEdit in a window (normally the user needs to press TAB). Or in a DataWindow control, you want the user to be able to press DownArrow and UpArrow to scroll among radio buttons in a DataWindow column (normally, pressing DownArrow or UpArrow goes to the next or preceding row).

  To do this, you define user events that correspond to Windows events that are not predefined for you by PowerBuilder.
- Within a window, you provide several ways to accomplish a certain task. For example, the user can click a button or select a menu item to update the database. In addition, when the user closes the window, you want to provide the option of updating the database. Instead of writing the code to update the database in each of these places, you can define a user event, then trigger that user event in each place you update the database.

Defining user events

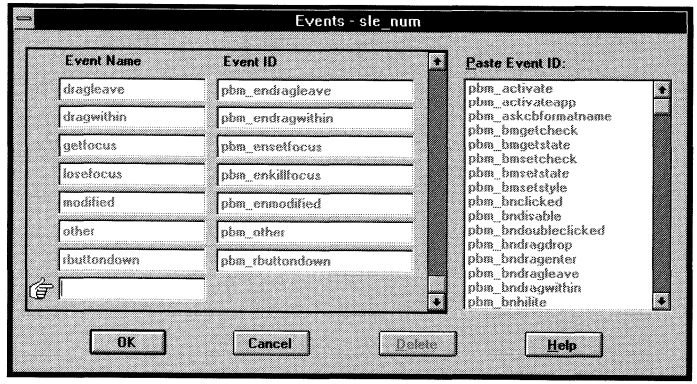
❖ To define a user event for a window, user object, or control:

1 Open the Window painter or User Object painter.

2 Select the window, user object, or control for which you want to declare the user event.

3 Select Declare➤User Events from the menu bar.

The Events dialog box displays.



The name and ID of the PowerBuilder and user events defined for the selected object or control display in the Event Name and Event ID listboxes. You cannot modify the predefined PowerBuilder events; they are protected. You can change the event ID for a user event or delete a user event, but you cannot change the name of a user event.

All PowerBuilder event IDs display in the Paste Event ID listbox. The list includes all the events directly supported by Windows plus 75 custom events (pbm_custom01 through pbm_custom75).

4 Enter the name of the event you are declaring in the Event Name box at the end of the list.

5 Double-click the ID of the event you want in the Paste Event ID listbox, as follows:

- If the event corresponds to a Windows message, choose the corresponding PowerBuilder event ID

  For more information, see "Understanding user event IDs" on page 309.

- If the event does not correspond to a Windows message, choose one of the PowerBuilder custom events (pbm_custom01 through pbm_custom75).

  For information about custom user events, see "Using custom events" on page 309.

PowerBuilder assigns the ID to the user event.

6 Click OK.

PowerBuilder closes the Declare Event dialog box and returns you to the painter.

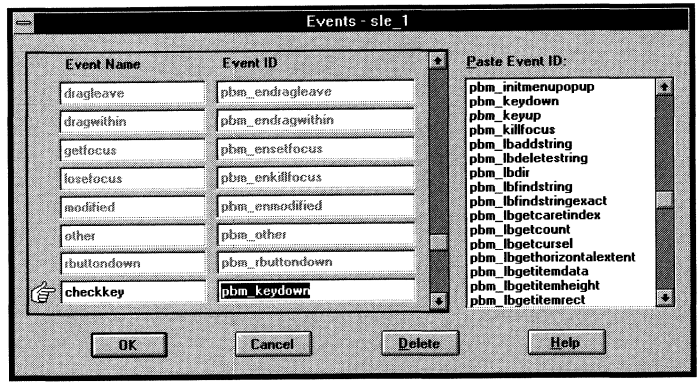
Example

For example, to declare an event for a SingleLineEdit control that is triggered when the user presses a key while in the control, you do the following:

1 Select the SingleLineEdit control in the window.

2 Select Declare➤User Events from the menu bar.

3 In the Events dialog box, specify a name (such as CheckKey) and double-click the event ID pbm_keydown, which maps to the appropriate Windows message.



Understanding user event IDs

The PowerBuilder naming convention for user event IDs is similar to the convention Windows uses to name messages. One or more Windows messages maps to each PowerBuilder event ID.

For example:

This Windows message	Maps to this event ID
wm_keydown	pbm_keydown
(button class message) bm_getcheck	pbm_bmgetcheck
(notification) bn_clicked	pbm_bnclicked

PowerBuilder uses the Windows wm_user message numbers to map custom events (pbm_custom*nn*), as described in the next section. PowerBuilder also has its own events; each of these has an ID. For example, the PowerBuilder event DragDrop has the event ID pbm_dragdrop.

For more information

For information about Windows messages, see your Microsoft Windows Software Developer's Kit (SDK) documentation.

The PowerBuilder online Help contains a mapping of Windows messages to PowerBuilder event IDs.

Using custom events

Custom user events (those that correspond to pbm_custom*xx* event IDs) are not meant to be used with standard controls such as CommandButtons and DropDownListBoxes. Custom user events are meant to be used only with DataWindow controls, windows, and user objects other than standard visual user objects (which behave like the built-in controls they inherit from). In order to define custom user events for the standard controls, you must have some Windows SDK experience and understand the following:

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). You need to know which controls have custom events, because these custom events overlap with the PowerBuilder custom events (pbm_custom*xx*). 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.

You need to make sure that you use a pbm_custom event ID that does not conflict with a custom event defined by Windows for the standard control. Otherwise, you might encounter unexpected behavior in your application.

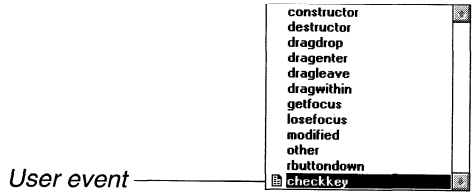
Using a user event

After defining the user event, you do the following:

- Write a script for the user event
- If necessary, write code that triggers the event

Writing the script

After declaring the user event, write a script that specifies the processing to occur when the event occurs. User events display in the Select Event listbox in the PowerScript painter below all the built-in events.



Triggering the event

If the user event corresponds to a Windows message (such as pbm_keydown in the example on page 308), the event will be triggered automatically when the corresponding action occurs in the application.

If the user event does not correspond to a Windows message (that is, if the event ID is one of the PowerBuilder custom events), you must trigger the event in a script using either the TriggerEvent function or the PostEvent function.

For information about TriggerEvent and PostEvent, see the *Function Reference.*

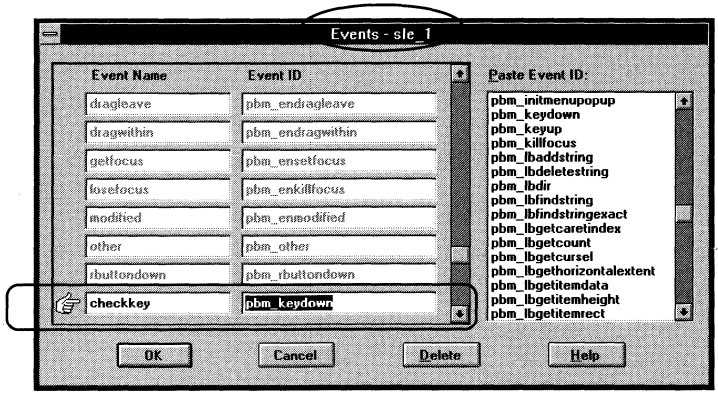
Examples of user event scripts

Here are two examples of using user events that map to Windows messages.

> **Example of *custom* user events**
> For an example of using *custom* user events, see "Using user events" in Chapter 10, "Working with User Objects."

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 custom event for each SingleLineEdit. Give the event any name you want, such as 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 custom event for sle_1, you could code:

```
// Script for user event 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 custom event for sle_2, you could code:

```
// Script for user event CheckKey,
// which is mapped to pbm_keydown

IF KeyDown(KeyEnter!) THEN       // Go to sle_3 if
   sle_3.SetFocus( )             // Enter pressed.
END IF
```

Situation You have a DataWindow with a column that uses the RadioButton edit style. You want to allow users to scroll through the RadioButtons when they press DownArrow or UpArrow (normally, pressing DownArrow or UpArrow scrolls to the next or preceding row).

Solution Declare a user event for the DataWindow control that maps to the event ID pbm_dwnkey (dwn is an acronym for "DataWindow notification") and write a script like the following for it.

```
// 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
```

```
// 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
```

PART FOUR

Working with Databases

This part describes how to use PowerBuilder to manage your database; how to build DataWindow objects to retrieve, present, and manipulate data in your applications; and how to use the Data Pipeline painter to copy data from one database to another.

CHAPTER 12

Managing the Database

About this chapter

Central to your PowerBuilder applications is the database. This chapter describes how you can 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.

Overview

You can use PowerBuilder to manage the following database components:

- Tables and columns
- Indexes
- Keys (if your DBMS supports them)
- Views

PowerBuilder also provides **extended attributes** that allow you to store application-based information about your table's columns in the database for use in your application. For example, you can define validation rules for a column. Once they are defined, anytime you use that column in a PowerBuilder application, each entry in the column is checked against the validation rule. If the entry doesn't pass validation, the user is informed.

You can also manage database security from within PowerBuilder.

About your DBMS

PowerBuilder supports many database management systems (DBMSs). For the most part, you work the same in PowerBuilder for each DBMS. But because each DBMS provides some unique functionality (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*.

How you work with the database

You do most of your database work using the Database painter. This painter is also the launching point for other painters concerned with database issues.

The Database painter

In the Database painter, you can:

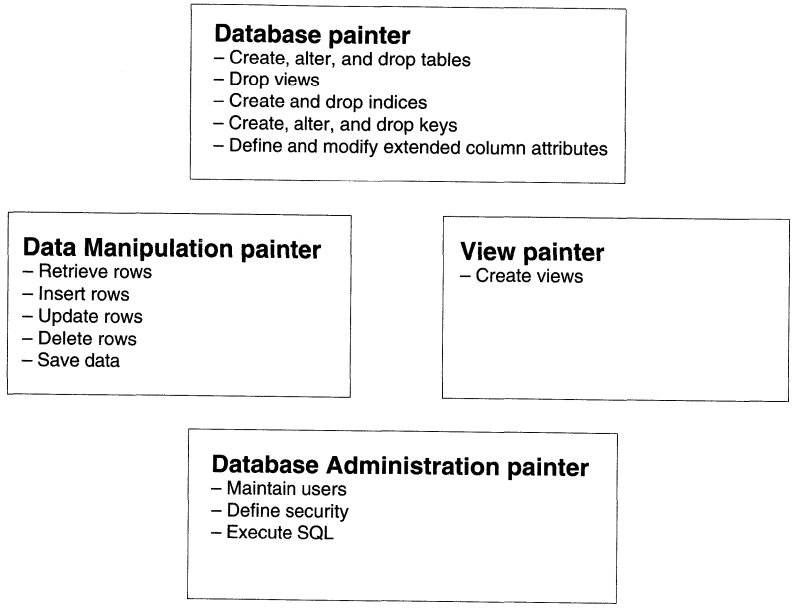
- Create, alter, and drop tables
- Drop views

- Create and drop indexes
- Create, alter, and drop primary and foreign keys
- Define and modify extended attributes for columns

Related painters

From the Database painter, you can also open three related painters:

- The Data Manipulation painter, where you retrieve and manipulate data from the database
- The View painter, where you create views
- The Database Administration painter, where you control access to the database and execute SQL directly

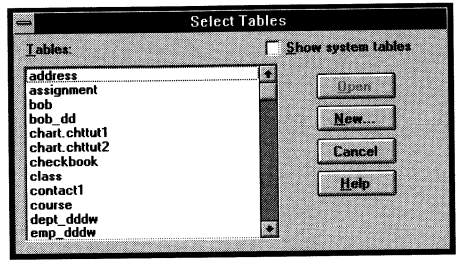


Using the Database painter

❖ To open the Database painter:

1 Click the Database button in the PowerBar or PowerPanel.

The Select Tables dialog box displays.

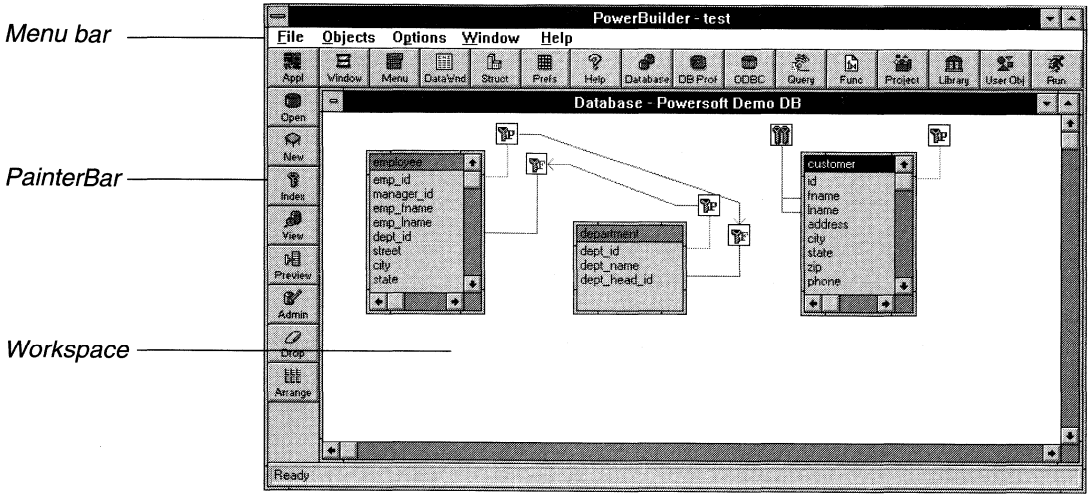


2 You can select tables to work on now or click Cancel to go to the painter workspace. You can open tables later from the workspace.

For more information, see "Working with tables" on page 329.

About the painter

Like the other PowerBuilder painters, the Database painter contains a menu bar, a customizable PainterBar, and a workspace.



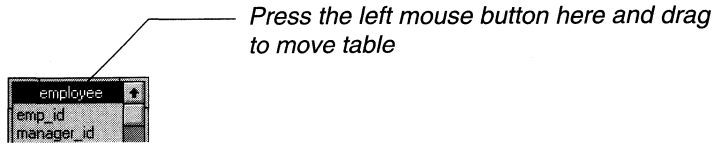
To work with database components, you open them in the workspace. Three database tables have been opened in the painter shown above: Employee, Department, and Customer. PowerBuilder displays the columns in the table.

PowerBuilder displays table columns and icons that mark a column or a set or columns as a primary key, a foreign key, or an index.

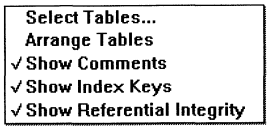
Vertical scrollbar
If the table contains more than eight columns, a vertical scrollbar displays. You can change the number of columns that display before the scrollbar is required by setting a different value for the Columns variable in the [Database] section of PB.INI.

Working with objects in the workspace

Moving objects You can move an object around the workspace by dragging it with the mouse. For example, to move a table, press the left mouse button on the title bar for the table and drag the table.

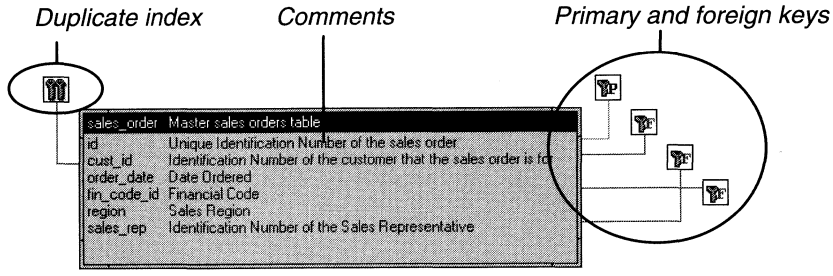


Resizing objects You can resize an object in the workspace by dragging a corner of it.



Displaying comments, indexes, and keys When a table is created or modified in PowerBuilder, you can assign comments to the table and to each of its columns. These comments can be useful to other developers who will be using the tables and need to understand the meaning of each of its columns. You can display comments for tables displayed in the workspace by pressing the right mouse button on the workspace background and selecting Show Comments from the popup menu.

Similarly, you can choose whether to display indexes and, if your DBMS supports them, primary and foreign keys (*referential integrity*) in the workspace. Your choices are recorded in PB.INI and will be used the next time you open the Database painter.



Keyboard alternatives

PowerBuilder provides keyboard alternatives to common workspace activities, as follows.

To	Do this
Select a table or view	Press TAB to move left to right among the opened tables and views, or press SHIFT+TAB to move right to left.
Display the Alter Table dialog box	Select the table for which you want to display the information, then press ENTER.
Browse an index or key	Press TAB or SHIFT+TAB to select the index or key to browse and press ENTER.
Scroll in the painter workspace	If you open more tables and views than can be displayed in the Database painter workspace at one time, you can scroll up or down to view all the tables and views. To scroll up, press UP ARROW. To scroll down, press DOWN ARROW.

Logging your work

As you work with your database, you will probably generate SQL statements. For example, as you define a new table in the Create Table dialog box in the Database painter, PowerBuilder is building 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 is building a CREATE INDEX statement.

You can record all SQL generated in a painter session in a log file if you want. This allows you to have a record of your work and also makes it easy to duplicate the work if you need to create the same or similar tables in another database.

In addition, each time you create or modify a database component such as a table, you have the option of generating the SQL *only* to a log file. You can then submit the statement to the database later if you want.

❖ To start logging your work in a text file:

1 Open the Database painter.

2 Select Options➤Start/Stop Log from the menu bar.

PowerBuilder opens a log file and displays the Activity Log icon at the bottom of the screen.

From now on, all your work will be saved in a temporary file.

❖ To view the log:

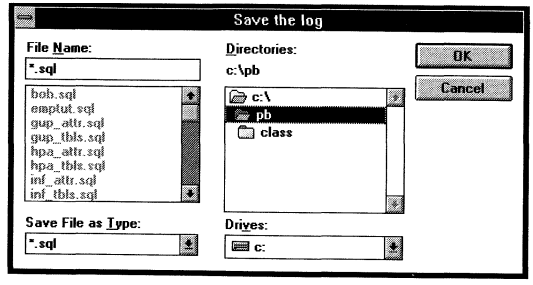
- Double-click the Activity Log icon.

  PowerBuilder opens a window showing the log. You can leave the window open as you work.

❖ To save the log to a permanent text file:

1 Select Options➤Save Log As from the menu bar.

The Save the Log dialog box displays.



2 Name the file and click OK. The default file extension is SQL, but you can change that if you want.

PowerBuilder saves the log as a text file with the specified name.

Submitting the log to your DBMS
You can later open a saved log file and submit it to your DBMS in the Database Administration painter.

For more information, see "Executing SQL" on page 375.

❖ To clear the log:

- Select Options➤Clear Log from the menu bar.

  PowerBuilder deletes all statements in the log, but leaves the log open.

❖ To stop the log:

- Select Options➤Start/Stop Log from the menu bar.
  or
  Close the Activity Log window.

  PowerBuilder closes the log. Your work will no longer be logged. However, the log file still exists. You can open it again and continue logging from where you left off.

Changing 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 (the information is recorded in the PB.INI file). You can change to a different database anytime.

For more about changing the database you are connected to, see *Connecting to Your Database*.

Creating and deleting a Watcom database

In PowerBuilder you work within an existing database. With one exception, creating and deleting databases are administrative tasks that are not performed directly in PowerBuilder.

For information about creating and deleting databases, see your DBMS documentation.

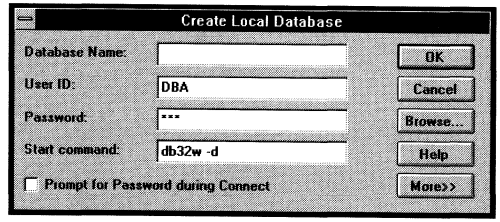
The one exception is that you can create and delete a local Watcom SQL database from within PowerBuilder.

❖ To create a local Watcom SQL database:

1 Open the Database painter.

2 Select File➤Create Database from the menu bar.

The Create Local Database dialog box displays.



3 In the Database Name box, specify the filename and path of the database you are creating. If you do not provide a path, the database will be created in the current directory or folder. If you do not provide a file extension, the database filename will be given the extension .DB.

4 Define other properties of the database as needed. Click the More button to define additional properties.

For complete information about filling in the dialog box, click the Help button.

5 Click OK.

What happens

When you click OK, PowerBuilder:

- 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.

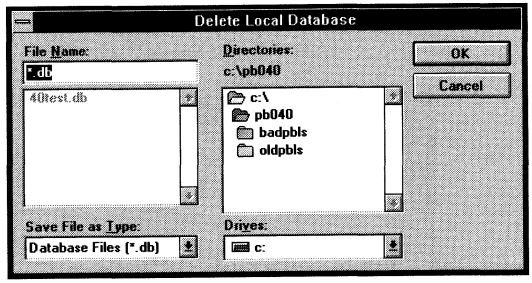
- Adds a data source to the ODBC.INI file. The data source has the same name as the database unless one with the same name already exists, in which case a suffix is appended.
- Creates a database profile and adds it to the PB.INI file. The profile has the same name as the database unless one with the same name already exists, in which case a suffix is appended.
- Connects to the new database.

♦ To delete a local Watcom SQL database:

1 Open the Database painter.

2 Select File➤Delete Database from the menu bar.

The Delete Local Database dialog box displays.



3 Select the database you want to delete.

You are prompted to confirm your action.

4 Click Yes to delete the database.

What happens

When you click Yes, PowerBuilder:

- Deletes the specified database
- Removes the data source from the ODBC.INI file
- Deletes the database profile from the PB.INI file

Working with tables

When you open the Database painter, the Select Tables dialog box lists all the tables and views in the current database that you have access to (including tables and views that were not created using PowerBuilder). You can create a new table or open existing tables.

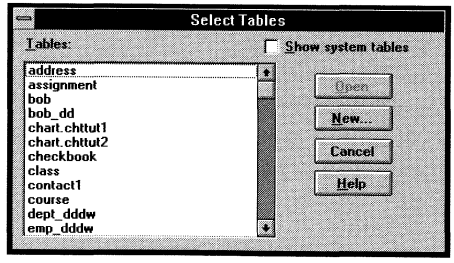
To suppress the table list
If you set the TableDir variable in the [Database] section of PB.INI to 0, PowerBuilder does not display the table list when you open the Database painter.

Opening a table

❖ To open a database table:

1 Click the Open button in the PainterBar.
or
Select Objects➤Tables from the menu bar.

The Select Tables dialog box displays.



2 To display system tables, select the Show System Tables checkbox.

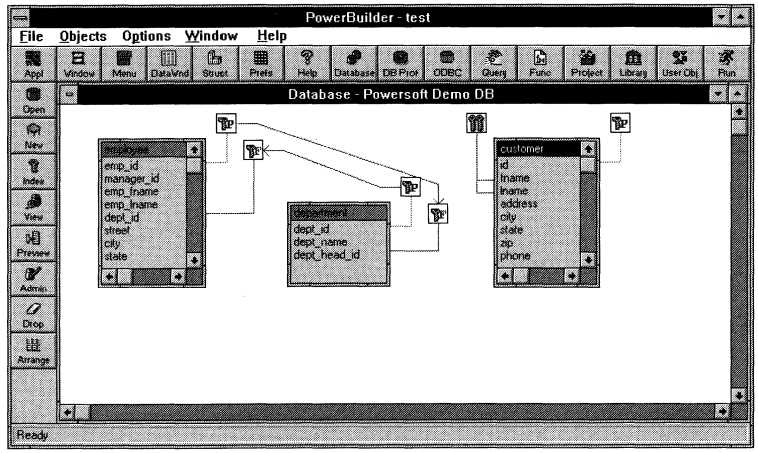
For more information, see "About the system tables" on page 331.

3 Select the tables to open by doing one of the following:

- Click the name of each table you want to open in the list displayed in the Select Tables dialog box, then click Open to open the selected tables.

- Double-click the name of each table you want to open. Each table is opened immediately. Then click Cancel to close the Select Tables dialog box.

Representations of the selected tables display in the Database painter workspace.



Three database tables have been opened in the painter shown above: Employee, Department, and Customer. PowerBuilder displays the columns in the table.

PowerBuilder caches the table list

The first time PowerBuilder has to build a list of tables in the database (for example, when you open the Database painter and the Select Tables dialog box displays), PowerBuilder retrieves the list of tables from the database and displays it. PowerBuilder saves the list internally and uses it the next time a list of tables needs to be displayed; this saves you a lot of time, because PowerBuilder doesn't have to regenerate what could be a very large list.

By default, the list is refreshed every 30 minutes (1800 seconds). To specify a different refresh rate, set the Database variable TableListCache to the number of seconds you want to elapse before a refresh is required. For example, to have PowerBuilder refresh the list every 15 minutes, set TableListCache to 900.

Changing the colors of the tables

You can change the colors used by the Database painter to display table information by assigning values to variables in the [Database] section of your PB.INI file, as follows:

Variable = red green blue

where *red*, *green*, and *blue* are numbers from 0 to 255 that specify the amount of each primary color.

You can set colors separately for each component in the table representations as follows:

Variable	Applies to
TableHeaderColor	Background of the header (default: dark gray, or **128 128 128**)
TableHeaderTextColor	Text in the header, used for column names and other information (default: black, or **0 0 0**)
TableDetailColor	Background of the detail area, which shows column names and other information (default: light gray, or **192 192 192**)
TableColumnNameTextColor	Column names in the detail area (default: black, or **0 0 0**)
TableDetailTextColor	Other information (such as comments) in the detail area (default: blue, or **0 0 128**)

For example, the following statement displays column names in red:

```
TableColumnNameTextColor=255 0 0
```

About the system tables

By default, PowerBuilder shows only user-created tables in the Select Tables dialog box. If you select Show System Tables in the dialog box, PowerBuilder also shows system tables. There are two kinds of system tables:

- System tables provided by your DBMS (see your DBMS documentation for details)
- PowerBuilder system tables

PowerBuilder stores 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 in your database. These system tables are called the Powersoft **repository**.

There are five PowerBuilder system tables in the repository:

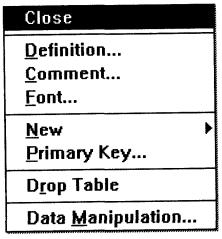
- PBCatCol
- PBCatEdt
- PBCatFmt
- PBCatTbl
- PBCatVld

For more about the tables in the Powersoft repository, see the appendix, "The Powersoft Repository."

Closing a table

You can remove a table from the workspace. (This action only removes the table from the Database painter workspace. It does not drop the table from the database.)

❖ To close a table:



1. Press the right mouse button on the table's name in the workspace.
2. Select Close from the popup menu.

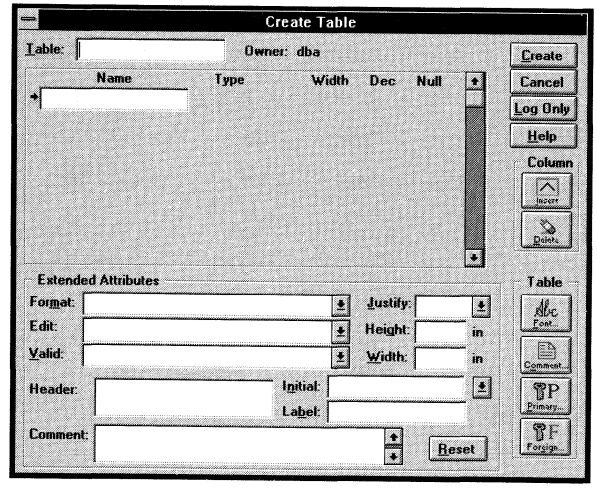
Creating a table

You can create tables from within PowerBuilder.

❖ To create a table in the current database:

1 Click the New button in the PainterBar.

The Create Table dialog box displays.



2 Enter the following required information:

- The name of the table you are creating.
- The name of each column in the table.
- The data type and other required information about the data in the column (such as column width, number of decimal places, and whether NULL values are allowed). All data types supported by the current DBMS are displayed in the Type dropdown listbox.

> **Specifying NULL**
> No in the NULL dropdown listbox means NULLs are not allowed; users must supply a value.

3 Optionally, you can specify the following information now or later when you modify the table:

- Fonts used to display the headings and data for the table when it is used in a DataWindow object (Font button in the Table group)

For more about fonts, see "Specifying fonts for the table" on page 336.

- Comments about the table (Comment button in the Table group)
- Keys (Primary and Foreign Key buttons in the Table group)

  For more about keys, see "Working with primary and foreign keys" on page 344.

- Extended attributes for columns, which specify how data for the columns is displayed and validated and specifies the text that is used to label the columns in DataWindow objects

  For more about extended attributes, see "Specifying extended column attributes" on page 337.

4 Save your work by doing one of the following:

- Click the Create button. PowerBuilder submits to the DBMS the CREATE TABLE statement it generated. The table is created in the DBMS.
- Click the Log Only button. The CREATE TABLE statement is written only to the log file; it is *not* submitted to the DBMS. You can later submit the statement if you choose.

  For more information, see "Logging your work" on page 324.

You return to the Database painter workspace.

Altering a table

Once a table has been created, you can do the following:

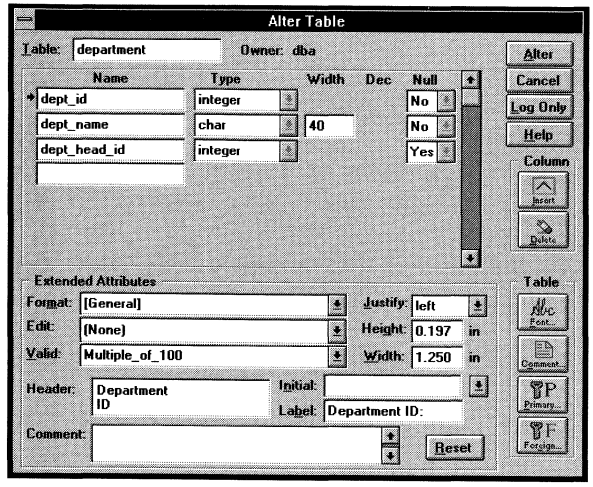
- Append columns to the table. Appended columns must allow NULL values.
- In some DBMSs, you can also increase the number of characters allowed for data in an existing character column and allow NULL values. However, you cannot prohibit NULL values in a column that had been defined to allow NULL values.
- Add or modify all PowerBuilder-specific information about the table and its columns.

❖ To alter a table:

1 Open the table in the Database painter.

2 Double-click the name of the table as represented in the workspace.

The Alter Table dialog box displays.



3 Make your changes.

4 Do one of the following:

- Click the Alter button. PowerBuilder submits to the DBMS the ALTER TABLE statement that it generated. The table is altered in the DBMS.
- Click the Log Only button. The ALTER TABLE statement is written only to the log file; it is *not* submitted to the DBMS. You can later submit the statement if you choose.

For more information, see "Logging your work" on page 324.

You return to the Database painter workspace.

Specifying fonts for the table

When you create or alter a table, you can choose the fonts used to display information from the table in a DataWindow object in your application.

If you don't specify fonts for a table, PowerBuilder uses the font information specified in the application object.

For more information, see Chapter 2, "Working with Applications."

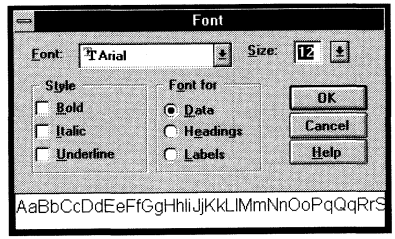
❖ To specify fonts for the table:

1 Press the right mouse button on the table name in the painter workspace. Select Font from the popup menu.

or

Click the Font button in the Create Table or Alter Table dialog box.

The Font dialog box displays.



2 Specify the fonts. You can specify the font, point size, and style for the following:

- Data—the values retrieved from the database
- Headings—the column identifiers used in grid, tabular, and n-up DataWindow objects
- Labels—the column identifiers used in freeform DataWindow objects

> **Choosing fonts**
> Make sure that fonts you pick will be available to your users when you distribute your application. If you pick a font that is on your computer, but not on your user's computer, the operating environment will use the font it considers closest to the one you specified. It might look very different.

3 Click OK to update the repository with the new font information.

Specifying extended column attributes

In addition to providing the information required to create a table, you can specify extended attributes for each column in the table. An extended attribute is PowerBuilder-specific information that enhances the definition of the column. PowerBuilder provides the following extended attributes for a column:

Extended attribute	Meaning
Comment	Describes the column. Whenever the table is opened in the Database painter, you can see the comment to understand the purpose of the column.
Heading and label	Text that is used in DataWindow objects to identify columns.
Alignment and size	How data is aligned and how much space to allocate to the data in a DataWindow object.
Display format	How the data is formatted in a DataWindow object.
Edit style	How the column is presented to the user in a DataWindow object. For example, you can display column values as radio buttons or in a dropdown listbox.
Validation rule	Criteria that a value must pass in order to be accepted in a DataWindow object.

Overriding definitions
You can override the extended attributes specified in the Database painter for a particular DataWindow object in the DataWindow painter.

You can specify extended attributes using a popup menu or using the Create or Alter Table dialog box.

❖ To specify extended attributes using a popup menu:

1 Open the table in the painter workspace.

2 Click the right mouse button on the column.

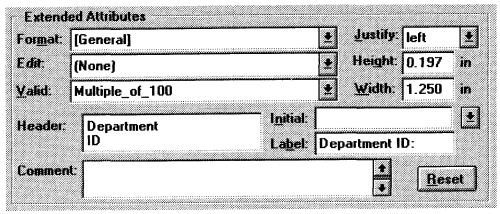
The column's popup menu displays.

3 Select the attribute from the menu.

4 Specify the attribute, as described next.

❖ To specify extended attributes using the Create or Alter Table dialog box:

1 For an existing table, double-click the table name to display the Alter Table dialog box. For a new table, open the Create Table dialog box. In either case, the bottom of the dialog box specifies the extended attributes.



2 Select the column you are modifying and specify the extended attributes, as described next.

3 Click Create or Alter to save your changes.

How the information is stored

Extended attributes are stored in the repository (in the PowerBuilder system tables in the database). PowerBuilder uses the information to display and validate data in DataWindow objects.

When using views

When you create a view, the extended attributes defined for the table columns used in the view are used by default.

Accessing column comments in a DataWindow object

A column's comments are generated as the column's Tag attribute in a DataWindow object. You can access the comments through the Describe PowerScript function. For example, to access the comments for the column EmpID in a DataWindow object, code:

```
string   s
s = dw_emp.Describe("EmpID.Tag")
```

You can use this technique to display comments to your users to make your application easier to use.

Specifying headings and labels

You can specify the text that identifies the column in a DataWindow object:

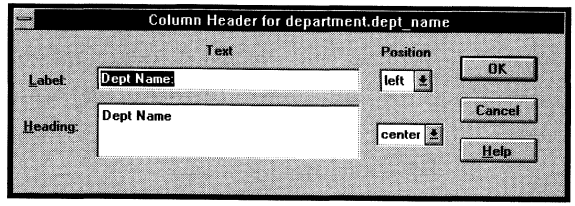
- Label text is used in freeform DataWindow objects
- Heading text is used in tabular, grid, group, and n-up DataWindow objects

By default, PowerBuilder uses as labels and headings the column names, replacing any underscore characters with spaces and capping each word in the name. For example, the default label or heading for the column Dept_name is Dept Name.

❖ To specify a different heading or label for a column:

1 Select Header from a column's popup menu.

The Column Header dialog box displays.



2 Type the text you want in the Heading and Label boxes. To define multiple-line headings, press CTRL+ENTER to begin a new line.

3 Specify where the text displays by choosing a value in the Position box.

Another method
You can also specify headings and labels in the Create Table and Alter Table dialog boxes.

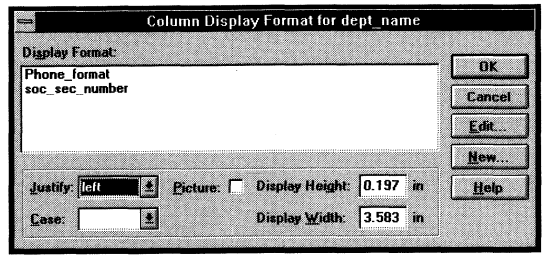
Specifying alignment and size

You can specify how you want to align and size a column in a DataWindow object.

❖ To specify the alignment and size of data:

1 Select Display from a column's popup menu.

The Column Display Format dialog box displays.



2 In the bottom of the dialog box, specify whether you want the data displayed left-aligned, right-aligned, or centered in the column by selecting a value in the Justify box.

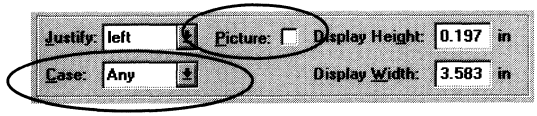
3 If necessary, adjust the height and width of the column when used in a DataWindow object.

You will seldom need to change these values. PowerBuilder uses the size that is appropriate for the data in the column as the default column size. Except when you are using picture columns, or if you have specified certain kinds of edit styles, you should accept the default display size and if necessary change the size of the column in the DataWindow painter.

Specifying additional properties for character columns

If the column is a character column, there are two additional attributes for a column:

- Case
- 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:**

1 Select Display from a column's popup menu.

The Column Display Format dialog box displays.

2 Select a value in the Case dropdown listbox to specify how the data should be displayed:

- 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 contains names of picture files (BMP or WMF files).

❖ **To specify that column values are names of picture files:**

1 Select Display from a column's popup menu.

The Column Display Format dialog box displays.

2 Select the Picture checkbox.

When the Picture checkbox is selected, PowerBuilder expects to find bitmap (BMP) or Windows metafile (WMF) filenames in the column and displays the contents of the picture file—not the name of the file—in DataWindow objects.

Because PowerBuilder cannot determine the size of the image until execution time, it sets both display height and display width to 0 when you check the Picture checkbox. You should enter the size and optionally the justification for the picture.

Using display formats, edit styles, and validation rules

The three most powerful extended attributes—display formats, edit styles, and validation rules—allow you to build robust, easy-to-use DataWindow objects. 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.
- You can associate an edit style with a Status column so that the column displays as three radio buttons, each one corresponding to one of the three allowable statuses. Users simply click a button to specify a status.
- You can associate a validation rule with a Salary column so that users can only enter a value within a particular range.

You can apply display formats, edit styles, and validation rules either in the Database painter or in the DataWindow painter.

For more about these extended attributes, see Chapter 15, "Displaying and Validating Data."

Working with indexes

In the Database painter workspace, you can create as many single- or multi-valued indexes for a database table as you need and 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.

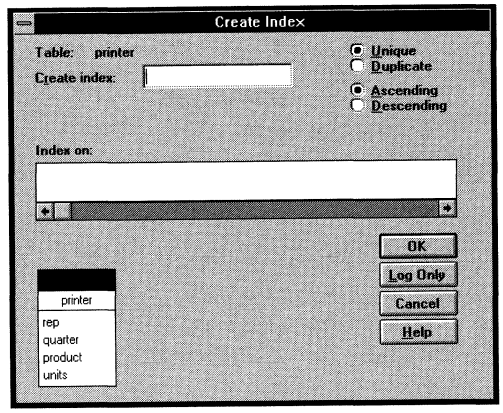
Creating an index

❖ To create an index:

1 Select the table that you want to create an index for.

2 Click the Index button in the PainterBar.
or
Select New➤Index from the table's popup menu.

The Create Index dialog box displays. Some of the information is DBMS-specific.



3 Enter a name for the index.

4 Select whether or not to allow duplicate values for the index.

5 Specify any other information required for your database (for example, in SQL Server specify whether the index is clustered, and in Watcom SQL specify the order of the index).

6 Click the names of the columns that make up the index.

The selected column names display in the Index On box.

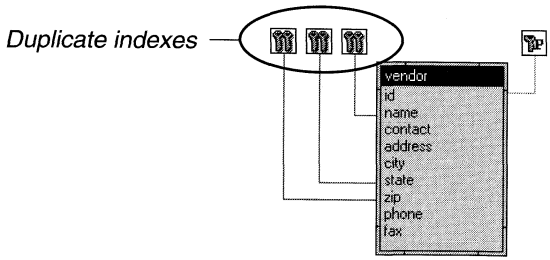
7 Save your work by doing one of the following:

- Click the OK button. PowerBuilder submits to the DBMS the CREATE INDEX statement it generated. The index is created in the DBMS.
- Click the Log Only button. The CREATE INDEX statement is written only to the log file; it is *not* submitted to the DBMS. You can later submit the statement if you choose.

For more information, see "Logging your work" on page 324.

You return to the Database painter workspace.

The new index is shown as a key connected to each column in the index.



Browsing an index

❖ To look at the definition of an existing index:

1 Open and select the table with the index you want to look at.

2 Double-click the icon representing the index.
or
Position the mouse pointer over the icon, display the popup menu, and select Browse from the popup menu.

PowerBuilder displays the Browse Index dialog box, which shows the index's definition.

Changing an index
You cannot alter an existing index. To make a change, delete the index and then create a different one.

Dropping an index

❖ **To drop an index from a table:**

1 Open and select the table with the index you want to drop.

2 Select the index and click the Drop button.
or
Position the mouse pointer over the icon representing the index, display the popup menu, and select Drop Index from the popup menu.

PowerBuilder prompts you to confirm the drop, then generates a DROP INDEX statement and submits it to the DBMS.

Working with primary and foreign keys

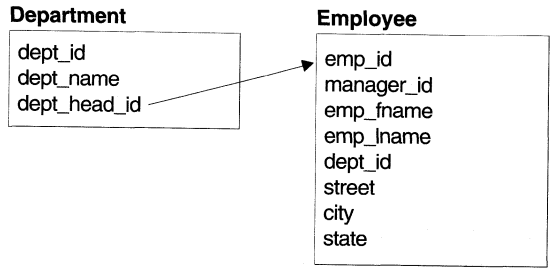
If your DBMS supports primary and foreign keys, you can work with the keys in PowerBuilder. When you open a table with keys, PowerBuilder gets the information from the DBMS and displays it in the painter workspace.

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. If you use keys, 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, 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. That is, 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

In the Database painter you can do the following:

- 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 keys
- Alter keys
- 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.

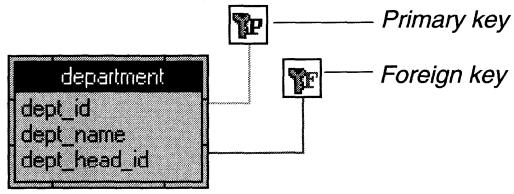
For complete information about using keys with your DBMS, see your DBMS documentation.

Viewing keys

When you open and expand a table containing primary and/or foreign keys, PowerBuilder displays the keys in the workspace. The keys are shown as icons with lines connected to the table.

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 can't see the lines
If the color of your window background makes it hard to see the lines for the keys and indexes, you can edit the [Database] section of your PB.INI file to change the colors used for the lines, as follows:

IndexKeyLineColor=*red green blue*
ForeignKeyLineColor=*red green blue*
PrimaryKeyLineColor=*red green blue*

where *red*, *green*, and *blue* are numbers from 0 to 255 that specify the amount of each primary color. For example, the following statement makes the line used with primary keys red:

```
PrimaryKeyLineColor=255 0 0
```

Opening related tables

When working with tables containing keys, you can easily open related tables.

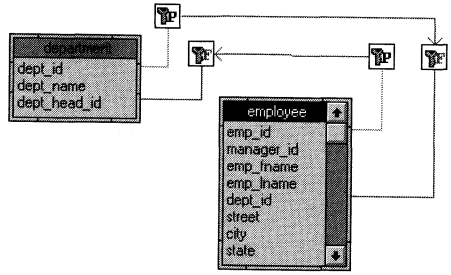
❖ To open the table that a particular foreign key references:

1 Open and expand the table containing the foreign key.

2 Click the right mouse button on the button representing the foreign key.

3 Select Open Referenced Table from the popup menu.

PowerBuilder opens and expands the table referenced by the foreign key.



❖ To open all tables referencing a particular primary key:

1 Open and expand the table containing the primary key.

2 Click the right mouse button on the button representing the primary key.

3 Select Open Dependent Table(s) from the popup menu.

PowerBuilder opens and expands all tables in the database containing foreign keys that reference the selected primary key.

Defining primary keys

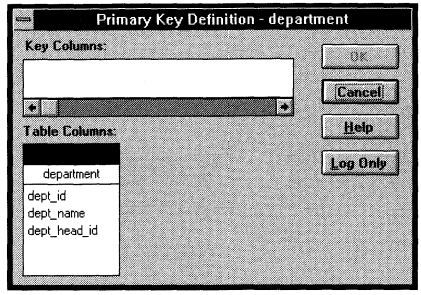
If your DBMS supports primary keys, you can define them in PowerBuilder.

❖ To define a primary key:

1 Display the Create Table dialog box for a table you are now creating or the Alter Table dialog box for an existing table.

2 Click the Primary Key button in the Table group.

The Primary Key Definition dialog box displays. Some of the information in the dialog box is DBMS-specific.



3 Select each of the columns comprising the primary key in the Table Columns box.

PowerBuilder displays the selected columns in the Key Columns box.

4 If you want, you can reorder the columns in the key by dragging them with the mouse.

5 Specify any information required by your DBMS (such as a name for the primary key).

For DBMS-specific information, see your DBMS documentation.

6 Click OK.

You return to either the Create Table or Alter Table dialog box.

7 Save your work by doing one of the following:

- Click the Create or Alter button. PowerBuilder submits the CREATE TABLE or ALTER TABLE statement to the DBMS.
- Click the Log Only button. The SQL statement is written only to the log file; it is *not* submitted to the DBMS. You can later submit the statement if you choose.

For more information, see "Logging your work" on page 324.

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 that you separately create a unique index to support the primary key before populating the table with data.

To see what your DBMS does, see your DBMS documentation.

Defining foreign keys

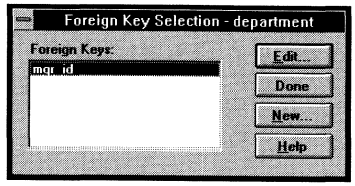
If your DBMS supports foreign keys, you can define them in PowerBuilder.

❖ To define a foreign key:

1 Display the Create Table dialog box for a table you are now creating or the Alter Table dialog box for an existing table.

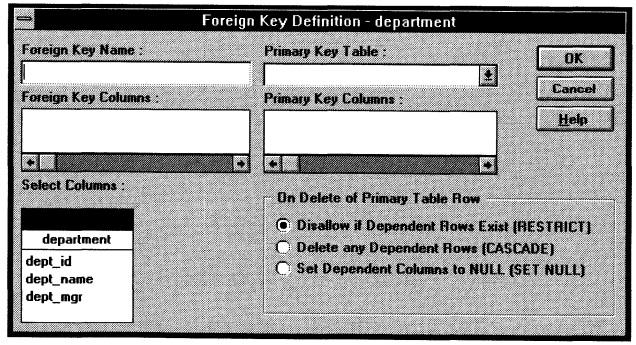
2 Click the Foreign Key button in the Table group.

The Foreign Key Selection dialog box displays. It lists all foreign keys defined for the current table.



3 Click New to define a new foreign key.

The Foreign Key Definition dialog box displays. Some of the information in the dialog box is DBMS-specific.



4 Name the foreign key in the Foreign Key Name box.

5 Select the columns for the foreign key in the Select Columns listbox.

The selected columns display in the Foreign Key Columns box. You can reorder the columns by dragging them with the mouse.

6 In the Primary Key Table listbox, select the table containing the primary key referenced by the foreign key you are defining.

PowerBuilder displays the selected table's primary key in the Primary Key Columns box.

7 Specify any information required by your DBMS (such as a delete rule).

For DBMS-specific information, see your DBMS documentation.

8 Click OK.

You return to the Foreign Key Selection dialog box.

9 Click Done.

You return to the Create Table or Alter Table dialog box.

10 Save your work by doing one of the following:

- Click the Create or Alter button. PowerBuilder submits the CREATE TABLE or ALTER TABLE statement to the DBMS.
- Click the Log Only button. The SQL statement is written only to the log file; it is *not* submitted to the DBMS. You can later submit the statement if you choose.

For more information, see "Logging your work" on page 324.

Dropping a key

You can drop keys from within PowerBuilder.

❖ To drop a key:

1 Open the table containing the key.

2 Select the key in the workspace.

3 Click the Drop button in the PainterBar.

You are prompted to confirm the deletion.

4 Click Yes.

PowerBuilder submits the ALTER TABLE statement to the DBMS.

Dropping a table

You can drop a table from within PowerBuilder.

❖ To drop a table:

1 Open and select the table you want to drop.

2 Click the Drop button in the PainterBar.

You are prompted to confirm the deletion.

3 Click Yes.

PowerBuilder submits the DROP TABLE statement to the DBMS.

Deleting orphaned table information

If you drop a table outside PowerBuilder, information will remain in the repository about the table, including extended attributes for the columns.

❖ To delete orphaned table information from the repository:

1 Select Options➤Synchronize PB Attributes from the menu bar.

You are prompted to confirm your action.

2 Click Yes.

PowerBuilder synchronizes the attributes.

Working with views

You can define and manipulate views in PowerBuilder. Typically you use 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 view of all the columns in the Employee table except Salary. Users of the view can see and update all information except the employee's salary.
- To combine information from multiple tables for easy access.

About views
Information in a view is not stored separately in the database. Every time the view is referred to, the associated SELECT statement is executed to retrieve the data.

For more information about using views, see your DBMS documentation.

In PowerBuilder, you can create single- or multiple-table views and can use a view to create a new view. You open and manipulate existing views in the Database painter. You define views in the View painter.

Opening a view

You open a view in the Database painter the same way you open tables.

❖ **To open a view:**

1 Click the Open button.
or
Select Objects➤Tables from the menu bar.

The Select Tables dialog box displays. This dialog box displays views along with tables.

2 Select the view.

Creating a view

❖ To create a view:

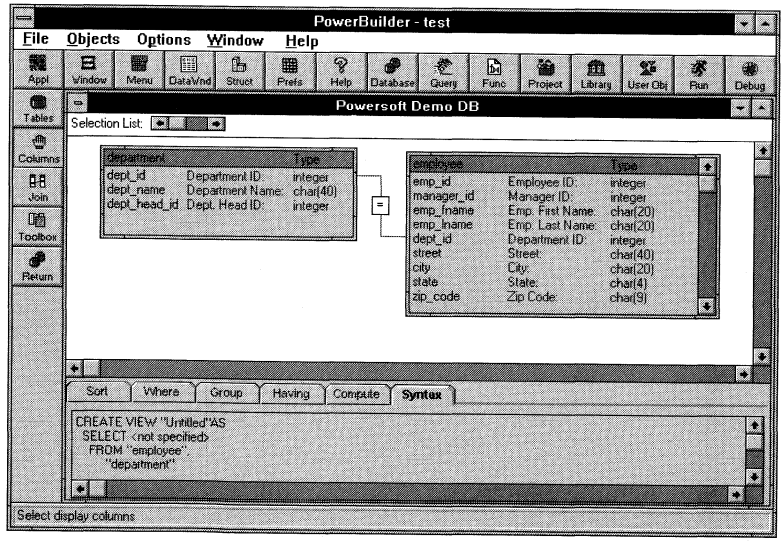
1 From the Database painter, click the View button in the PainterBar.
or
Select Objects➤New➤View from the menu bar.

The Select Tables dialog box displays listing all tables and views defined in the database.

2 Select the tables and/or 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.
- 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.



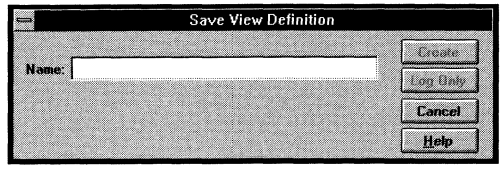
3 Select the columns to include in the view (see page 356).

4 Join the tables if there is more than one in the view (see page 358).

5 Specify WHERE, GROUP BY, and HAVING criteria if appropriate (see page 359).

6 When the view has been completed, click the Return button.

You are prompted to name the view.



7 Provide a name for the view and do one of the following:

- Click the Create button. PowerBuilder submits the CREATE VIEW statement to the DBMS.
- Click the Log Only button. The SQL statement is written only to the log file; it is *not* submitted to the DBMS. You can later submit the statement if you choose.

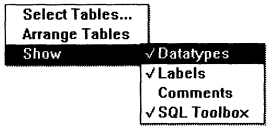
  For more information, see "Logging your work" on page 324.

You return to the Database painter. If you clicked Create to create the view, the view is displayed in the workspace.

Specifying what is displayed

By default, tables in the workspace display the label and data type of each column (the label information comes from the Powersoft repository). You can choose to hide this information as well as to display comments that have been defined for the columns in the repository. You can also choose to hide the SQL toolbox to give you more room to see the tables.

❖ To hide or display comments, data types, labels, and the SQL toolbox:



1 Point to any unused area of the workspace and select Show from the popup menu.

A cascading menu displays.

2 Select or deselect Datatypes, Labels, Comments, or SQL Toolbox as needed.

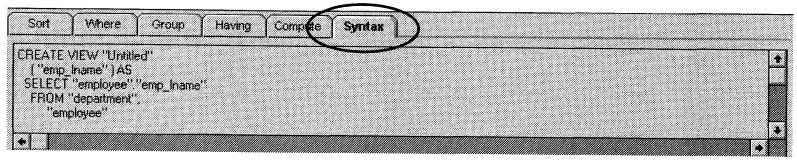
Shortcut
You can also click the Toolbox button in the PainterBar to toggle the display of the SQL toolbox.

Displaying a view's SQL statement

You can display the SQL statement that defines a view. How you do it depends on whether you are in the View painter creating a new view, or in the Database painter and want to look at the definition of an existing view.

❖ To display the SQL statement from the View painter:

- Display the Syntax tab in the View painter.

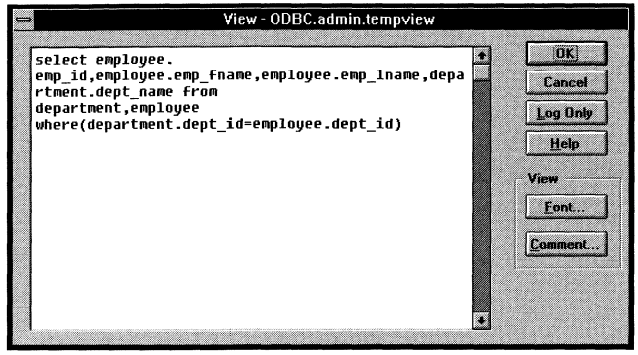


PowerBuilder displays the SQL it is generating. The display is updated each time you make a change to the view.

❖ To display the SQL statement from the Database painter:

1. Open the view.
2. Double-click the name of the view in its representation in the painter workspace.
   or
   Select Definition from the view's popup menu.

PowerBuilder displays the View Definition dialog box showing the completed SELECT statement used to create the view.



From this dialog box, you can modify the fonts and comments associated with the view.

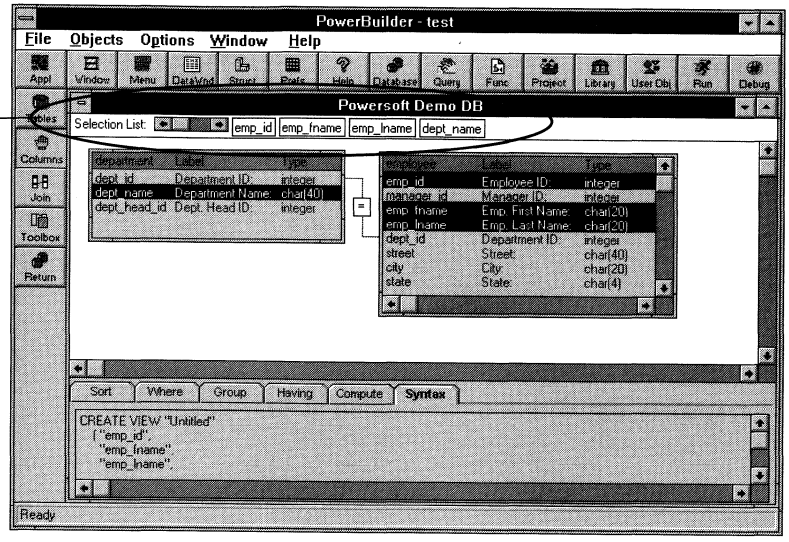
Dialog box is read-only
You cannot alter a view in this dialog box. To alter a view, drop it and then create it again.

Selecting columns for the view

Click the columns you want to include in the view from the table representations in the workspace.

Columns that will be included in the view are highlighted and shown in the Selection List above the tables.

Columns that will be included

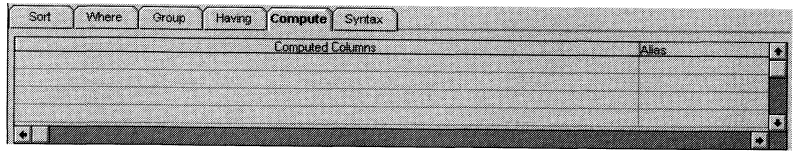


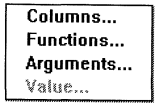
❖ To select all columns from a table:

- Select Select All from the table's popup menu.

❖ To include a computed column in the view:

1 Display the Compute tab in the View painter.





2 Enter the expression for the computed column. You can paste the following into the expression from the popup menu:

- Names of columns in the tables used in the view

- Functions provided by the DBMS

> **About the functions**
> You can use functions that are provided by your DBMS here. When you select Functions from the popup menu, PowerBuilder displays functions defined by your DBMS (these are *not* the same as the functions provided by PowerBuilder that you use in the DataWindow painter workspace). For more information about any of these functions, see your DBMS documentation.

3 Name the computed column in the Alias box.

Joining tables

If the view contains more than one table, you should join the tables on their common columns. When the View painter initially is opened for a view containing more than one table, PowerBuilder makes its best guess as to the join columns, as follows:

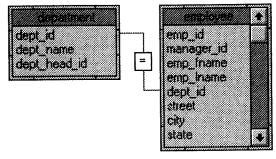
- If there is a primary/foreign key relationship between the tables, PowerBuilder automatically joins them.
- If there are no keys, PowerBuilder tries to join tables based on common column names and types.

❖ To join tables:

1 Click the Join button.
or
Select Objects➤Joins from the menu bar.

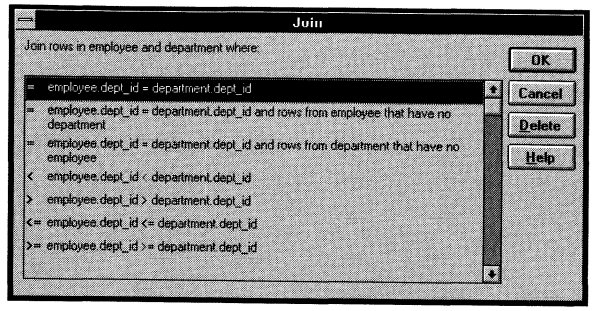
The pointer changes to the Join indicator.

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 your DBMS documentation.

Specifying WHERE, GROUP BY, and HAVING criteria

In the SELECT statement associated with a view definition, you can use the following:

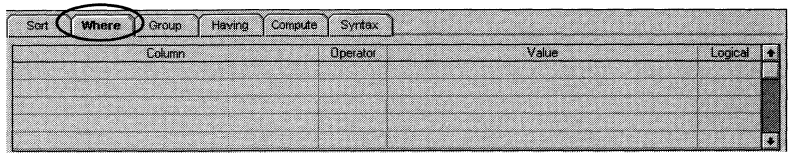
- A WHERE clause to limit the view
- A GROUP BY clause to group the retrieved data
- A HAVING clause to limit the groups specified in the GROUP BY clause

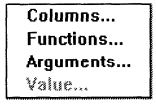
Defining WHERE criteria

You limit the rows that are retrieved in the view by specifying selection criteria that correspond to the WHERE clause in the SQL statement.

❖ To define WHERE criteria:

1 Display the Where tab in the View painter.



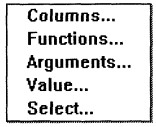


2 Specify the left-hand side of the expression in the Column box. You can paste the following from the popup menu in the Column box:

- Names of columns in the tables used in the view
- Functions supported by the DBMS

> **About the functions**
> You can use functions that are provided by your DBMS here. When you select Functions from the popup menu, PowerBuilder displays functions defined by your DBMS (these are *not* the same as the functions provided by PowerBuilder that you use in the DataWindow painter workspace). For more information about any of these functions, see your DBMS documentation.

3 Select an operator from the dropdown listbox in the Operator box.



4 Specify the right-hand side of the expression in the Value box. You can paste the following from the popup menu:

- Names of columns in the tables used in the view.
- Functions supported by the DBMS.
- Values from the database if you specified a column in the Column box, as follows: if you select Values from the popup menu, PowerBuilder will retrieve all the values in the database for the column specified in the Column box. You can select one of the values to paste into the Value box.

- A SELECT statement for the expression (a nested subquery). You go to another painter where you can paint the SELECT statement. When you have completed it, you return to the Where tab.

5 If your WHERE clause consists of more than one statement, select the AND or OR logical operator from the Logical listbox.

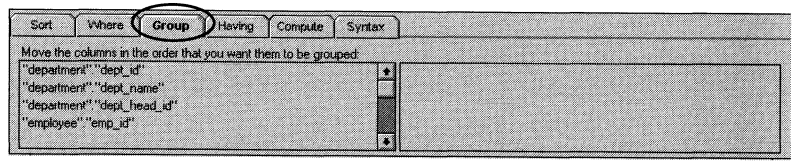
Defining GROUP BY criteria

You can group the retrieved rows. Grouping in a SQL SELECT statement causes all rows in the tables meeting the same grouping criteria to be combined together into one row before being brought down into PowerBuilder.

For more about using GROUP BY in a SQL SELECT statement, see your DBMS or SQL documentation.

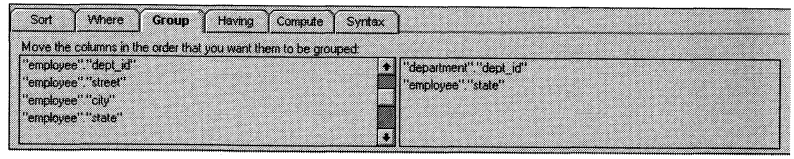
❖ To define GROUP BY criteria:

1 Display the Group tab in the View painter.



PowerBuilder displays the columns in the tables used in the view.

2 Drag the columns you want to use to form the groups into the listbox in the right side of the Group tab. You can choose as many columns as you want.



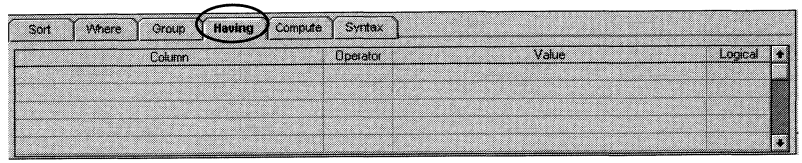
The columns are grouped in the order shown. For example, if the first column is Dept_id and the second is State, then the grouping will be by dept_id and within dept_id by state.

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 $40,000.

❖ **To define HAVING criteria:**

1 Display the Having tab in the View painter.



2 Enter the criteria for the HAVING clause the same way you enter or paint WHERE criteria (see "Defining WHERE criteria" on page 360).

Dropping a view

❖ **To drop a view:**

1 Select the view you want to drop in the Database painter workspace.

2 Click the Drop button in the PainterBar.

PowerBuilder prompts you to confirm the drop, then generates a DROP VIEW statement and submits it to the DBMS.

Exporting table or view syntax

You can export the syntax for a table or view to the log. This feature is useful when you want to create a backup definition of the table or view before you alter it or when you want to create the same table or view in another DBMS.

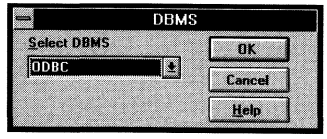
To export to another DBMS, you must have the PowerBuilder interface for that DBMS.

❖ To export the syntax of an existing table or view to a log:

1 Select the table or view in the painter workspace.

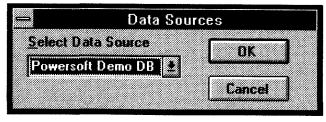
2 Select Objects➤Export Table/View Syntax To Log from the menu bar.

If you selected a table, the DBMS dialog box displays.



3 Select the DBMS to which you want to export the syntax.

4 If you selected ODBC, specify a data source.



5 Supply any information you are prompted for.

PowerBuilder exports the syntax to the log. Powersoft repository 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 about using the log, see "Logging your work" on page 324.

Manipulating data

As you work on the database, you will often want to look at existing data or create some data for testing purposes. Also, you will want to test display formats, validation rules, and edit styles on real data.

PowerBuilder provides the Data Manipulation painter for such purposes. In this painter, you can:

- Retrieve and manipulate database information
- Save the contents of the database in a variety of formats (such as Excel, dBASE, or Lotus 1-2-3)

Opening the Data Manipulation painter

❖ **To open the Data Manipulation painter:**

1 Select the table or view whose data you want to manipulate.

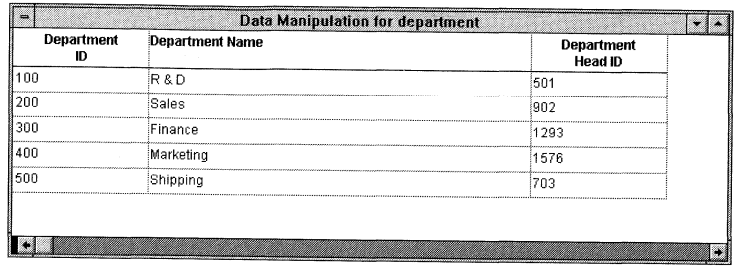
2 Do one of the following:

- Click the Preview button in the PainterBar. This will display the data in a grid format.
- Select Objects➤Data Manipulation from the menu bar and choose the format option from the cascading menu that displays:
  - Grid, which uses a rigid format
  - Tabular, which uses rows and columns
  - Freeform, which resembles a freeform data entry form

The Data Manipulation painter opens and all rows are retrieved. As the rows are being retrieved, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval.

The Data Manipulation painter is actually a DataWindow. 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.

The window shown below is in the grid format:



Retrieving data

❖ To retrieve rows from the database:

- Click the Retrieve button in the PainterBar.
  or
  Select Rows➤Retrieve from the menu bar.

  PowerBuilder retrieves all the rows in the current table or view. As the rows are being retrieved, the button changes to Cancel. You can click it to stop the retrieval.

Modifying data

You can add, modify, or delete rows. When you are finished manipulating the data, you can apply the changes to the database.

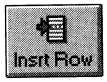
If looking at data from a view
You cannot update data in a view.

❖ To modify existing data:

- Tab to the field and enter a new value. The Data Manipulation painter uses validation rules, display formats, and edit styles that you have defined for the table in the Database painter.

  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 an empty row.

2 Enter data for a row.

To save the changes to the database, you must apply them, as described below.

❖ 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.

❖ To apply changes to the database:

- Click the Update Database button.

PowerBuilder updates the table with all the changes you have made.

Sorting and filtering data

You can define and use sort criteria and filters for the rows.

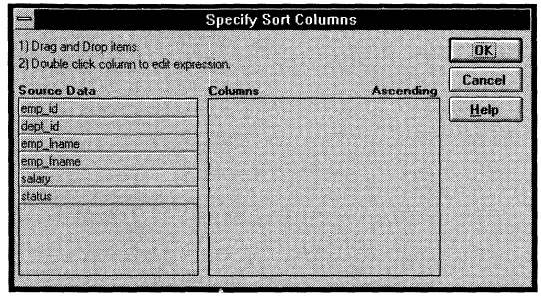
The sort criteria and filters you define in the Data Manipulation painter are for testing only and are not saved with the table or passed to the DataWindow painter.

Sorting rows

❖ To sort the rows:

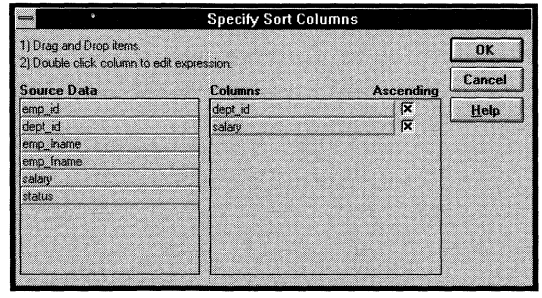
1 Select Rows➤Sort from the menu bar.

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 checkbox with an X in it displays under the Ascending heading to indicate that the values will be sorted in ascending order. To sort in descending order, uncheck the checkbox.



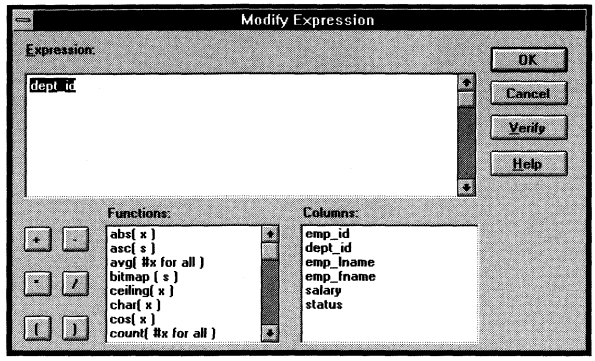
Precedence of sorting

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 will be sorted by dept ID. Within dept ID, rows will be sorted by salary.

To change the precedence order, drag the column names in the Column box to the order you want.

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 an item in the Columns box.

The Modify Expression dialog box displays.



4 Specify the expression and click OK.

You return to the Specify Sort Columns dialog. The expression is 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.

5 When you have specified all the sort columns and expressions, click OK.

PowerBuilder sorts the rows.

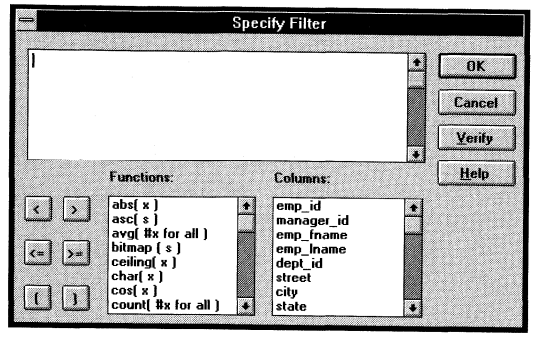
Filtering rows

You can limit which rows are displayed by defining a filter.

❖ 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 will be displayed. You can paste PowerScript 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.

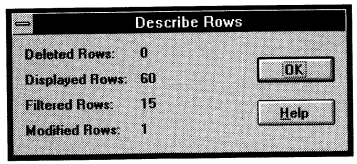
Viewing row information

You can display information about the data you have retrieved.

❖ To display the row information:

♦ Select Rows➤Described from the menu bar.

The Describe Rows dialog box displays.



The Describe Rows dialog box shows 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.

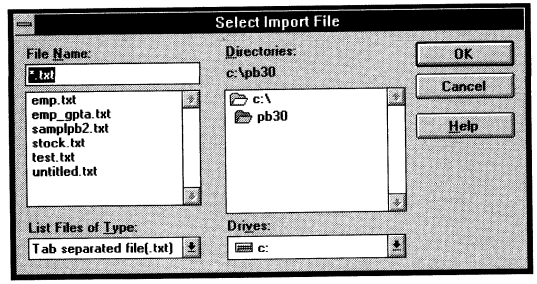
Importing data

You can import data from an external source and display it in the Data Manipulation painter, 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 that you can import into the painter are shown in the List Files of Type dropdown listbox.

3 Click OK.

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.

Printing data

You can print the data displayed in the Data Manipulation painter. 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. Rulers display around the page borders.

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.

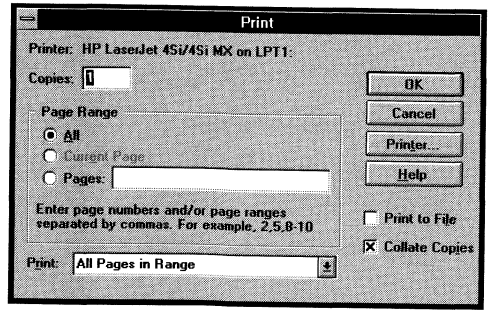
Preview zooms in or out as appropriate.

4 When you have finished looking at the print layout, select File➤Print Preview from the menu bar again.

❖ To print:

1 Select File➤Print from the menu bar.

The Print dialog box displays.



2 Fill in the dialog box as needed. Click Help to get more information about the dialog box.

PowerBuilder prints the report on the specified printer.

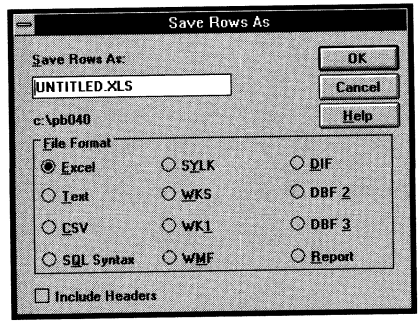
Saving data

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.

> **Saving the data as a report**
> You can save the data in a Powersoft report file (PSR file) by choosing the Report file format. You can display a PSR file in a DataWindow control using the following statement:
>
> *DataWindowControl*.DataObject = "*file*.psr"
>
> For more information about PSR files, see Chapter 14, "Enhancing DataWindow Objects."

3 Name the file.

4 If you want the columns' headings saved in the file, select the Include Headers box.

5 Click OK.

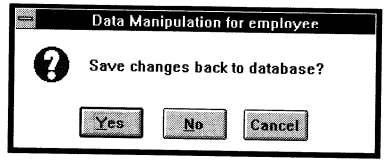
PowerBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Returning to the Database painter workspace

❖ To leave the Data Manipulation painter and return to the Database painter workspace:

1 Select File➤Close from the menu bar.

If you have made changes to the database but not yet saved them, PowerBuilder asks you whether you want to update the database.



2 If you have unsaved changes, respond to the prompt.

You return to the Database painter.

Administering the database

You can use the Database Administration painter to control access to the database and to create SQL for immediate execution.

Opening the Database Administration painter

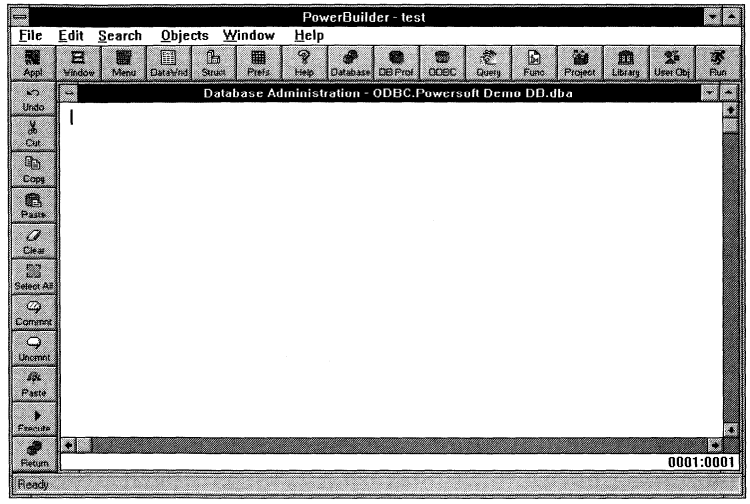
❖ To open the Database Administration painter:

♦ Click the Database Administration button in the PainterBar.
or
Select Objects➤Database Administration from the menu bar in the Database painter.

Adding Database Administration painter to PowerBar
You can also add the Database Administration button to the PowerBar. If you do this, you will be able to open the painter anytime you want—you don't need to go to the Database painter first.

For instructions, see "Customizing toolbars" in Chapter 1, "The World of PowerBuilder."

The Database Administration painter displays.



About the painter

The Database Administration painter is very much like the PowerScript painter. But instead of building scripts in it, you build SQL statements to send to the DBMS.

The painter provides the same editing capabilities as the PowerScript painter.

For more information, see Chapter 3, "Writing Scripts."

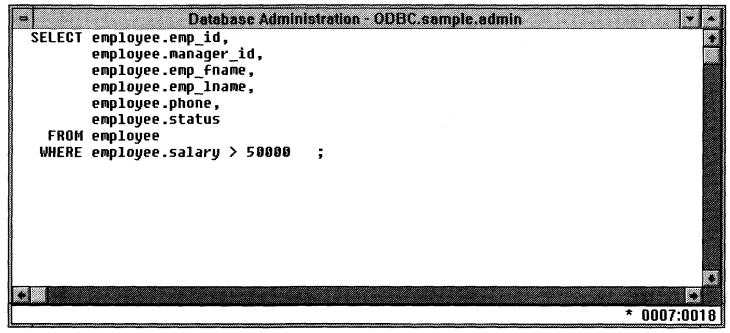
Controlling database access

The Database Administration painter provides a series of dialog boxes you can use to control access to the current database, such as by defining access to specific tables, creating and deleting users, and so on. The dialog boxes are tailored to your DBMS. They are all accessed from the Objects menu.

Executing SQL

You can use the Database Administration painter to build SQL statements and execute them immediately.

The painter's workspace acts as a notepad in which you can enter SQL statements. The workspace shown below contains a SQL SELECT statement:



About the statement terminator

By default, PowerBuilder uses the semicolon as the SQL statement terminator in the Database Administrator painter. You can override this by defining a different character in the [Database] section of PB.INI, as follows:

```
TerminatorCharacter=x
```

where *x* is any character.

About comments

By default, PowerBuilder strips off comments when it sends SQL to the DBMS. You can have comments included by deselecting Strip Comments on the Objects menu.

Entering SQL

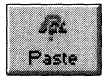
You can enter a SQL statement in three ways:

- Pasting the statement
- Typing the statement in the workspace
- Opening a text file containing the SQL

Pasting SQL

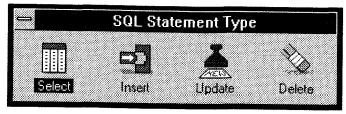
You can paste SELECT, INSERT, UPDATE, and DELETE statements to the workspace.

❖ To paste a SQL statement to the workspace:



1 Click the Paste SQL button in the PainterBar.
or
Select Edit➤Paste SQL from the menu bar.

The types of SQL statements you can paint display in a dialog box.



2 Double-click the appropriate icon to select the statement type.

The Select Table dialog box displays.

3 Select the tables 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 painting.

4 Follow the procedure described in the following table for the statement you are painting.

In each case, you can select Options➤Show SQL Syntax from the menu bar to see the SQL as you dynamically build it:

Type of statement	What you do
SELECT	Define the statement exactly as in the Select painter when building a view. You choose the columns to select. If you want, you can define computed columns, specify sorting and joining criteria, and WHERE, GROUP BY, and HAVING criteria. For more information, see "Working with views" on page 352.
INSERT	Type the values to insert into each column. You can insert as many rows as you want.
UPDATE	First 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.
DELETE	Specify the WHERE criteria to indicate which rows to delete.

5 When you have completed painting the SQL statement, click the Return button in the PainterBar in the Select, Insert, Update, or Delete painter.

You return to the Database Administration painter with the SQL statement pasted into the workspace.

Typing SQL

If you want, you can simply type one or more SQL statements directly in the workspace.

You can enter any statement supported by your DBMS. This includes statements you can paint as well as statements you cannot paint (such as a database stored procedure or CREATE TRIGGER statement).

Importing SQL from a text file

You can import SQL that has been saved in a text file into the Database Administration painter.

❖ To read SQL from a file:

1 Select File➤Import from the menu bar.

The File Import dialog box displays.

2 Select the file containing the SQL and click OK.

PowerBuilder inserts the SQL at the current insertion point.

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 Objects 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.

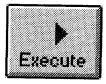
DBMS-specific information
The information displayed in the SQL Statement Execution Plan dialog box depends on your DBMS.

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 Objects➤Execute SQL from the menu bar.

  PowerBuilder submits the SQL in the workspace to the DBMS.

What happens next

If the SQL retrieves data, the data appears in a window identical to a grid Data Manipulation painter.

For a description of what you can do with the data, see "Manipulating data" on page 364.

If there is a database error, you see a message box describing the problem.

CHAPTER 13

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

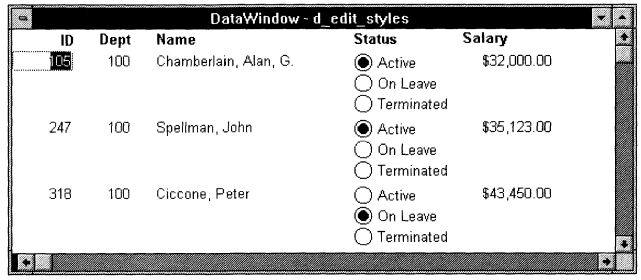
Overview

A DataWindow object is an object that you use to retrieve, present, and manipulate data from a relational database or other data source (such as an Excel worksheet or dBASE file).

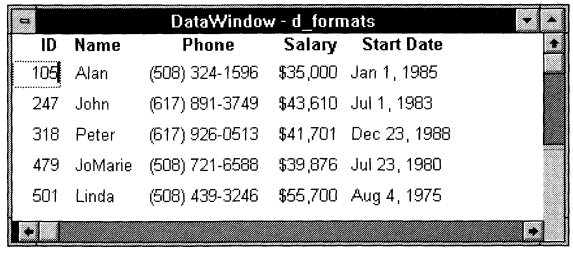
You can choose from several presentation styles. For example, you can display the data in a row-and-column format or in a freeform format.

DataWindow objects have knowledge about the data they are retrieving. You can specify display formats and other data attributes so the data is used in the most meaningful way by users. For example:

- If a column can take only a small number of values, you can have the data appear as radio buttons in a DataWindow so users know what their choices are.

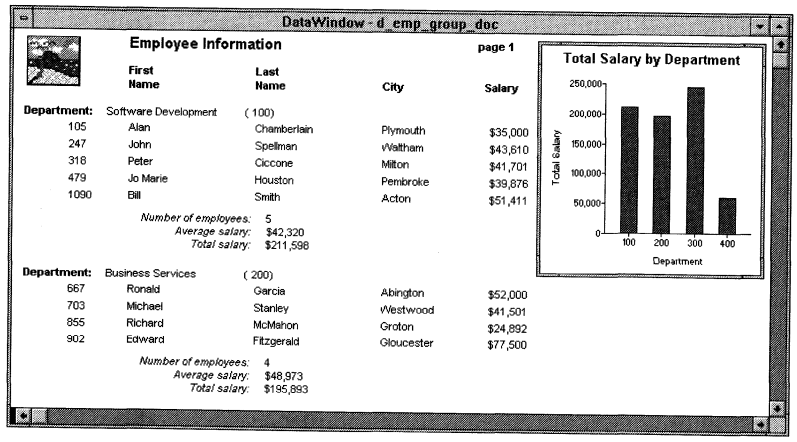


- You can format the display of data. For example, you can display phone numbers, salaries, and dates in formats appropriate to the data.



- If a column can take numbers only in a specific range, you can specify a simple validation rule for the data so you don't need to write any code to make sure users enter valid data.

There are many other ways you can enhance the presentation and manipulation of data in a DataWindow object as well. For example, you can include computed fields, pictures, and graphs that are tied directly to the data retrieved by the DataWindow.



How to use DataWindow objects

This section describes the overall process for creating and using DataWindow objects.

❖ To use DataWindow objects:

1 Create the DataWindow *object* in the DataWindow painter. In this painter, you define the data source, presentation style, and all other attributes of the object, such as display formats, validation rules, sorting and filtering criteria, and graphs.

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 function 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.

About reports and the Report painter

You build DataWindow objects in the DataWindow painter.

Included with PowerBuilder is a related painter: the Report painter, which is the same painter that is in InfoMaker. It is used to build reports. By default, the button that opens the Report painter is not in the PowerBar, but you can customize the PowerBar to add the button.

Reports and DataWindow objects are the same objects. When you create a report in the Report painter in InfoMaker or PowerBuilder, you are actually creating a DataWindow object that you can open and modify in the PowerBuilder DataWindow painter.

The only difference between a report and a DataWindow object is that reports are by definition not updatable; DataWindow objects can be updatable. That means that:

- The DataWindow painter has an Update item on the Rows menu. Here you specify which columns in the DataWindow are updatable. The Report painter does not have an Update item on its Rows menu.
- In the DataWindow painter you can specify validation rules for columns. You don't do that in the Report painter because users don't enter values in reports.
- When you preview a report, you go to Print Preview. When you preview an updatable DataWindow, you can manipulate the data.
- You can open and mail Powersoft report files (PSR files) in the Report painter, but not in the DataWindow painter.

Everything else is identical in the DataWindow painter and Report painter.

The DataWindow painter functions
The *Function Reference* describes all functions you can use in expressions in the DataWindow painter and the Report painter. Because DataWindow objects and reports are fundamentally the same object, the information applies to both.

The Report painter is provided with PowerBuilder as a convenience. You as a PowerBuilder developer need never use the Report painter—you can do all your work in the DataWindow painter (with the exception of opening and mailing PSR files). However, if you are creating a DataWindow object that is not updatable (that is, a *report*), you might want to use the Report painter so you won't have to make sure the object is specified as not updatable.

Two versions of the Report painter

There are actually two versions of the Report painter that come with PowerBuilder. You can see them in the Customize window where you add buttons to the PowerBar. The first Report painter button lets you design and run reports (that is, DataWindow objects that are not updatable). The second button lets you only run reports.

Building a DataWindow object

You use the DataWindow painter to work with DataWindow objects.

Connecting to a database

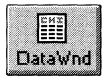
To use the DataWindow painter, you must be connected to the database whose data you will be accessing.

When you open the DataWindow painter, PowerBuilder connects you to the DBMS and database you used last (the information is recorded in the PB.INI file). If you need to connect to a different database, do so before working with a DataWindow object.

For information about changing your database connection, see *Connecting to Your Database*.

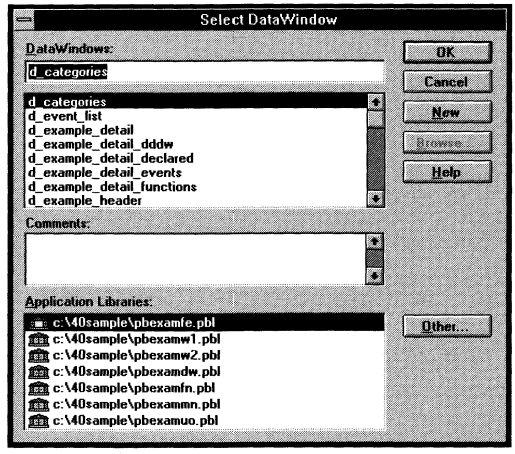
Modifying an existing DataWindow object

❖ To modify an existing DataWindow object:



1 Click the DataWindow button in the PowerBar or PowerPanel.

The Select DataWindow dialog box lists the DataWindow objects in the current library.



2 Select the existing DataWindow object and click OK.

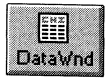
For more on selecting existing objects in this dialog box, see Chapter 1, "The World of PowerBuilder."

PowerBuilder opens the DataWindow painter and displays the DataWindow object in the workspace.

To learn how to modify an existing DataWindow object, see Chapter 14, "Enhancing DataWindow Objects."

Creating a new DataWindow object

❖ To create a new DataWindow object:

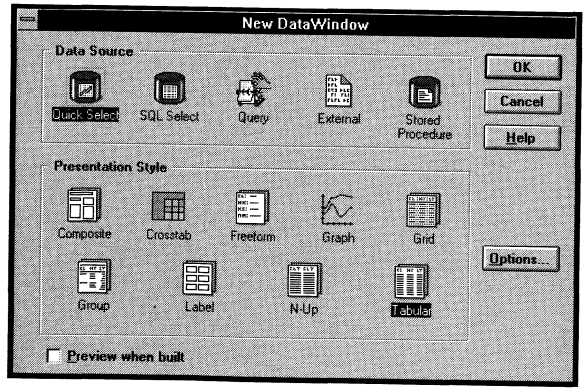


1 Click the DataWindow button in the PowerBar or PowerPanel.

The Select DataWindow dialog box lists the DataWindow objects in the current library.

2 Click the New button.

The New DataWindow dialog box displays.



3 Choose a presentation style—how you want the data to be arranged—for the DataWindow object.

For more information, see "Choosing a presentation style" on page 389.

4 (Optional) Choose options for the DataWindow object.

For more information, see "Choosing DataWindow-wide options" on page 396.

5 Choose a data source for the DataWindow object.

For more information, see "Defining the data source" on page 399.

6 If you want to immediately preview (run) the basic DataWindow object after defining the presentation style and data source, select the Preview When Built checkbox.

For more information on previewing, see Chapter 14, "Enhancing DataWindow Objects."

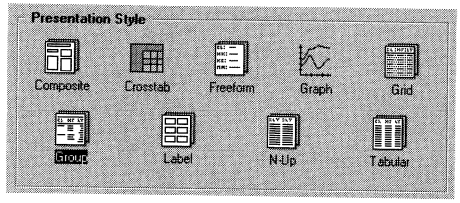
After choosing and defining the data source, you go to the painter's workspace. At this point, you can save the DataWindow object and begin using it in a window. However, you may want to enhance it first.

7 (Optional) Enhance your DataWindow object using the painter's workspace. For example, you can customize the display of column data and add objects such as text, bitmaps, computed fields, and graphs.

You can also preview the DataWindow object at any time in the DataWindow painter.

Choosing a presentation style

The presentation style you select for a DataWindow object determines the format PowerBuilder uses to first display the DataWindow object in the DataWindow painter workspace. You can use the format as displayed or modify it to meet your needs.

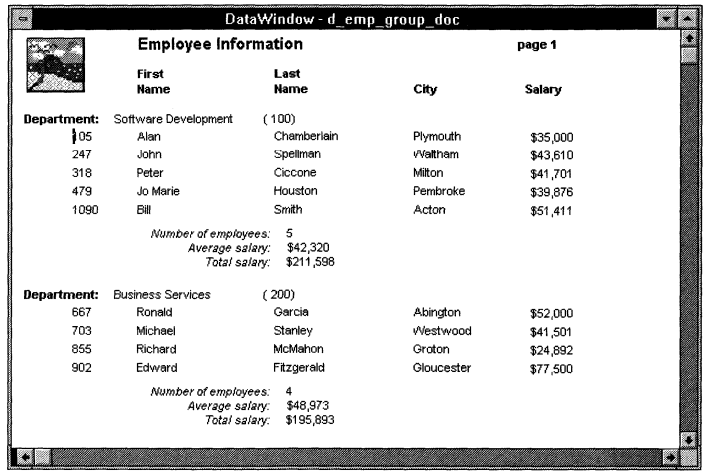


You can choose from the following presentation styles in the New DataWindow dialog box:

- Tabular
- Freeform
- Grid
- Label
- N-Up
- Group
- Composite
- Graph
- Crosstab

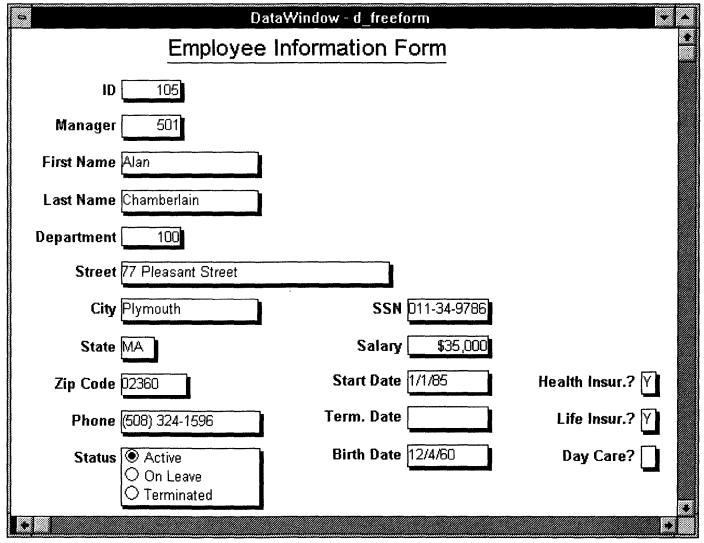
Using the Tabular style

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. Tabular style is often used when you want to group data.



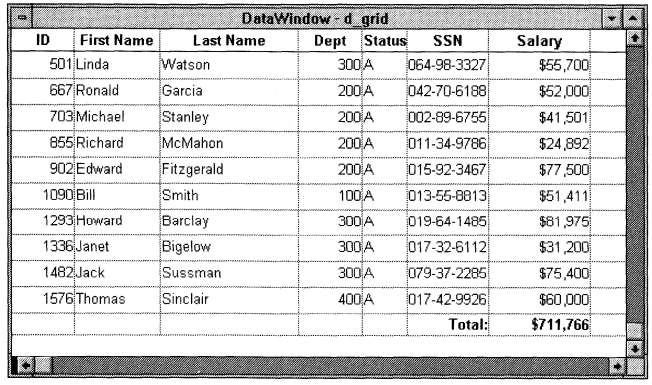
Using the Freeform style

The Freeform 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. Freeform style is often used for data entry forms.



Using the Grid style

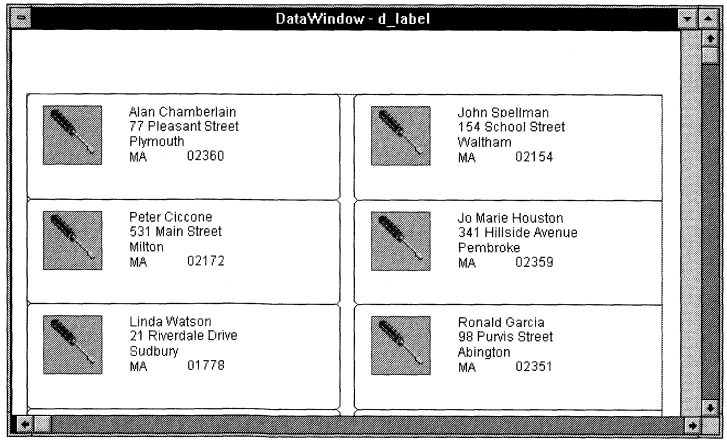
The Grid style presents data in a row-and-column format with grid lines separating rows and columns. Everything you place in a grid DataWindow object must fit in one cell in the grid. You cannot move columns and headings around as you can in the tabular and freeform styles. But, unlike the other styles, users can reorder and resize columns with the mouse in a grid DataWindow object during execution.



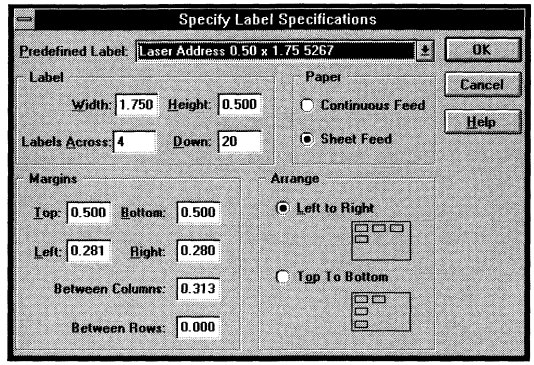
ID	First Name	Last Name	Dept	Status	SSN	Salary
501	Linda	Watson	300	A	064-98-3327	$55,700
667	Ronald	Garcia	200	A	042-70-6188	$52,000
703	Michael	Stanley	200	A	002-89-6755	$41,501
855	Richard	McMahon	200	A	011-34-9786	$24,892
902	Edward	Fitzgerald	200	A	015-92-3467	$77,500
1090	Bill	Smith	100	A	013-55-8813	$51,411
1293	Howard	Barclay	300	A	019-64-1485	$81,975
1336	Janet	Bigelow	300	A	017-32-6112	$31,200
1482	Jack	Sussman	300	A	079-37-2285	$75,400
1576	Thomas	Sinclair	400	A	017-42-9926	$60,000
					Total:	$711,766

Using the Label style

The Label style presents data as labels. Choose this style to create mailing labels or other kinds of labels.



If you choose the Label style, after specifying the data source, you are asked to specify the attributes for the label in the Specify Label Specifications dialog box:



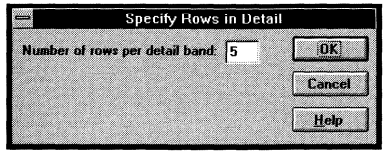
You can choose from a list of predefined label types in the Predefined Label box or enter your own specifications manually. For more information on the dialog box, click the Help button.

Where the label definitions come from
PowerBuilder gets the information about the predefined label formats from PBLAB040.INI in the PowerBuilder directory.

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, but 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 each period repeats in a row.

After you select a data source, you are asked how many rows to display across the page.



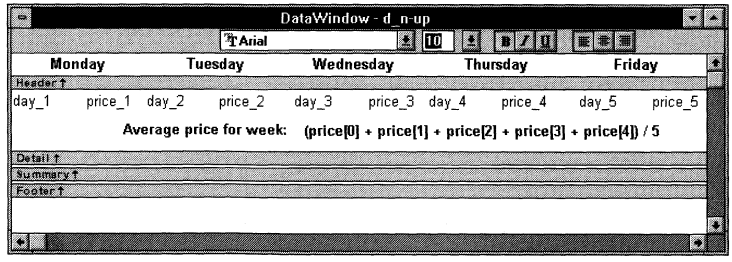
For each column in the data source, PowerBuilder defines *n* columns in the DataWindow (*column_1* to *column_n*), where *n* is the number you specified in the preceding dialog box.

Example

For example, if you have a table of daily stock prices, you can define the DataWindow as five across, so each row in the DataWindow displays five days' prices (Monday through Friday). Here is a table with two columns that record the closing stock price each day for three weeks (this table is not in the Powersoft Demo Database).

Day	Price	Day	Price
1/31/94	62.00	2/10/94	68.00
2/1/94	63.00	2/11/94	67.00
2/2/94	66.00	2/14/94	67.00
2/3/94	65.00	2/15/94	71.00
2/4/94	62.00	2/16/94	70.00
2/7/94	65.00	2/17/94	72.00
2/8/94	69.00	2/18/94	75.00
2/9/94	66.00		

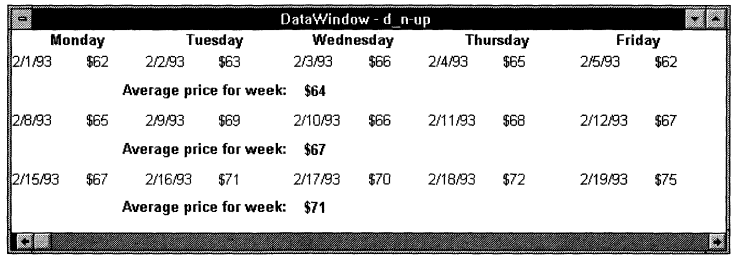
In the following n-up DataWindow object, 5 was selected as the number of rows to display across the page, so each line in the DataWindow shows five days' stock prices. A computed field was added to get the average closing price in the week.



About computed fields in n-up DataWindows
You use subscripts, such as price**[0]**, to refer to particular rows in the detail band in n-up DataWindows.

For more information, see Chapter 14, "Enhancing DataWindow Objects."

Here is the DataWindow in preview:



Notice that each line in the DataWindow object contains five rows from the database.

Another way to get multiple-column DataWindows
In an n-up DataWindow, the data goes across then down, which works fine for periodic data. But it you want your data to go down the page then across in multiple columns, such as in a phone list, you should create a standard tabular DataWindow, then specify newspaper columns.

For more on newspaper columns, see the chapter on using DataWindow objects in *Building Applications*.

Using the Group presentation 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 that has grouping properties defined for you.

For more about the Group presentation style, see Chapter 16, "Filtering, Sorting, and Grouping Rows."

Using the Composite presentation style

The Composite presentation style allows you to combine multiple DataWindows in the same object. It is particularly handy if you want to print more than one DataWindow on a page.

For more about the Composite presentation style, see Chapter 17, "Using Nested Reports."

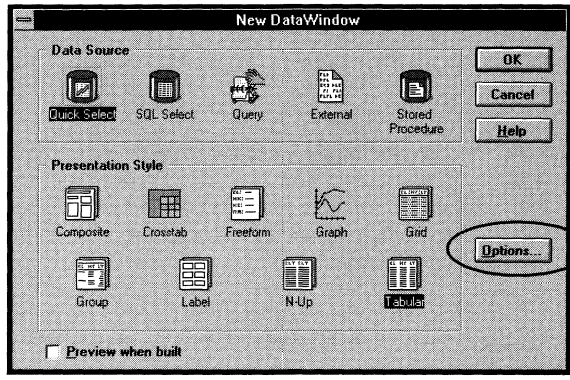
Using the Graph and Crosstab presentation styles

In addition to the preceding presentation styles, which are text-based, PowerBuilder provides two styles that allow you to display information graphically: Graph and Crosstab.

For more information about these two presentation styles, see Chapter 18, "Working with Graphs," and Chapter 19, "Working with Crosstabs."

Choosing DataWindow-wide options

When creating a new DataWindow object, you can specify default colors and borders by clicking the Options button after selecting a presentation style.



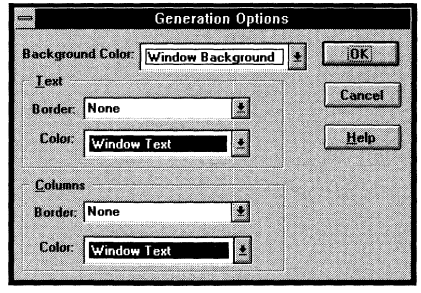
Click here to set default properties for DataWindow objects

The colors and borders you choose here become the defaults in new DataWindow objects using the currently selected presentation style. You can have different default settings for tabular, freeform, grid, group, n-up, and label DataWindow objects (you don't set defaults for composite, graph, and crosstab DataWindow objects).

❖ To specify default colors and borders:

1 Click the Options button in the New DataWindow dialog box.

The Generation Options dialog box displays.



2 Change one or more of the following attributes:

Attribute	Meaning
Background color	The default color for the entire DataWindow object. You can override this setting for specific parts of the DataWindow object later.
Text border and color	The default border and colors used for labels and headings in the DataWindow object.
Column border and color	The default border and color used for the retrieved data in the DataWindow object.

3 Click OK.

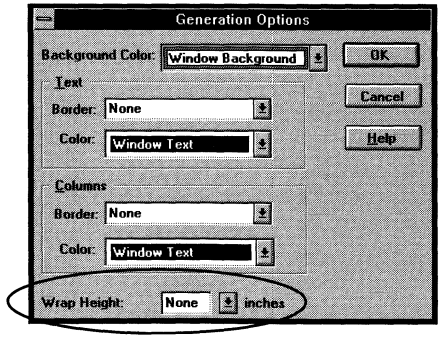
You return to the New DataWindow dialog box.

About the colors

For each DataWindow component, you can select Window Background, AppWorkSpace, or Window Text. If you choose one of these three colors for a component, your DataWindow object will use the color that the particular user has specified in the Windows Control Panel for the corresponding window element.

In freeform DataWindow objects

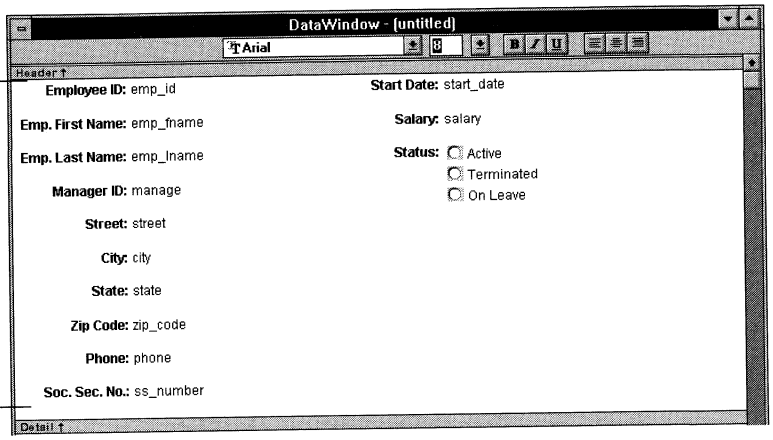
If you are defining generation options for freeform DataWindows, there is an additional choice in the dialog box: Wrap Height.



The value you specify for wrap height determines the default height of the detail band in the generated DataWindow.

You can use this setting to easily create multiple columns of fields in the freeform DataWindow. For example, if you specify three inches as the wrap height, PowerBuilder puts labels and database columns down the left side of the DataWindow until it reaches three inches, then continues putting labels and database columns in a second column in the DataWindow, and so on.

The following shows a generated freeform DataWindow after the default wrap height was set to three inches. PowerBuilder filled out the three inches, then put the Start Date, Salary, and Status columns to the right.

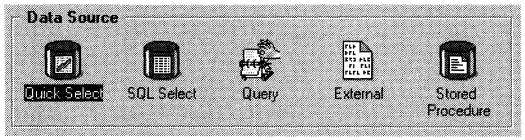


Your choices are saved

PowerBuilder saves the choices you make for generation options in the [Data window] section of the PB.INI file, so the next time you create a DataWindow object with the same presentation style, your new choices will be the defaults.

Defining the data source

The data source you choose here 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 DataWindow painter to specify the data to retrieve into the DataWindow object.

Data source can also refer to where the data comes from, such as a *Watcom SQL data source* (meaning a database file) or a *dBASE data source* (meaning a DBF file). The manual *Connecting to Your Database* uses the term *data source* in this second way.

You can choose from the following data sources when first defining your DataWindow object in the New DataWindow dialog box:

- Quick Select
- SQL Select
- Query
- External
- Stored Procedure

About stored procedures

The Stored Procedure data source icon displays only if the DBMS you are currently connected to supports stored procedures that return result sets.

How to choose the data source

If the data is in a database

If the data for the DataWindow object will be retrieved from a database, choose one of the following data sources:

Use this data source	If
Quick Select	The data is coming from one table or from multiple tables *that are related through foreign keys* and you only need to choose columns, selection criteria, and sorting (you don't need to specify grouping, computed columns, and so on)
SQL Select	You want more control over the SELECT statement that is generated for the data source
Query	The data has already been defined and saved in a query
Stored procedure	The data is defined in a database stored procedure

If the database is not in a database

If the data is not coming from a database, select External as the data source. This includes the following situations:

- If the DataWindow object will be populated from a script
- If data will be imported from an external file, such as a tab-separated text file (TXT file) or a dBASE file (DBF file).

> **If data is in a text or dBASE file**
> Another way to get text or dBASE data in a DataWindow is to use an ODBC driver. PowerBuilder ships ODBC drivers for both text and dBASE files.
>
> For more information, see *Connecting to Your Database*.

- If data will be imported from a DDE application

Why use a DataWindow object if data not coming from DBMS

It is not obvious why you might want to use a DataWindow object if the data is not coming from the database. But there are many times when you want to take advantage of the intelligence of a DataWindow object, even if the data is not coming from a database.

Consider this situation: your application uses a Login window, which reads or writes data to or from an external file that lists users and their privileges. It is true that you can do that without using a DataWindow object, by using file-related PowerScript functions. But then you have to manually code any validation or editing procedures you need to perform on the data.

Instead, do that kind of processing in a script-based DataWindow object so you can take advantage of the following features provided automatically by PowerBuilder:

- Data validation—because you are using a DataWindow object, 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.

Once you choose a data source in the New DataWindow dialog box and click OK, you specify the data. The data source you choose determines which dialog box displays and how you define the data.

Using Quick Select

The easiest way to define a data source is using Quick Select. With Quick Select, you can choose columns from one table or from multiple tables if they are joined through foreign keys. After you choose the columns, you can:

- Specify whether you want to sort the retrieved rows
- Specify criteria that a row must meet in order to be retrieved

What you can't do using Quick Select

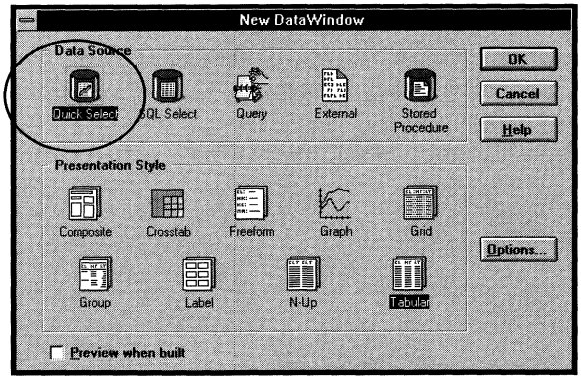
When you choose Quick Select as your data source, you can't:

- Specify grouping before rows are retrieved
- Include computed columns
- Specify retrieval arguments

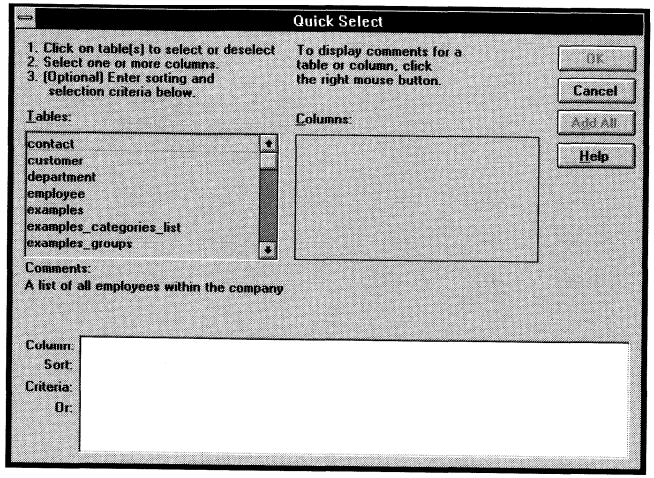
To use these options, choose SQL Select as your data source.

❖ To define the data source using Quick Select:

1 Click Quick Select in the New DataWindow dialog box, click a presentation style, then click OK.

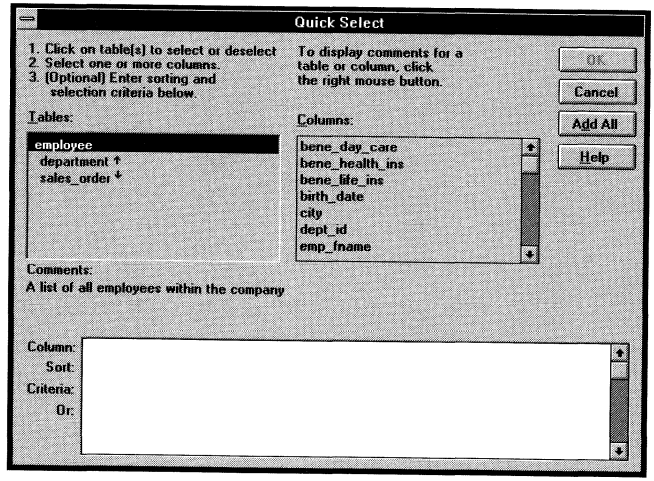


The Quick Select dialog box displays. All tables and views to which you have access in the current database are listed in the Tables box. To display a comment about a table, point at the table and click the right mouse button.



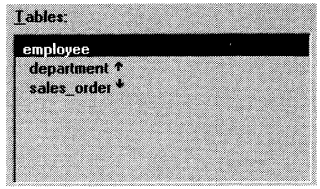
2 Select a table containing the data you want to use.

The table's column names display in the Columns box, and other tables having a key relationship with the selected table display in the Tables box. These tables are indented and marked with an arrow to indicate their relationship to the selected table.



Meaning of the up and down arrows

An arrow displays next to a table that has a key relationship with the selected table. The arrow points up if the table contains a primary key that points to a foreign key in the selected table. The arrow points down if the table contains a foreign key that is pointed to by the primary key of the selected table. For example,



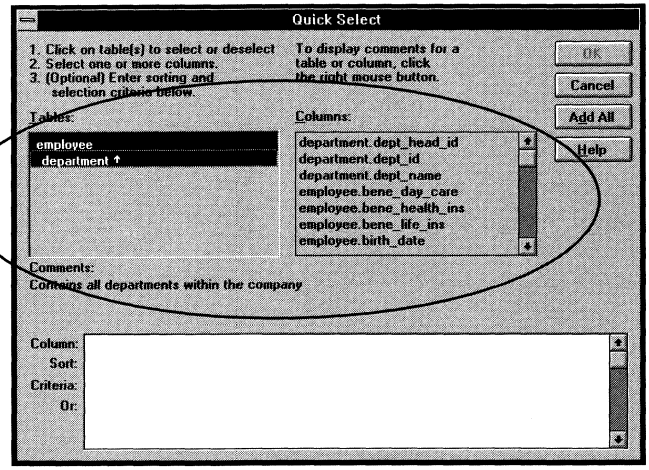
indicates that the Department table has a primary key that points to a foreign key in the Employee table (on the Dept_ID column) and that the Sales_Order table has a foreign key that is pointed to by the primary key in the Employee table (the Emp_ID column).

3 Select any additional tables containing data you want to use.

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.birth_date display when the Employee table and the Department table are selected.

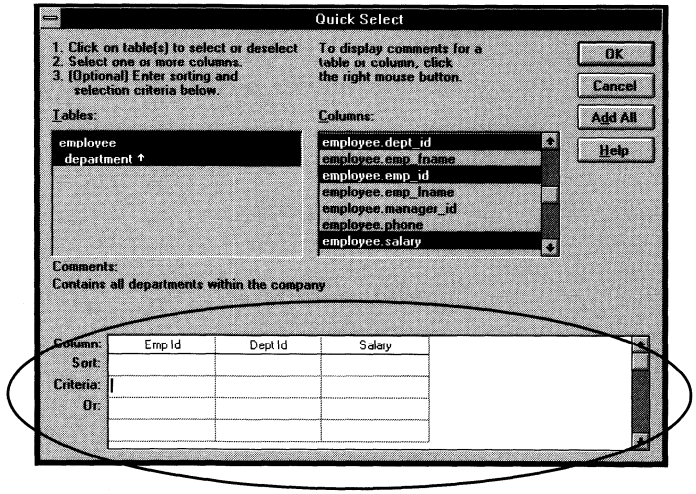


To return to the original table list
Click the table you first selected at the top of the table list.

4 Select the columns you want to use:

- To add columns, select them in the Columns box.
- To add all columns, click Add All.
- To remove columns, deselect them in the Columns box.
- To view comments that describe a table or column, press and hold the right mouse button in a table name or column name.

The selected columns display in the grid at the bottom of the dialog box.



5 Work in the grid to finalize column ordering, sort rows before you retrieve data, and specify what data to retrieve:

- To reorder a column, move the pointer to the column name, press the left mouse button, and drag the column to a new position.
- To specify the sorting of rows before retrieval, enter the sorting criteria in the grid.

For information, see "Specifying sorting criteria" on page 407.

- To specify what data to retrieve, enter expressions in the grid.

  For information, see "Specifying selection criteria" on page 408.

6 Click OK.

The Quick Select dialog box closes.

What happens

You go to the DataWindow painter workspace, with the following two exceptions:

- If you selected the Preview When Built checkbox in the New DataWindow dialog box, you go to Preview mode first.

  For more information, see Chapter 14, "Enhancing DataWindow Objects."

- If you are building a label, n-up, group, graph, or crosstab DataWindow, you need to provide additional information.

For information about	See
Label	Page 391
N-up	Page 392
Group	Chapter 16, "Filtering, Sorting, and Grouping Rows"
Graph	Chapter 18, "Working with Graphs"
Crosstab	Chapter 19, "Working with Crosstabs"

Using Quick Select

When you use Quick Select to define the data, you cannot define retrieval arguments for the SELECT statement that are supplied during execution. If you decide later that you want to use retrieval arguments, you can define them from the DataWindow painter workspace by modifying the data source.

For more information, see Chapter 14, "Enhancing DataWindow Objects."

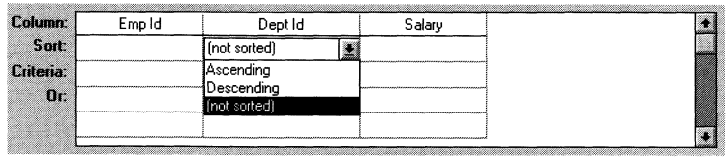
Specifying sorting criteria

In the grid at the bottom of the Quick Select dialog box, you can specify whether 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 dropdown listbox.

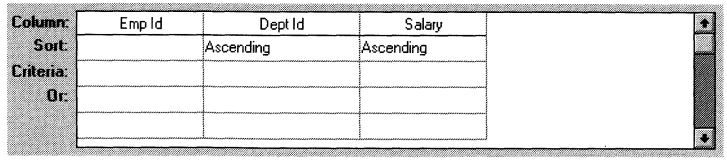


2 If you want to sort rows in ascending order, select Ascending. If you want to sort rows in descending order, select Descending.

Doing multilevel sorts

You can specify as many columns for sorting as you want. PowerBuilder processes the sorting criteria left-to-right in the grid. That is, 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.

In the following example, rows will be sorted first by dept ID. Within dept ID, rows will be sorted by salary:



If you want to do a multilevel sort that doesn't match the column order in the grid, you can reorder the columns as described on page 405.

Specifying selection criteria

You can specify selection criteria in the grid to determine 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 and Marketing who make more than $50,000.

As you specify selection criteria, PowerBuilder builds a WHERE clause for the SELECT statement.

❖ To specify selection criteria:

- Enter the criteria as expressions in the grid below the column names starting in the row below the Sort row. If the column is too narrow for the criteria, drag the grid line to enlarge the column.

> **About edit styles**
> If a column has an edit style associated with it in the repository (that is, the association was made in the Database painter), the edit style is used in the grid—except that dropdown listboxes are used for columns with code tables and columns using the CheckBox and RadioButton edit styles.

Supported operators

You can use these SQL relational operators in the criteria:

Operator	Meaning
=	Is equal to (default operator)
>	Is greater than
<	Is less than
< >	Does not equal
> =	Is greater than or equal to
< =	Is less than or equal to
LIKE	Matches this pattern
IN	Is 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*.

You can use the OR and AND logical operators to connect expressions, as described below.

The LIKE operator Use LIKE to search for strings that match a predetermined pattern. When you use LIKE, you can use the percent sign (%) wildcard or the underscore character (_) to match unknown characters in a pattern. The percent sign, like the DOS wildcard asterisk (*), matches multiple characters; the underscore matches a single character. For example, Good% matches all names that begin with Good, and Good _ _ _ matches all seven-letter names that begin with Good.

The IN operator Use IN to compare a value to 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 more than one column in a line

When you specify criteria for more than one column in a line in the grid, PowerBuilder assumes a logical AND between the criteria. A row from the database is retrieved if *all* criteria in the line are met.

Using more than one line

When you specify two or more lines of selection criteria, PowerBuilder assumes a logical OR. A row from the database is retrieved if the criteria in *any* of the lines is met.

Using AND and OR in an expression

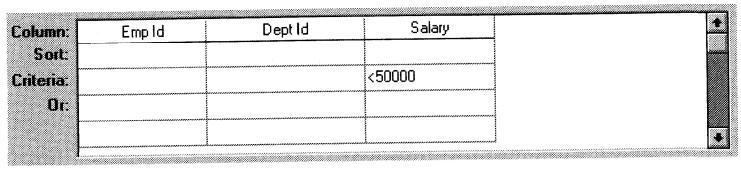
As just described, by default, criteria expressions in one line are logically ANDed; expressions in different lines are logically ORed. You can begin an expression with the AND or OR operator to override these defaults.

Operator	Meaning
OR	When one expression OR another expression is true, the row is selected.
AND	When one expression AND another expression are true, the row is selected.

This technique is particularly handy when you want to retrieve a *range* of values in a column. See Example 6 below.

Examples

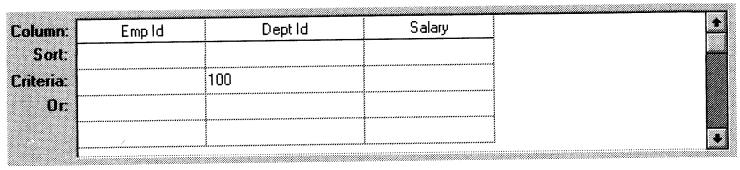
Example 1 Enter the following expression in the grid to retrieve information for employees whose salaries are less than $50,000.



The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE salary < 50000
```

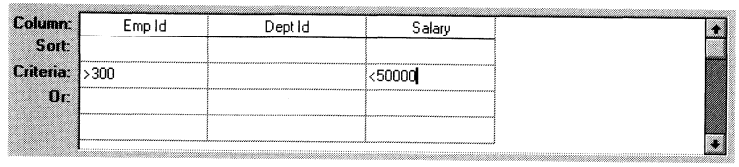
Example 2 Enter the following expression in the grid to retrieve information for employees who belong to department 100. Note that you do not need to use the = operator, since = is the default operator in an expression.



The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id = 100
```

Example 3 Enter the following expressions in the grid to retrieve information for employees whose employee ID is greater than 300 *and* whose salary is less than $50,000.

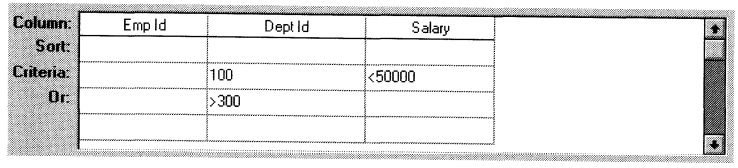


The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE emp_id >300 AND salary <50000
```

Example 4 Enter the following expressions in the grid to retrieve information for employees who either:

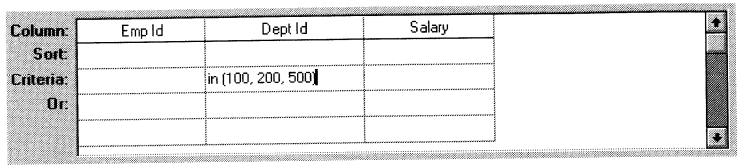
- Belong to department 100 *and* have a salary less than $50,000
- Or belong to a department whose ID is greater than 300, no matter what their salary



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
```

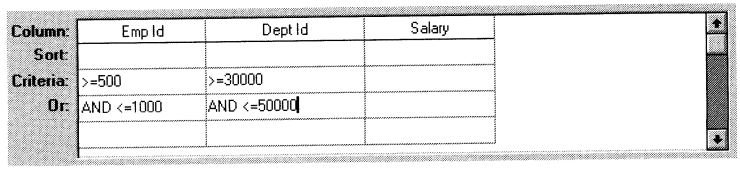
Example 5 Enter the following expressions in the grid to retrieve information for employees who are in department 100 *or* 200 *or* 500:



The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id IN (100, 200, 500)
```

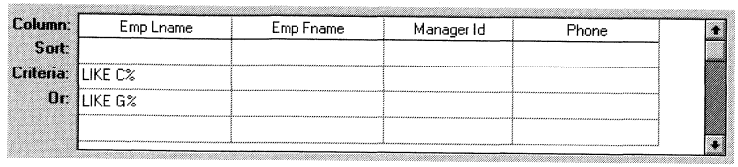
Example 6 Enter the following expressions in the grid to retrieve information for employees who have an employee ID from 500 to 1000 and a salary from $30,000 to $50,000:



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 Enter the following expressions in the grid to retrieve information for employees who have last names that begin with C or G:



The SELECT statement that PowerBuilder creates is:

```
SELECT emp_lname, emp_fname, manager_id, phone
FROM employee
WHERE emp_lname LIKE 'C%'
OR emp_lname LIKE 'G%'
```

Providing this functionality to users

You can allow your users to specify selection criteria using these same techniques in a DataWindow during execution:

- You can automatically pop up a window prompting users to specify criteria each time just before data is retrieved.

  For more information, see Chapter 14, "Enhancing DataWindow Objects."

- You can place the DataWindow in query mode using the Modify function.

  For more information, see *Building Applications*.

Using SQL Select

When you choose SQL Select as the data source, 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 during execution

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 442.

❖ To define the data using SQL Select:

1 Click SQL Select in the New DataWindow dialog box and click OK.

The Select Tables dialog box displays.

2 Select the tables and/or views that you will use in the DataWindow object (page 415).

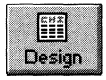
3 Select the columns to be retrieved from the database (page 418).

4 Join the tables if you have selected more than one (page 422).

5 Select retrieval arguments if appropriate (page 424).

6 Specify WHERE, ORDER BY, GROUP BY, and HAVING criteria if appropriate (page 426)

7 If you want to eliminate duplicate rows, select Distinct from the Options menu. This adds the DISTINCT keyword to the SELECT statement.

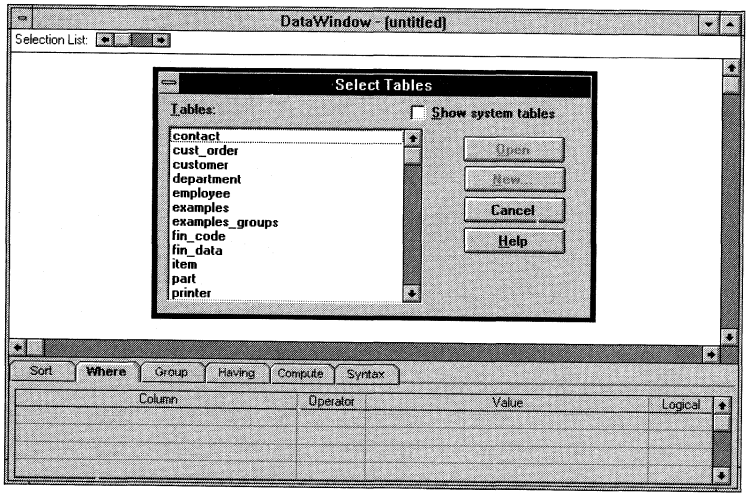


8 When you have finished defining the data source, click the Design button in the PainterBar.

The DataWindow painter workspace displays (unless you need to provide more information, in which case you are prompted for it).

Selecting tables and views

After you click SQL Select in the New DataWindow dialog box, choose a presentation style, and click OK, the Select Tables dialog box displays at the top of the workspace.



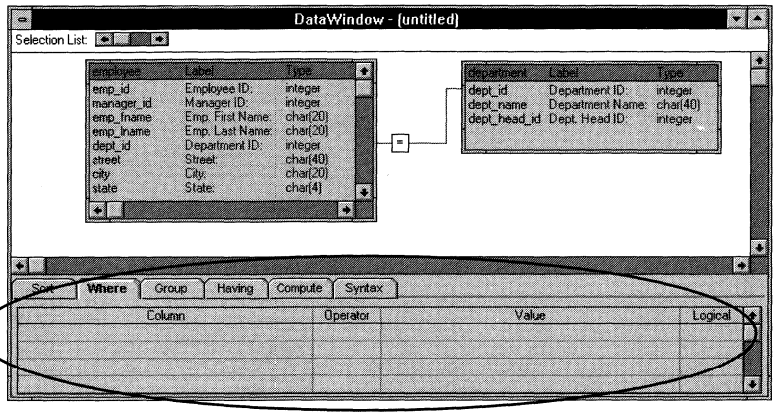
All tables and views to which you have access in the current database are listed. The SQL toolbox displays at the bottom of the workspace; you will use the tabs in the toolbox to specify your SQL SELECT statement.

❖ 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. When your selection is complete, click the Open button to open them.
- Double-click the name of each table or view you want to open. Each object opens immediately in the workspace behind the Select Tables dialog box. Then click the Cancel button to close the Select Tables dialog box.

Representations of the selected tables and views display in the Select painter workspace. You can move or size each table to fit the workspace as needed.

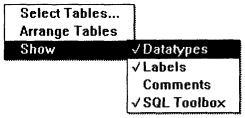


SQL toolbox with tabs for you to specify the SELECT statement

Specifying what is displayed

By default, tables in the workspace display the label and data type of each column (the label information comes from the Powersoft repository). You can choose to hide this information as well as to display comments that have been defined for the columns in the repository. You can also choose to hide the SQL toolbox to give you more room to see the tables.

❖ To hide or display comments, data types, labels, and the SQL toolbox:



1 Point to any unused area of the workspace and select Show from the popup menu.

   A cascading menu displays.

2 Select or deselect Datatypes, Labels, Comments, or SQL Toolbox as needed.

Shortcut
You can also click the Toolbox button in the PainterBar to toggle the display of the SQL toolbox.

Changing the colors of the tables

You can change the colors used by the Select painter to display table information by assigning values to variables in the [Database] section of your PB.INI file, as follows:

Variable = red green blue

where *red*, *green*, and *blue* are numbers from 0 to 255 that specify the amount of each primary color.

You can set colors separately for each component in the table representations as follows:

Variable	Applies to
TableHeaderColor	Background of the header (default: dark gray, or **128 128 128**)
TableHeaderTextColor	Text in the header, used for column names and other information (default: black, or **0 0 0**)
TableDetailColor	Background of the detail area, which shows column names and other information (default: light gray, or **192 192 192**)
TableColumnNameTextColor	Column names in the detail area (default: black, or **0 0 0**)
TableDetailTextColor	Other information (such as comments) in the detail area (default: blue, or **0 0 128**)

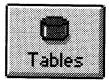
For example, the following statement displays column names in red:

```
TableColumnNameTextColor=255 0 0
```

Adding and removing tables and views

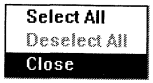
You can add more tables and views to your workspace at any time.

❖ To add more tables and views to the workspace:



- Click the Tables button in the PainterBar and select additional tables or views.

❖ To remove a table or view from the workspace:



- Point at the table or view name and select Close from the popup menu.

❖ To remove all tables and views so you can select tables again:

- Select Options➤Undo All from the menu bar.

  This removes all tables and views from the workspace and displays the Select Tables dialog box.

If more than one table is open

If you select more than one table in the Select painter workspace, PowerBuilder joins columns in the following way:

- If the tables have a primary/foreign key relationship, PowerBuilder automatically joins them.
- If the tables have no key relationship, PowerBuilder makes its best guess and tries to join tables based on common column names and types.

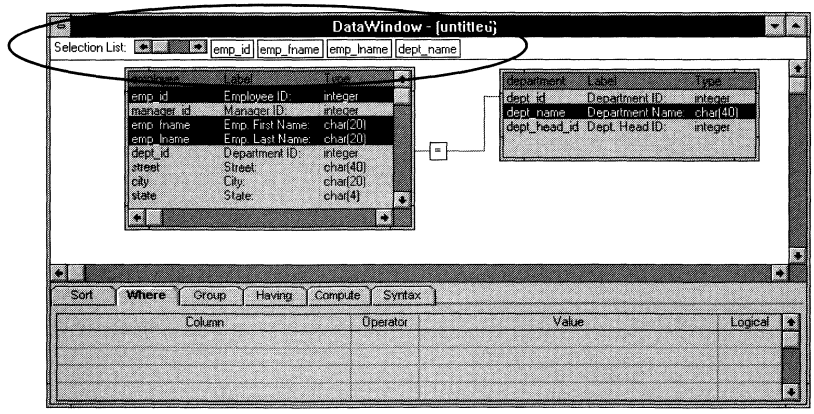
Defining or redefining a join

You may need to define a join or redefine PowerBuilder's best-guess join, which could be incorrect. PowerBuilder's best guesses for selected tables can differ depending on the order in which you select the tables.

For information about joins, see "Joining tables" on page 422.

Selecting columns

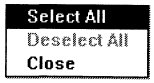
Click the columns you want to include from the table representations in the workspace. PowerBuilder highlights selected columns and places them in the Selection List along the top of the Select painter.



❖ 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.

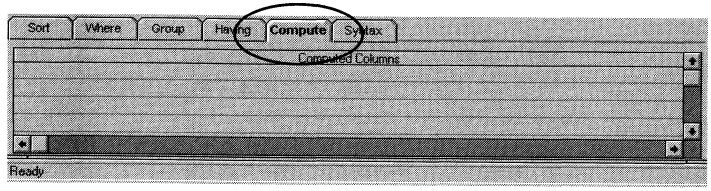
❖ To select all columns from a table:



- Select Select All from the table's popup menu.

❖ To include computed columns:

1 Click the Compute tab in the SQL toolbox at the bottom of the workspace.



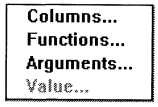
Each row in the Compute tab is a place for entering an expression that defines a computed column.

2 Enter an expression for the computed column, for example:

```
salary / 12
```

or a function supported by your DBMS (the following is a Watcom SQL function):

```
substr(emp_fname,1,5)
```



You can use the popup menu to paste the following into the expression:

- Names of columns in the tables used in the DataWindow
- Functions supported by the DBMS

> **About these functions**
> The functions listed by PowerBuilder here are functions *that are provided by your DBMS.* They are not PowerScript functions. (This is because you are now defining a SELECT statement that will be sent to your DBMS for processing.)

- Any retrieval arguments you have specified

3 Click the next row to define another computed column, click another tab to make additional specifications, or click the Design button to return to the workspace.

PowerBuilder adds the computed columns to the list of columns you have selected.

About defining computed columns here

Computed columns you define in the Select painter are added to the SQL statement and used by the DBMS to retrieve the data. So 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 (though if you have many rows, this updating can result in slower performance).

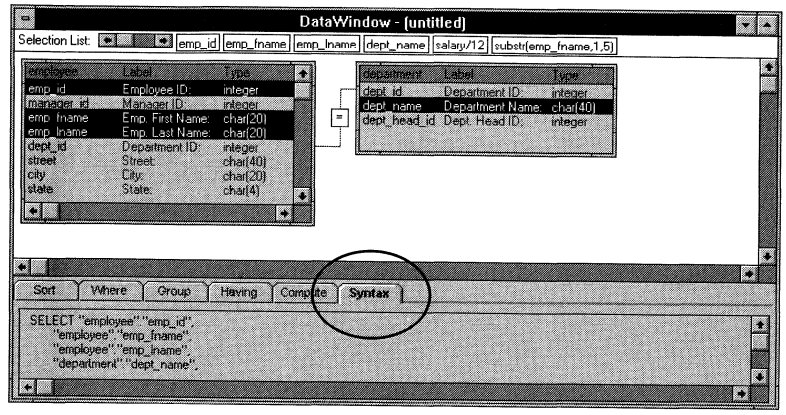
For more information, see Chapter 14, "Enhancing DataWindow Objects."

Displaying the underlying SQL statement

As you specify the data for the DataWindow object in the Select painter, PowerBuilder is generating a SQL SELECT statement. It is this SQL statement that will be sent to the DBMS when you retrieve data into the DataWindow. You can look at the SQL as it is being generated.

❖ To display the SQL statement:

- Click the Syntax tab in the bottom of the workspace.



In the Syntax tab, PowerBuilder displays the SQL SELECT statement it is generating. You can click the Syntax tab at any time as you continue defining the data for the report or form. The SQL SELECT statement is updated each time you make a change. You may need to use the scroll bar to see all parts of the statement.

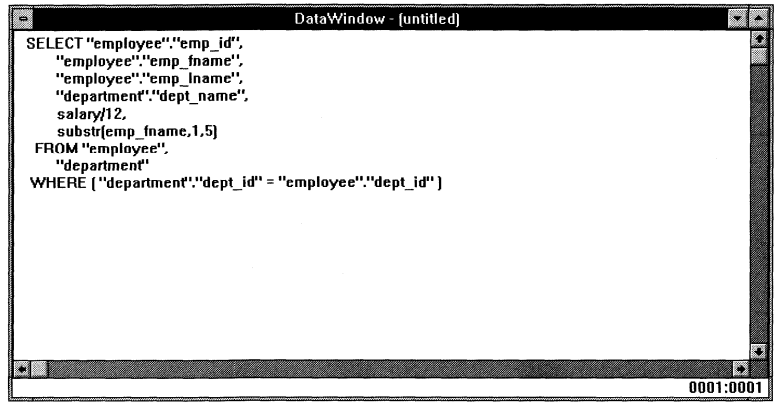
Editing the SELECT statement syntactically

Instead of modifying the data source graphically, you can directly edit the SELECT statement.

❖ To specify the data source syntactically:

1. When in the Select painter, select Options➤Convert to Syntax from the menu bar.

   PowerBuilder displays the SELECT statement in a text window.



2 Edit the SELECT statement. You can use CTRL+X to cut text to the clipboard, CTRL+C to copy text to clipboard, CTRL+V to paste text from the clipboard, and CTRL+Z to undo the most recent change.

3 Do one of the following:

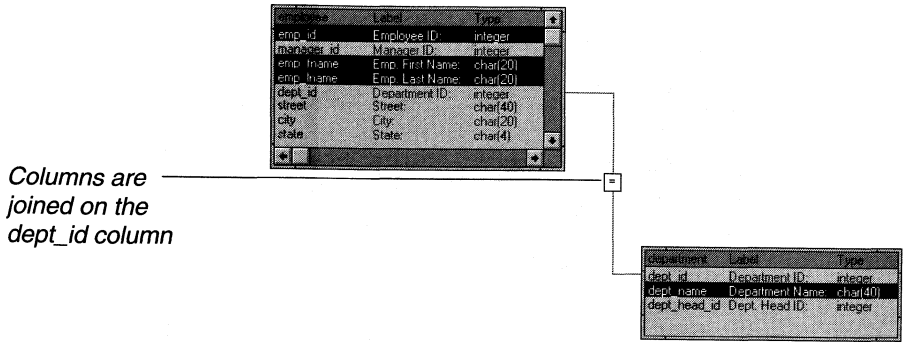
- Select Options➤Convert to Graphics from the menu bar to return to the Select painter.
- Click the Design button to go to the DataWindow painter workspace.

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 makes its best guess about 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.

PowerBuilder links joined tables in the Select painter workspace.



Sometimes PowerBuilder's best-guess join is incorrect (and PowerBuilder's joins can differ depending on the order in which you select the tables). So you may need to delete a join and manually define a join.

❖ To delete a join:

1 Double-click the join operator connecting the tables.

The Join dialog box displays.

2 Click Delete.

❖ To join tables:

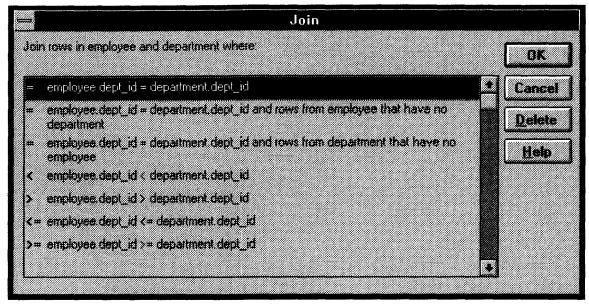
1 Click the Join button.
or
Select Objects➤Joins from the menu bar.

When the pointer is positioned over table columns, it changes to the Join symbol.

2 Click the columns on which you want to join the tables.

3 To create a join other than the normal equality join, click the join operator 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 your DBMS documentation.

Using retrieval arguments

If you know which rows will be retrieved into the DataWindow object during execution—that is, if you can fully specify the SELECT statement without having to provide a variable—you don't need to specify retrieval arguments.

> **Changing your mind later about using arguments**
> If you decide later that you need arguments, you can return to the Select painter from the DataWindow painter workspace to define the arguments as described here.

If criteria will be provided *during execution* that determine which rows are retrieved, you need to define retrieval arguments when defining the SQL SELECT. For example, consider these situations:

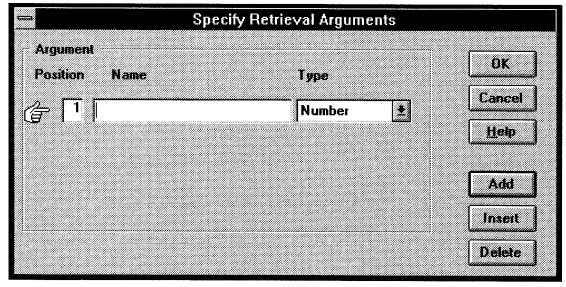
- The user will type an employee ID into an edit box and click a button to retrieve the row in the Employee table for that employee. You don't know the employee ID when you are defining the DataWindow object in the DataWindow painter, so you must have that information passed to the SELECT statement as an argument during execution.
- The user will select a department from a dropdown listbox and click a button to retrieve all rows from a table for that department. The department will be passed as an argument during execution.

> **Using retrieval arguments during execution**
> If a DataWindow object has retrieval arguments, to retrieve data during execution you call the Retrieve PowerScript function and pass the arguments in the function.
>
> For more information, see the *Function Reference*.

❖ To define retrieval arguments:

1 Select Objects➤Retrieval Arguments from the menu bar.

The Specify Retrieval Arguments dialog box displays.



2 Enter a name and data type for each argument.

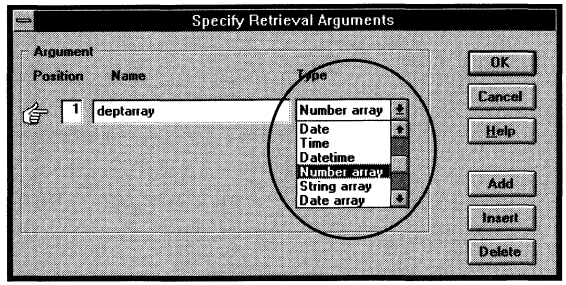
You can enter any valid SQL identifier for the argument name. The position number identifies the position of the argument in the Retrieve PowerScript function you code in a script to retrieve data into the DataWindow object.

3 Click OK.

You return to the Select painter workspace.

Specifying arrays

You can specify an array of values as your retrieval argument. Choose the type of array from the Type dropdown listbox.



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 employee in department 100, 200, or 500. If you want your user to specify the list of departments to retrieve, you would define the retrieval argument as a number array (such as *100, 200, 500*).

In the script that does the retrieval, you declare an array and reference it in the Retrieve function, such as:

```
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 function (such as *100, 200, 500* if x[1] = 100, x[2] = 200, and x[3] = 500).

Referencing retrieval arguments

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 use the following:

- A WHERE clause to limit the data that is retrieved from the database
- An ORDER BY clause to sort the retrieved data before it is brought into the DataWindow object

- A GROUP BY clause to group the retrieved data before it is brought into the DataWindow object
- A HAVING clause to limit the groups specified in the GROUP BY clause

About defining these criteria here
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 choose to 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 16, "Filtering, Sorting, and Grouping Rows."

Specifying retrieval arguments

If you have defined retrieval arguments, you will reference the argument(s) in the WHERE or HAVING clause. For example, if the DataWindow object is retrieving all rows from the Department table where the DeptID matches a value provided by the user during execution, your WHERE clause will look something like this:

```
WHERE DeptID = :Entered_id
```

where Entered_id was defined previously as an argument in the Specify Retrieval Arguments dialog box.

How retrieval arguments are referenced
In SQL statements, variables (called host variables) are always prefaced with a colon to distinguish them from column names.

For example, to reference the variable Entered_id in a SQL statement, enter **:Entered_id**

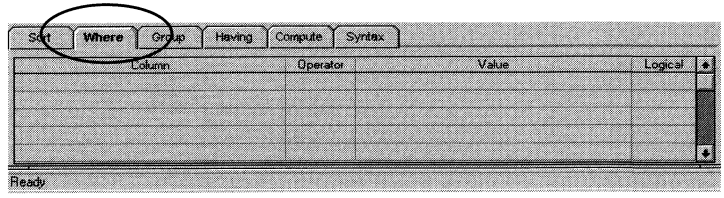
Defining WHERE criteria

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.

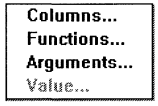
For example, if you are retrieving information about employees, you can limit the employees to those in Sales and Marketing, or to those in Sales and Marketing who make more than $50,000.

❖ To define WHERE criteria:

1 Click the Where tab in the bottom of the workspace.



Each row in the Where tab is a place for entering an expression that limits the grouping of rows.



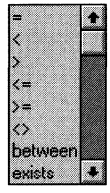
2 Point in the first row under Column and click to display columns.
or
Select Columns from the popup menu.

3 Select the column you want to use in the left-hand side of the expression.

The equality (=) operator displays in the Operator column.

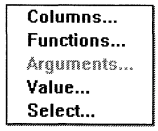
> **To use a function or retrieval argument in the expression**
> To use a function in the expression, select Functions from the popup menu and click on a function. A list of functions provided by the DBMS displays.
>
> If you have defined retrieval arguments, you can use a retrieval argument in the expression. To use a retrieval argument, select Arguments from the popup menu

4 To 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:

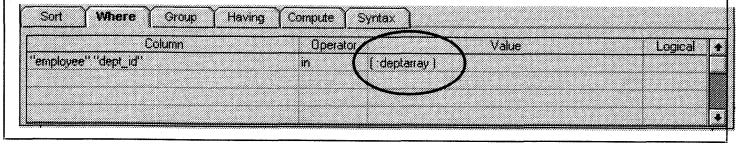
- Type a value.



- Paste a column, function, or retrieval argument (if there is one) by selecting Columns, Functions, or Arguments from the popup menu.

If pasting a retrieval argument that is an array

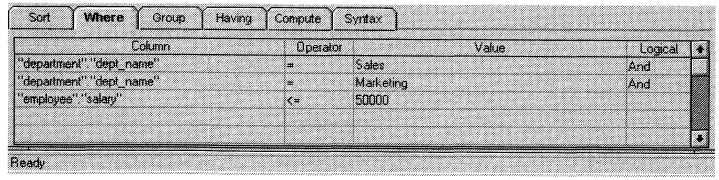
If the retrieval argument is an array being used with the IN operator, make sure you type the parentheses surrounding the retrieval argument, such as:



- Paste a value from the database by selecting Value from the popup menu: PowerBuilder retrieves all rows from the database and displays in a dialog box all values for the column specified in the left-hand side of the expression (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 popup menu: the Nested Select dialog box displays. Here you can define a nested SELECT statement. After you define the nested select statement, click the Return button to return to the Where tab.

6 Continue to define additional Where expressions as needed.

For each additional expression, select a logical operator (AND or OR) to connect your multiple boolean expressions into one expression that PowerBuilder evaluates as true or false to limit the rows that are retrieved.



7 If you haven't already done so, define sorting (Sort tab), grouping (Group tab) and limiting (Having) or click the Design button to go to the DataWindow painter workspace.

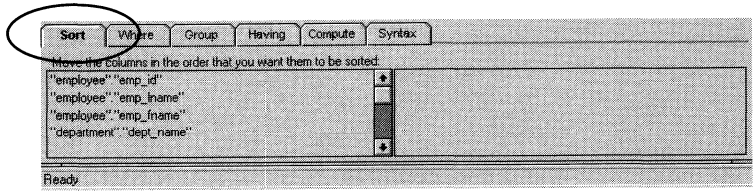
Defining ORDER BY criteria

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 in the bottom of the workspace.



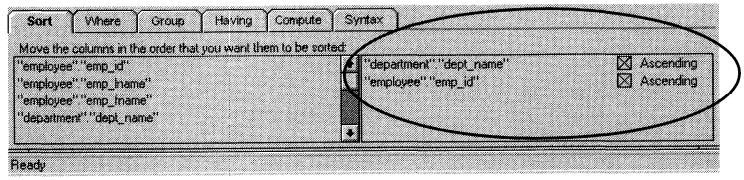
The columns you selected display in the order of selection. You may need to scroll to see your selections.

2 Click the first column you want to sort on and drag it to the right side of the Sort tab.

This specifies the column for the first level of sorting. By default, the column is sorted in ascending order. To specify descending order, deselect the Ascending checkbox.

3 Continue to specify additional columns for sorting in ascending or descending order as needed. Columns are sorted in the order of display in the right side of the Sort tab. To change the sort order, drag the column names in the right side to the desired positions.

In the following, rows will be sorted first by department name; within department, rows will be sorted by employee ID.



4 If you haven't already done so, define limiting (Where tab), grouping (Group tab), and limiting groups (Having) or click the Design button to go to the DataWindow painter.

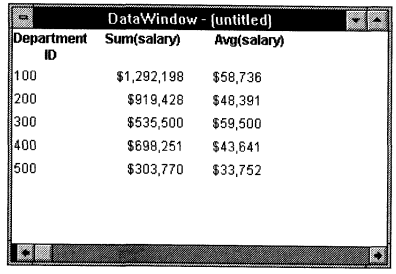
Defining GROUP BY criteria

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.

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. Here 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 Powersoft Demo DB, you will get five rows back, one for each department.

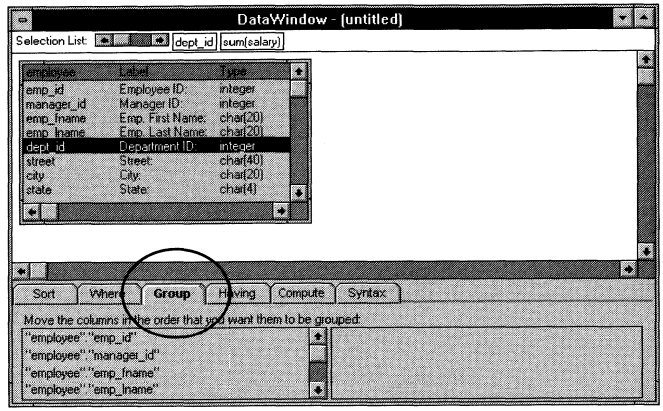


Department ID	Sum(salary)	Avg(salary)
100	$1,292,198	$58,736
200	$919,428	$48,391
300	$535,500	$59,500
400	$698,251	$43,641
500	$303,770	$33,752

For more about GROUP BY, see your DBMS documentation.

❖ To define GROUP BY criteria:

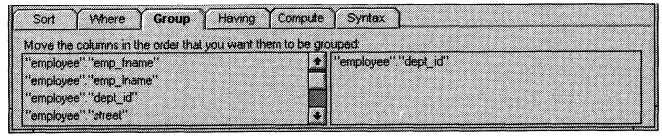
1 Click the Group tab in the bottom of the workspace.



The columns in the tables used in the DataWindow object display in the left side of the Group tab. You may need to scroll to see your selections.

2 Click the first column you want to group on and drag it to the right side of the Group tab.

This specifies the column for grouping. In the following, the DataWindow object will be grouped by department ID.



3 Continue to specify additional columns for grouping within the first grouping column as needed.

Columns are grouped in the order of display in the right side of the Group tab. To change the grouping order, drag the column names in the right side to the desired positions.

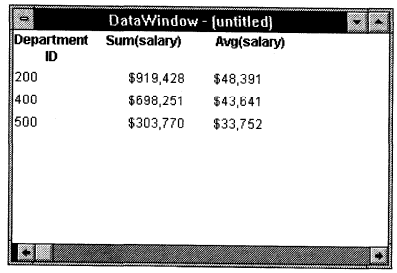
4 If you haven't already done so, define sorting (Sort tab), limiting (Where tab), and limiting groups (Having) or click the Design button to go to the workspace.

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:

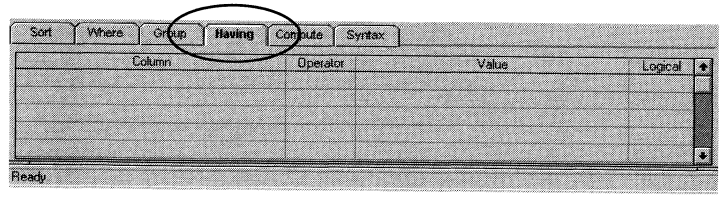
```
SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
HAVING avg(salary) < 50000
```

If you specify this with the Employee table in the Powersoft Demo Database, you will get three rows back, because there are three departments that have average salaries less than $50,000.



❖ To define HAVING criteria:

1 Click the Having tab in the bottom of the workspace.



Each row in the Having tab is a place for entering an expression that limits which groups are retrieved.

2 Specify the criteria for the HAVING clause the same way you specify WHERE criteria.

See "Defining WHERE criteria" on page 427.

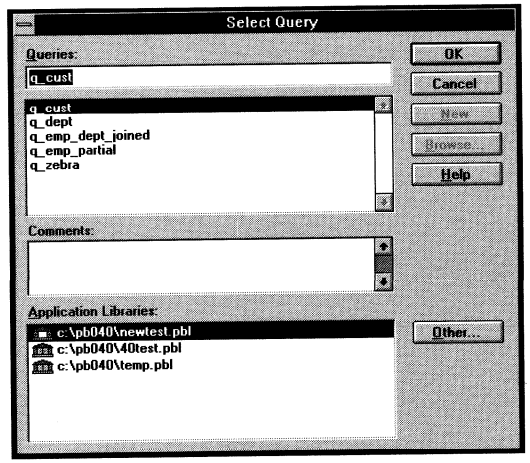
Using Query

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 Click Query in the New DataWindow window and click OK.

The Select Query dialog box displays. It lists all queries that have been defined in the current PowerBuilder library.



2 Select the existing query and click OK.

For information about selecting existing objects from this dialog box, see Chapter 1, "The World of PowerBuilder."

The DataWindow painter workspace displays (with some presentation styles, you need to provide additional information before going to the workspace).

For more information

To learn how to create queries, see "Defining queries" on page 442.

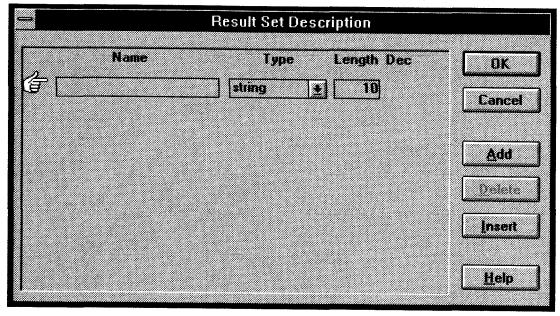
Using External

If the data for the DataWindow object is not coming from a database (either through a native Powersoft 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**.

❖ To define the data using External:

1 Click External in the New DataWindow dialog box and click OK.

The Result Set Description dialog box displays with boxes for you to specify each column in the result set.



2 Enter the name and type of each column. Available data types are listed in the dropdown listbox.

3 Click OK.

You go to the DataWindow painter workspace with the columns you specified in the result set placed in the DataWindow object.

What you do next

In a script, you will need to tell PowerBuilder how to get data into the DataWindow in your application. Typically, you will import data during execution using a PowerScript Import function (such as ImportFile and ImportString) or do some data manipulation and use the SetItem function to populate the DataWindow.

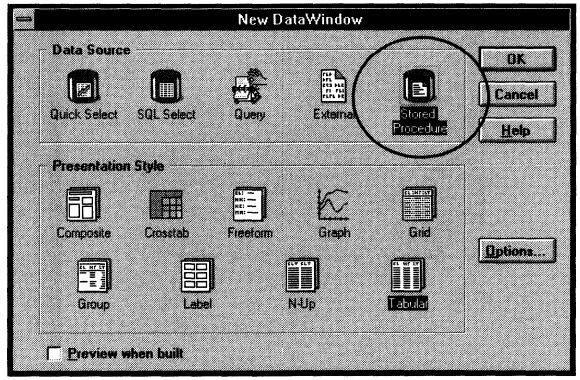
For more about these functions, see the *Function Reference*.

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.

You can specify that the data for a DataWindow object is retrieved through a stored procedure if your DBMS supports stored procedures. For information on support for stored procedures, see your database documentation.

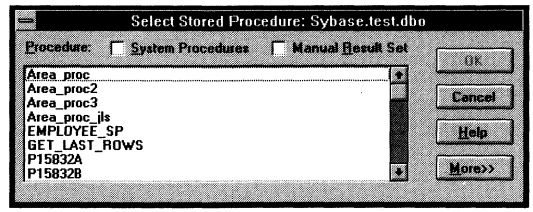
When the Stored Procedure icon displays
The icon for the Stored Procedure data source displays in the New DataWindow dialog box only if the database to which you are connected supports stored procedures.



❖ To define the data using Stored Procedure:

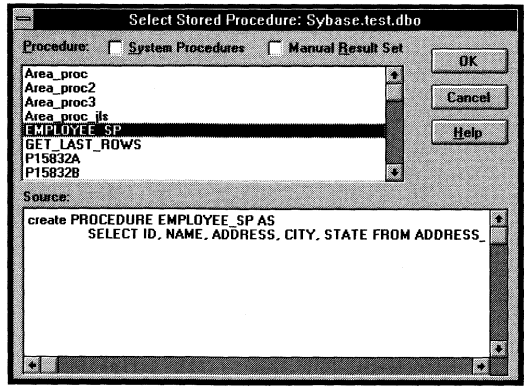
1 Select Stored Procedure as the data source in the New DataWindow dialog box.

The Select Stored Procedure dialog box displays a list of the stored procedures in the current database.

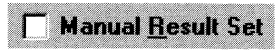


2 Select a stored procedure from the list.

If you want to list system procedures, select the System Procedures checkbox. If you want to see the syntax of the selected stored procedure, click More. If this is not the stored procedure you want, select another one.

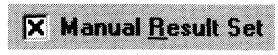


3 If you want PowerBuilder to execute the stored procedure and build the result set description automatically, make sure the Manual Result Set checkbox is deselected and click OK.



PowerBuilder executes the stored procedure and builds the result set description for you. You go to the DataWindow painter workspace with the columns placed in the DataWindow.

4 If you want to manually define the result set description, select the Manual Result Set checkbox and click OK.

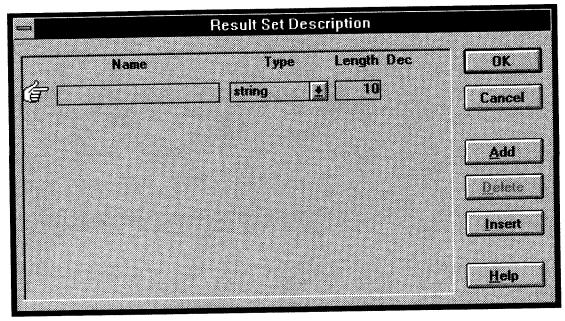


The Result Set Description dialog box displays.

Your preference is saved

PowerBuilder records whether you want it to build result set descriptions automatically for stored procedure DataWindows as the variable Stored_Procedure_Build in the [Data Window] section of PB.INI. The value 1 indicates that PowerBuilder will build the result set description for you; the value zero indicates you will be prompted to define the result set description.

5 If the Result Set Description dialog box displays, enter the name and type of the first column in the result set. To add additional columns, click Add. When you have defined the entire result set, click OK.

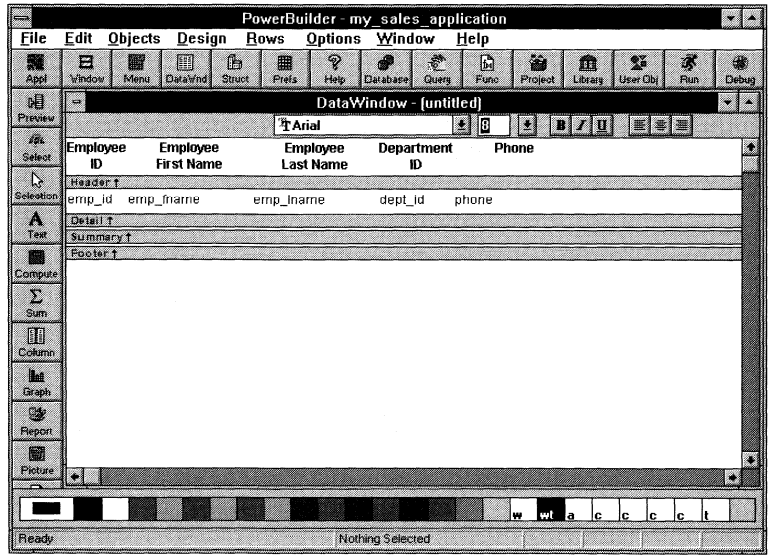


You go to the DataWindow painter workspace with the columns specified in the result set placed in the DataWindow.

Generating and saving a DataWindow object

Once you have selected a presentation style and data source, PowerBuilder generates the DataWindow object and takes you to the DataWindow painter workspace (with some presentation styles, you are prompted for additional information before you go to the workspace).

Here is the workspace for a tabular DataWindow object:



When generating the DataWindow object, PowerBuilder uses the following information from the repository:

For	PowerBuilder uses
Tables	Fonts specified for labels, headings, and data
Columns	Text specified for labels and headings Display formats Edit styles Validation rules

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.

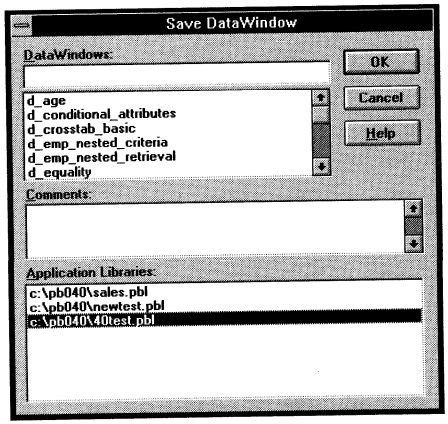
At this point you have a functioning DataWindow object. You should save it before making any changes.

❖ To save a 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 DataWindow painter workspace.

If you have not previously saved the DataWindow object, PowerBuilder displays the Save DataWindow window.



2 Name the DataWindow object in the DataWindows box (see below).

3 Write comments to describe the DataWindow object. These comments display in the Library painter. It is a good idea to use comments so you and others can easily remember the purpose of the object later.

4 Specify the library in which to save the object.

5 Click OK.

PowerBuilder saves the DataWindow object in the specified library.

Naming the DataWindow object

The DataWindow object name can be any valid PowerBuilder identifier with up to 40 characters.

For information about PowerBuilder identifiers, see *PowerScript Language*.

A recommendation

When you name DataWindow objects, you should use a two-part name: a standard prefix that identifies the object as a DataWindow (such as d_) and a suffix that helps you identify the particular object.

For example, you might name a DataWindow object that displays employee data d_emp_data.

Defining queries

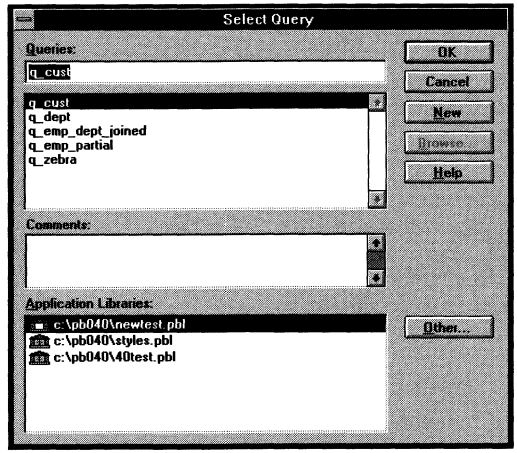
A query is a SQL SELECT statement created with the Query painter and saved with a name so that it can be used repeatedly as the data source for a DataWindow object.

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 Click the Query button in the PowerBar or PowerPanel.

The Select Query dialog box displays.



2 Click New to define a new query.

The Select Tables dialog box displays.

At this point, you select tables and 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. The Query painter is virtually the same as the Select painter.

For more about defining the SELECT statement, see "Using SQL Select" on page 413.

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:

- In the Query painter, click the Preview button.
  or
  Select Options➤Preview from the menu bar.

  PowerBuilder retrieves the rows satisfying the currently defined query in a grid-style DataWindow.

At this point, you can manipulate the retrieved data as you do in the Data Manipulation painter (except that you cannot change data in the database).

For more about the Data Manipulation painter, see Chapter 12, "Managing the Database."

When you have finished previewing the query, click the Design button to return to the Query painter workspace.

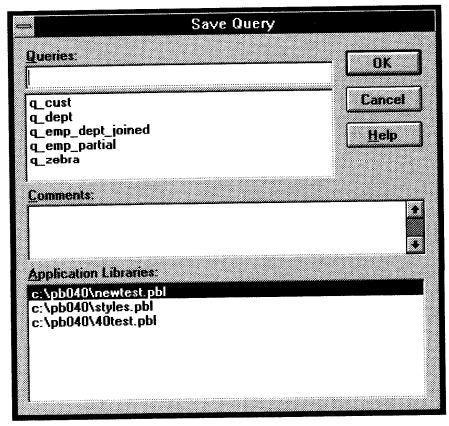
Saving the query

❖ To save a query:

1 Select File➤Save 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 workspace.

  If you have not previously saved the query, PowerBuilder displays the Save Query dialog box.



2 Name the query in the Queries box (see below).

3 Write comments to describe the query. These comments display in the Library painter. It is a good idea to use comments so you and others can easily remember the purpose of the query later.

4 Specify the library in which to save the query.

5 Click OK.

PowerBuilder saves the query in the specified library.

Naming the query

The query name can be any valid PowerBuilder identifier with up to 40 characters.

For information about PowerBuilder identifiers, see *PowerScript Language*.

A recommendation

When you name queries, you should use a two-part name: a standard prefix that identifies the object as a query (such as q_) and a suffix that helps you identify the particular query.

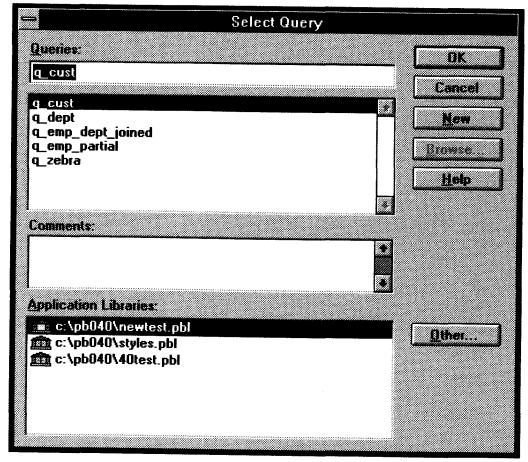
For example, you might name a query that displays employee data q_emp_data.

Modifying a query

❖ To modify a query:

1 Click the Query button in the PowerBar or PowerPanel.

The Select Query dialog box displays.



2 Select the query you want to modify and click OK.

For more information about selecting an object in this dialog box, see Chapter 1, "The World of PowerBuilder."

3 Modify the query as needed.

What's next

Usually after you have generated your DataWindow, you will want to preview (execute) it to see how it looks. After that, you will probably want to use the DataWindow painter workspace to enhance the DataWindow object before using it. PowerBuilder provides many ways for you to make a DataWindow object easier to use and more informative for users.

For more information, see Chapter 14, "Enhancing DataWindow Objects."

CHAPTER 14

Enhancing DataWindow Objects

About this chapter

Before you put a DataWindow object into production, you will probably want to enhance it to make it easier to use and interpret data. You do that in the DataWindow painter workspace. This chapter describes basic enhancements you can make to a DataWindow object.

Contents

Related topics

Other ways to enhance DataWindow objects are covered in later chapters:

Chapter	Contents
15 "Displaying and Validating Data"	How to specify display formats, edit styles, and validation rules for column data
16 "Filtering, Sorting, and Grouping Rows"	How to limit which rows are displayed, the order in which they are displayed, and whether they are divided into groups
17 "Using Nested Reports"	How to place reports inside DataWindow objects

Chapter	Contents
18 "Working with Graphs"	How to use graphs to visually present information retrieved in a DataWindow object
19 "Working with Crosstabs"	How to use crosstabs to present analyses of data retrieved in a DataWindow object

Working in the workspace

Once you have specified your presentation style and data source, PowerBuilder generates a basic DataWindow object and places you in the DataWindow painter workspace where you can enhance the DataWindow.

This section presents an overview of working in the workspace:

- Understanding the workspace (described next)
- Using the toolbars (page 452)
- Using the popup menus (page 453)
- Keyboard shortcuts (page 454)
- Selecting objects (page 455)
- Resizing bands (page 457)
- Using zoom (page 457)
- Undoing changes (page 457)

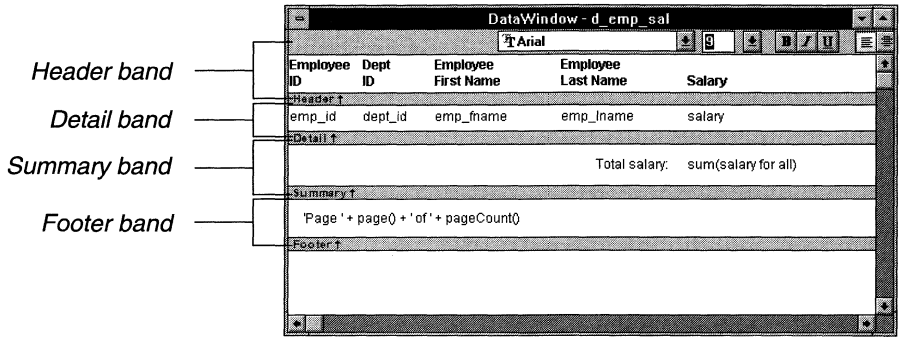
Applicability of information
Some of the information presented in this section doesn't apply to all the DataWindow presentation styles.

Understanding the workspace

With most presentation styles, the DataWindow painter workspace is divided into areas called **bands**. Each band corresponds to a section of the displayed DataWindow object. (Label and Graph DataWindow objects are the exceptions; they have no bands.)

A DataWindow object is 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.

Here is the workspace for a tabular DataWindow object:



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

About the header band

The header band contains heading information that is displayed at the top of every screen or page during execution.

When PowerBuilder generates the basic DataWindow object, 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 (for example, a date) in the header band and can include pictures, graphic objects, 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 PowerScript function in the header band.

For more about using computed fields, see "Adding computed fields" on page 507.

About 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:

$$\frac{\textit{Height of the DataWindow object} - \textit{Height of headers and footers}}{\textit{Height of the detail band}}$$

When PowerBuilder generates the basic DataWindow object, 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 workspace
If the DataWindow uses one table, the names of the columns in the workspace are the same as the names in the table.

If the DataWindow uses more than one table, the names of the columns in the workspace 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 names.)

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 objects, such as text, pictures, drawing objects, and graphs.

About the summary and footer bands

You use the summary and footer bands of the DataWindow 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.

These bands can contain any information you want, including text, drawing objects, graphs, and computed fields containing aggregate totals.

Using the toolbars

Like other painters, the DataWindow painter contains a customizable PainterBar. Most of the items in the default PainterBar are used to place objects in the DataWindow object, just as you use the PainterBar to place controls in a window in the Window painter.

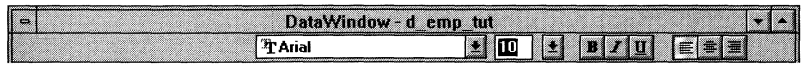
For example, to place a rectangle in the DataWindow object, click the Rectangle button, then click the location in the DataWindow object where you want the rectangle to appear.

For complete information about manipulating the PainterBar, see Chapter 1, "The World of PowerBuilder."

About the StyleBar and ColorBar

Like the Window painter, the DataWindow painter also has two other toolbars:

- A StyleBar, which you can use to apply text attributes



- A ColorBar, which you can use to apply colors to elements of the DataWindow control

You can display these toolbars in the DataWindow painter window or in the PowerBuilder frame, which encompasses all windows. You can also hide the ColorBar.

❖ To specify the position of the StyleBar or ColorBar:

1. Select Color Toolbar or Text Style Toolbar from the Options menu.
2. Choose the position of the toolbar from the cascading menu.

Using the popup menus

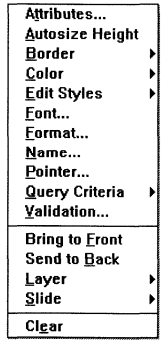
Each element of the DataWindow object (such as text, columns, computed fields, bands, graphs, crosstabs, even the DataWindow object itself) has a popup menu that you can use to modify the object's attributes.

❖ To use a popup menu:

1. Position the mouse over the object you want to modify.
2. Click the right mouse button.

   The popup menu displays.

Here is the popup menu for a column:



When you want to modify an object, display its popup menu and pick the property you want to change. The operations you can perform on objects are described in this and the next chapter.

Keyboard shortcuts

The following table lists the keyboard shortcuts available in the DataWindow painter:

Action	Key combination	Comments
Boldface text	CTRL+B	Toggles boldface on and off
Center text	CTRL+N	
Clear	DEL	Clears all selected objects
Close painter	CTRL+F4	
Close window	ALT+F4	Available everywhere in PowerBuilder
Debug	CTRL+D	Go to Debug painter
Edit text	CTRL+E	Activates the Text box in the StyleBar
File editor	SHIFT+F6	Available everywhere in PowerBuilder
Font family	CTRL+F	Activates the Font box in the StyleBar
Italicize text	CTRL+I	Toggles italics on and off
Left justify text	CTRL+L	
Move object	ARROW	
Next child window	CTRL+F6	
PowerPanel	CTRL+P	Available everywhere in PowerBuilder
Preview DataWindow	CTRL+W	Executes the DataWindow
Resize object	SHIFT+ARROW	
Return focus to control	CTRL+O	Returns focus to the control from the StyleBar
Right justify text	CTRL+G	
Run	CTRL+R	Available everywhere in PowerBuilder
Select above	CTRL+UP ARROW	Selects all objects above the currently selected object

Action	Key combination	Comments
Select all	CTRL+A	Selects all objects in DataWindow
Select below	CTRL+DOWN ARROW	Selects all objects below the currently selected object
Select left	CTRL+LEFT ARROW	Selects all objects that are left of the currently selected object
Select right	CTRL+RIGHT ARROW	Selects all objects that are right of the currently selected object
Switch to	CTRL+ESC, ALT+ESC	Available everywhere in PowerBuilder
Tab	TAB	Tabs to next object
Tab backwards	SHIFT+TAB	Tabs to previous object
Underline text	CTRL+U	Toggles underline on and off
Undo/Redo	CTRL+Z	Undoes the most recent change (including the most recent undo)

Selecting objects

The DataWindow painter provides several ways for you to select objects to act on. You can select multiple objects and can act upon all the selected objects as a unit. For example, you can move all of them or change the fonts used to display text for all of them.

❖ To select one object:

- Click it.

  The object displays with handles on it. Previously selected objects are no longer selected.

❖ To select neighboring multiple objects:

1. Press and hold the left mouse button at one corner of the neighboring objects.
2. Drag the mouse over the objects you want to select.

   PowerBuilder displays a bounding box.

3 Release the mouse button.

All the objects in the region are selected.

❖ To select non-neighboring multiple objects:

1 Click the first object.

2 Press and hold the CTRL key and click additional objects.

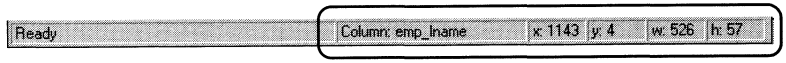
All the objects are selected.

❖ To select objects by type or position in the DataWindow:

To select	Select this item from the menu bar or press this shortcut key	
All objects	Edit➤Select➤Select All	(CTRL+A)
All text	Edit➤Select➤Select Text	
All columns	Edit➤Select➤Select Columns	
Objects in relation to the currently selected object	Edit➤Select➤Select Above Edit➤Select➤Select Below Edit➤Select➤Select Left Edit➤Select➤Select Right	(CTRL+UP ARROW) (CTRL+DOWN ARROW) (CTRL+LEFT ARROW) (CTRL+RIGHT ARROW)

Displaying information about the selected object

The name, X and Y coordinates, width, and height of the selected object is displayed in the MicroHelp bar:



If multiple objects are selected, *Group Selected* displays in the Name area and the coordinates and size do not display.

Resizing bands

You can change the size of any band in the DataWindow object.

❖ To resize a band:

- Press the left mouse button on the bar representing the band and drag the bar up to shrink the band or down to enlarge the band.

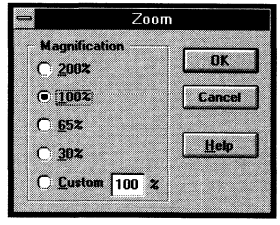
Using zoom

You can zoom the display in and out so you can get a better idea of how your DataWindow object looks. For example, if you are working with a large DataWindow object, you can zoom out so you can see all of it on your screen. Or you can zoom in on a group of objects to better see their details.

❖ To zoom the display:

1 Select Design➤Zoom from the menu bar.

The Zoom dialog box displays.



2 Select a built-in zoom percentage.
or
Set a custom zoom percentage by typing an integer in the Custom box.

PowerBuilder adjusts the display of the DataWindow object.

Undoing changes

You can undo your most recent change in the workspace by pressing CTRL+Z or selecting Edit➤Undo from the menu bar. (If you have just undone an action, the menu item changes to Edit➤Redo.)

Previewing a DataWindow object

You can preview a DataWindow object to view it as it will appear to the user and test the processing that takes place in it. While previewing a DataWindow object, you can:

- Retrieve data (see page 460)
- Modify data (page 461)
- Sort and filter data (page 462)
- Get information about data (page 465)
- Import data (page 465)
- See what the DataWindow will look like when printed (page 466)
- Print data (page 469)
- Save data in external formats (page 470)
- Work with Powersoft report files (PSR files) (page 471)
- Mail reports (page 474)

Changes you make to the DataWindow object while previewing are maintained when you return to the workspace. For example, if you specify sorting while previewing, the sorting becomes part of the design of the DataWindow object.

❖ To preview the DataWindow object:

1 Click the Preview button in the PainterBar.
or
Select Design➤Preview from the menu bar.
or
Press CTRL+W.

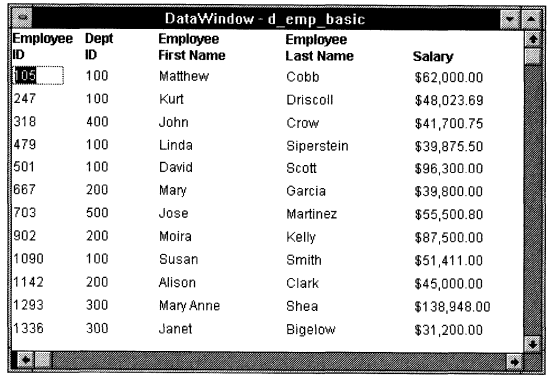
You are now in preview. The bars that indicate the bands disappear, and by default PowerBuilder retrieves all the rows from the database (you are prompted to supply arguments if you defined retrieval arguments).

> **In external DataWindow objects**
> If the DataWindow uses the External data source, no data is retrieved. You can import data, as described on page 465.

In DataWindow objects that have stored data
If the DataWindow has stored data in it, no data is retrieved from the database.

For more information, see "Storing data in a DataWindow object" on page 515.

As the rows are being retrieved, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval.



Employee ID	Dept ID	Employee First Name	Employee Last Name	Salary
105	100	Matthew	Cobb	$62,000.00
247	100	Kurt	Driscoll	$48,023.69
318	400	John	Crow	$41,700.75
479	100	Linda	Siperstein	$39,875.50
501	100	David	Scott	$96,300.00
667	200	Mary	Garcia	$39,800.00
703	500	Jose	Martinez	$55,500.80
902	200	Moira	Kelly	$87,500.00
1090	100	Susan	Smith	$51,411.00
1142	200	Alison	Clark	$45,000.00
1293	300	Mary Anne	Shea	$138,948.00
1336	300	Janet	Bigelow	$31,200.00

Suppressing automatic retrieval
If you don't want PowerBuilder to automatically retrieve rows when you enter preview, set PreviewRetrieve to 0 in the DataWindow section in the Preferences painter.

2 Test your DataWindow object. For example, modify some data, update the database, re-retrieve rows, and so on, as described below.

3 To leave preview, click the Design button.

You return to the workspace.

PowerBuilder caches the data

By default, after PowerBuilder first retrieves data during preview, it stores the data internally. If you go back to design mode and then return to preview, PowerBuilder displays the stored data instead of going out to the database and re-retrieving rows. This can save you a lot of time since data retrieval can be time-consuming.

SHARE must be loaded
In order for PowerBuilder under Windows to cache the data, you must be running the DOS SHARE program. (If you are using Windows for Workgroups, SHARE is built in; do not use the DOS SHARE program.)

For more about SHARE, see your DOS documentation.

However, if you are working with data that is constantly changing, then the data in preview and the data in the database can get out of sync. Generally you don't care about this when you are simply previewing a DataWindow. If you do care, there are two things you can do:

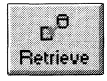
- Turn off the caching of retrieved data by setting Preview_RetainData to 0 in the DataWindow section in the Preferences painter. From then on, PowerBuilder always retrieves data from the database whenever it needs to display data.
- Explicitly retrieve data, as described next.

Retrieving data

By default, PowerBuilder retrieves rows when you enter preview for the first time in a session (unless data has been stored in the DataWindow object, as described in "Storing data in a DataWindow object" on page 515).

You can also manually retrieve rows anytime.

❖ To retrieve rows from the database:



- Click the Retrieve button in the PainterBar.
  or
  Select Rows➤Retrieve from the menu bar.

  PowerBuilder retrieves all the rows (you are prompted to supply arguments if you defined retrieval arguments).

  As rows are being retrieved, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval.

Modifying data

You can add, modify, or delete rows while previewing. 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.

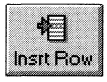
For more about updating DataWindow objects, see "Controlling updates" on page 518.

❖ To modify existing data:

- Tab to the field and enter a new value. Preview 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 while previewing is equivalent to calling the InsertRow function and then the ScrollToRow function during execution.

❖ 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 while previewing is equivalent to calling the DeleteRow function during execution.

❖ 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 while previewing is equivalent to calling the Update function during execution.

Sorting and filtering data

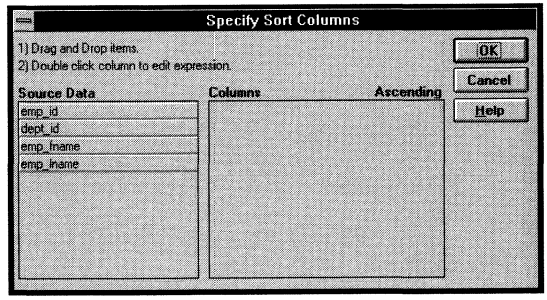
You can define and use sort criteria and filters for the rows. The sort criteria and filters you define while previewing are maintained when you return to the workspace.

Sorting rows

❖ To sort the rows:

1 Select Rows➤Sort from the menu bar.

The Specify Sort Columns dialog box displays. It lists the columns in the source data. If you have already specified sorting in the workspace, the current sort columns are listed.



2 Drag the columns that you want to sort the rows by to the Columns box 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 it outside the dialog box.

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*.

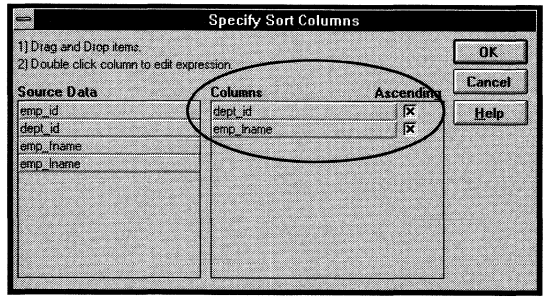
To specify expressions, double-click a column name and modify the expression in the Modify Expression dialog box. When you are finished, click OK.

3 Click OK.

PowerBuilder sorts the rows.

Example

For example, to sort employees by department, and within department by last name, specify the following:



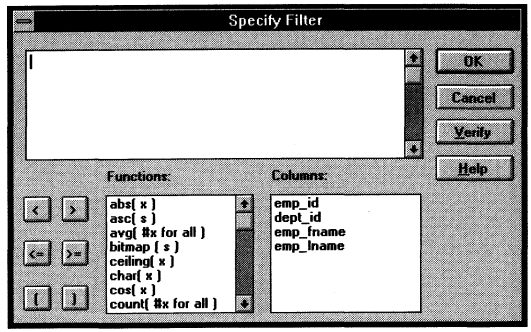
Filtering rows

You can limit which rows are displayed by defining a filter. A filter is like selection criteria with one important difference: selection criteria limit the rows that are retrieved from the database. Filters specify which of the retrieved rows are displayed.

❖ 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 will be displayed. You can paste PowerScript functions, columns, and operators in the expression. You can use the logical operators OR and AND to connect expressions.

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.

Examples

For example, to display only employees whose salary is greater than $60,000, specify this filter:

```
salary > 60000
```

To display only active employees (employees whose status is A), specify the following filter. The A is in quotes because Status is a string column.

```
status = "A"
```

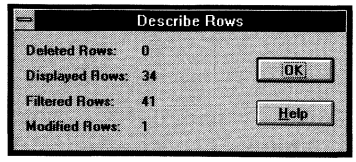
Viewing row information

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.



The Describe Rows dialog box shows 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.

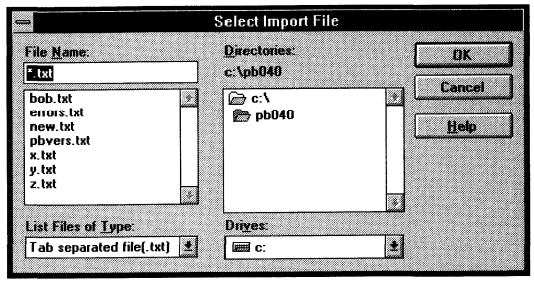
Importing data

While previewing, you can import and display data from an external source, 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 that you can import into the painter are shown in the List Files of Type dropdown listbox.

3 Click OK.

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 retrieved columns
When importing data from a file, the data must match all the columns in the retrieved data (typically, the columns specified in the SELECT statement), not just the columns that are displayed in the DataWindow object.

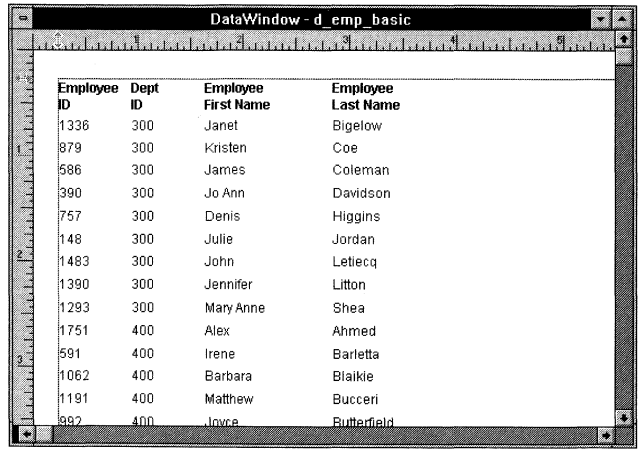
Using print preview

You can print the data displayed while previewing. Before printing, you can preview the output on the screen.

❖ To preview printed output before printing:

- Select File➤Print Preview from the menu bar.

  Preview displays the DataWindow object as it will print. Rulers display around the page borders. (You can turn off the display of rulers by toggling File➤Print Preview Rulers from the menu bar.)

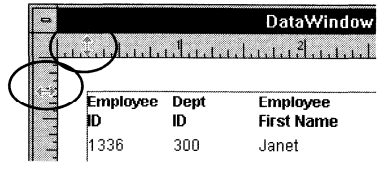


Changing margins

You can dynamically change margins while previewing a DataWindow.

❖ To change the margins while previewing:

- 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.



Changing margins during execution

Using the Modify function, you can display a DataWindow in print preview during execution. While in print preview your user 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 function to display the page count in your application (for example, in MicroHelp), you should code a script for the user event to recalculate the page count.

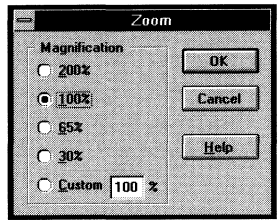
Zooming the page

You can reduce or enlarge the amount of the page that displays on your screen. This does not affect the printed output.

❖ To zoom the page:

1 Select File➤Print Preview Zoom from the menu bar.

The Zoom dialog box displays.



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.

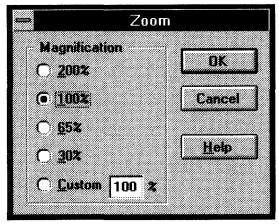
Zooming the contents

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:

1 Select Display➤Zoom from the menu bar.

The Zoom dialog box displays.



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

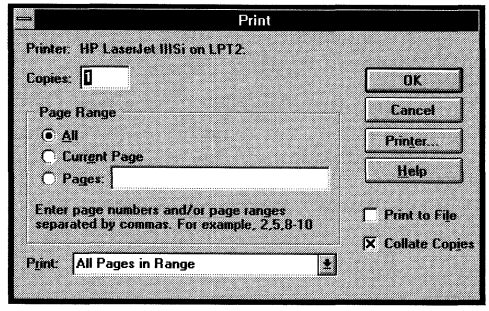
You can print your DataWindow while previewing. You can print all pages, a range of pages, only the current page, or only odd or even pages. You can also specify whether you want multiple copies, collated copies, and printing to a file.

To change printers or settings before printing
You can choose File➤Printer Setup from the menu bar in preview.

❖ To print:

1 Select File➤Print from the menu bar.

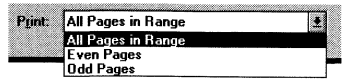
The Print dialog box displays.



2 Specify the number of copies of the report.

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 dropdown listbox.

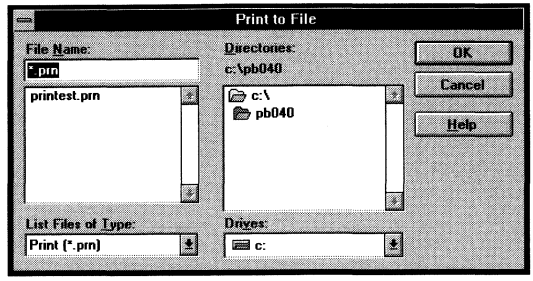


5 If you want to print to a file rather than on the printer, select the Print to File checkbox.

6 If you want to change the collating option, deselect or select the checkbox.

7 Click OK.

8 If you specified print to file, the Print to File dialog box displays. Enter a filename. The extension PRN identifies it as a file prepared for the printer. Change drive and/or directory if you want. Click OK.



PowerBuilder prints the report on the default printer (or to a file if specified). All displayed rows are included.

If you've printed to a file
Under DOS, you can print the file on the printer by copying it at the command line to the printer. For example, if you've printed a report to a file named monthly.prn, you could print it on the printer specified as lpt2 with this DOS command:

```
copy /b monthly.prn lpt2
```

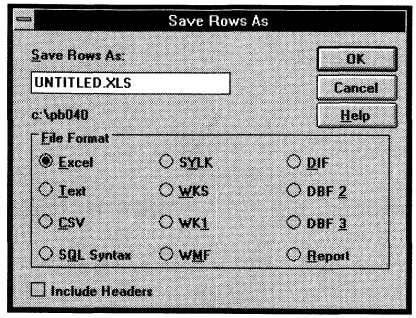
Saving data

You can save the data retrieved during preview 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. When you choose a format, PowerBuilder supplies the appropriate file extension.

To save a PSR file
Choose Report to save a Powersoft PSR file.

For more information about the Report file format, see "Working with PSR files" next.

3 Name the file.

4 If you want the columns' headings saved in the file, select the Include Headers checkbox.

5 Click OK.

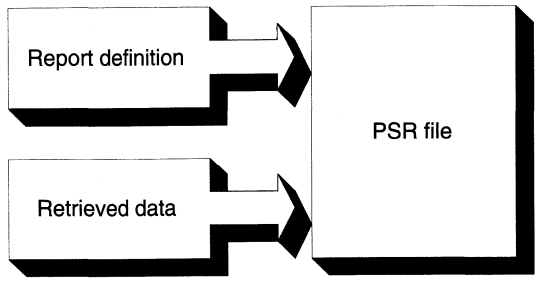
PowerBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Working with PSR files

A PSR file is a special file with the extension PSR created by PowerBuilder or InfoMaker. PSR stands for **P**owersoft **R**eport.

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.

About reports
A report is the same as a nonupdatable DataWindow. For more information, see "About reports and the Report painter" in Chapter 13, "Defining DataWindow Objects."



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.

How PSR files are created

PowerBuilder creates a PSR file when you:

- Save data while previewing in the Report file format.

  For more information, see "Saving data" on page 470.

- Mail a report to an InfoMaker user using electronic mail.

  For more information, see "Mailing reports" on page 474.

Opening a PSR file

PSR files are used primarily by InfoMaker, Powersoft's end-user product. 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.

InfoMaker users can open a PSR file in File Manager, in a mail message, and using the File menu in the Report painter. PowerBuilder users can open a PSR file in the Report painter.

Windows and PSR files
When InfoMaker is installed, the Powersoft report file type is registered with Windows and associated with InfoMaker.

❖ To open a PSR file in InfoMaker using File Manager:

- Double-click the PSR filename.

  InfoMaker displays the report in preview.

❖ To open a PSR file in InfoMaker from a mail message:

- Double-click the PSR filename.

  InfoMaker displays the report in preview.

You can also open PSR files using the File menu in the Report painter in PowerBuilder (and InfoMaker).

❖ To open a PSR file in the Report painter in PowerBuilder:

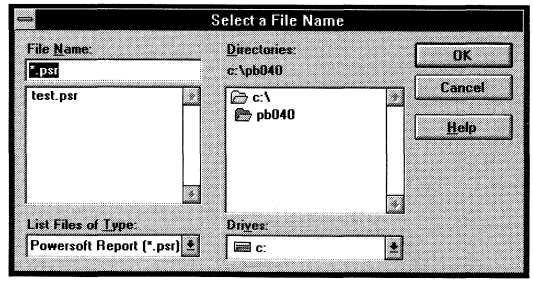
1 Set the DefaultFileOrLib variable in the [Data Window] section of the PB.INI file to allow you to open PSR files. DefaultFileOrLib determines which objects you can open in the Report painter:

Value of DefaultFileOrLib	What you can open in the Report painter
0	DataWindow objects (reports) in PBLs
1	PSR files
2	DataWindow objects in PBLs and PSR files

2 Open a PSR file as follows:

- If DefaultFileOrLib is 1, open the Report painter or select File➤Open from the menu bar of the Report painter.
- If DefaultFileOrLib is 2, open the Report painter and select a DataWindow object from a library in the Open dialog box. You go to the workspace. Now select File➤Open File from the menu bar to open a PSR file.

In either case, the Select a File Name dialog box displays listing the PSR files in the current directory.



3 Select the PSR file you want. Change drives and directories if needed.

PowerBuilder displays the report in the workspace.

4 Click the Preview button to preview the report.

PSR files and retrieval

When you are previewing a PSR file, you see the data that was saved in the PSR file when it was created. This is true until you explicitly retrieve data again using the Retrieve button or the Rows➤Retrieve command.

If you attempt to retrieve data with a PSR file, you must be sure that you are properly connected to the right database. Otherwise, you will receive a database error message.

If you retrieve new data while previewing a PSR file, you cannot go back to the old data contained in the file. To go back to the old data, you can leave the PSR file without saving and then reopen the PSR file.

For more information

For complete information about using InfoMaker to work with PSR files, see the InfoMaker documentation.

Mailing reports

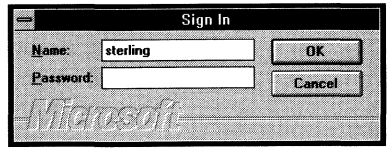
While previewing in the Report painter (but not in the DataWindow painter), you can mail a report as a PSR file to an InfoMaker user who is using a MAPI-compliant mail system, such as Microsoft Mail. (MAPI stands for *messaging application program interface* and is one of the programming interfaces to mail systems.)

For more about PSR files, see "Working with PSR files" on page 471.

❖ To mail a report:

1 While previewing a report in the Report painter, select File➤Send from the menu bar.

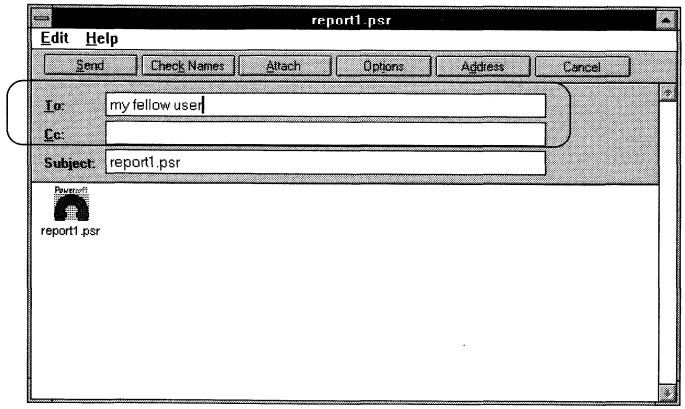
If you are not already logged on to Mail, you will be prompted for your password.



2 Enter your password and click OK.

A form for mailing the report displays. PowerBuilder creates and attaches the appropriate PSR file (which holds the report and data).

3 Complete the form and send the message.



PowerBuilder mails the PSR file.

The recipient can open the report by double-clicking it if InfoMaker is installed.

Working in a grid DataWindow

If you are previewing a grid-style DataWindow object, you have full access to the functionality of a grid DataWindow. You can do the following:

- 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 also available to your users
Users of your application can also manipulate columns these ways in a grid DataWindow during execution.

❖ To resize a column:

1 Position the mouse pointer at a column boundary.

The pointer changes to a two-headed arrow.

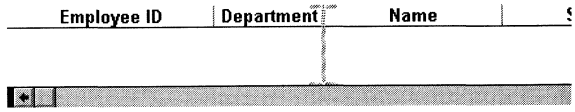
2 Press and hold the left mouse button and drag the mouse to move the boundary.

3 Release the mouse button when the column is the correct width.

❖ To reorder columns:

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.

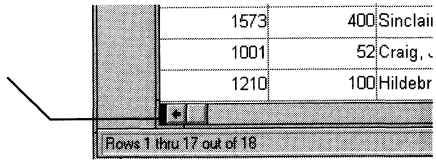
3 Release the mouse button.

The column is reordered.

❖ To use split horizontal scrolling:

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



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.

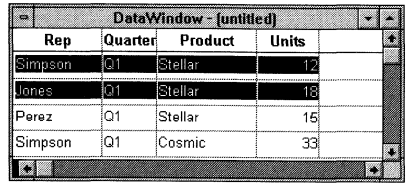
You now have two independently scrolling regions in the grid DataWindow object.

❖ To copy data to the clipboard:

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.
or
Press CTRL+C.

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.

Modifying general DataWindow attributes

This section describes the general DataWindow attributes that you can modify in the workspace. It covers:

- Changing the DataWindow object style (described next)
- Setting colors (page 480)
- Specifying pointers to use when the mouse is over the DataWindow (page 481)
- Modifying text (page 482)
- Defining the tab order (page 483)
- Naming objects in the DataWindow (page 485)
- Using borders (page 485)
- Specifying variable-height detail bands (page 486)
- Modifying the data source (page 488)

Changing the DataWindow object style

A DataWindow object has a Style window, which specifies:

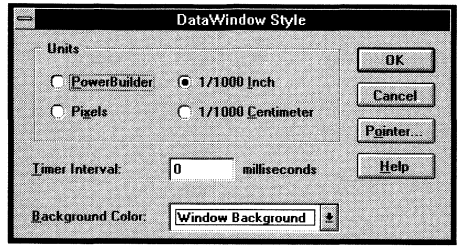
- The unit of measure used in the DataWindow object
- A timer interval for events in the DataWindow object
- A background color for the DataWindow object

PowerBuilder assigns defaults when it generates the basic DataWindow object. You can change the defaults.

❖ To change the default style attributes:

1 Double-click the background of the DataWindow object.
or
Select Design➤DataWindow Style from the menu bar.

The DataWindow Style dialog box displays.



2 Click the unit of measure you want to use to specify distances when working with the DataWindow object:

- PowerBuilder units
- Pixels (smallest element on the display monitor)
- Thousandths of an inch
- Thousandths of a centimeter

> **When printing**
> If you plan to print the contents of the DataWindow object during execution, 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. The default is 0 (which means 60,000 milliseconds or 1 minute).

4 Select a background color from the Background Color dropdown listbox. The default color is the window background color.

> **Another method**
> You can also set the background color by clicking the right mouse button on the background of the DataWindow object and selecting Color from the popup menu.

5 Click OK.

PowerBuilder saves your selections and returns you to the workspace.

Setting colors

You can set different colors for each element of a DataWindow object to enhance the display of information.

❖ To set colors:

- Do one of the following:

To set colors for	Do this
The DataWindow object's background	Position the mouse on an empty spot in the DataWindow object and select Color from the popup menu for the DataWindow itself. (You can also double-click on an empty spot and set the color from the DataWindow Style window.)
A band	Position the mouse pointer on the bar that represents the band and select Color from the popup menu. The choice you make here overrides the background color for the DataWindow object.
An object	Position the mouse pointer on the object and select Color from the popup menu. You can set colors for an object's background and text.

Using the ColorBar
You can also set colors (and define your own custom colors) using the ColorBar.

For more information, see "Choosing Colors" in Chapter 7, "Working with Controls."

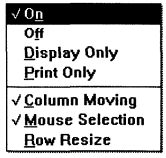
Specifying properties of a grid DataWindow

In grid DataWindow objects, you can specify:

- When grid lines are displayed
- How users can interact with the DataWindow during execution

❖ To specify basic grid DataWindow properties:

1 Position the mouse pointer on the background in a grid DataWindow and display the popup menu.



2 Select Grid from the popup menu.

A cascading menu displays.

3 Select the option you want.

Option	Result
On	Grid lines always display
Off	Grid lines never display (users cannot resize columns during execution)
Display Only	Grid lines display only when DataWindow object displays online
Print Only	Grid lines display only when the contents of the DataWindow are printed
Column Moving	Users can move columns during execution
Mouse Selection	Users can select data during execution (and, for example, copy it to the clipboard)
Row Resize	Users can resize rows during execution

Specifying pointers

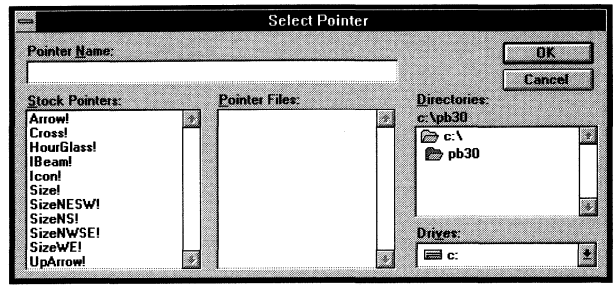
Just as with colors, you can specify different pointers to use when the user's mouse is over a particular area of the DataWindow object. For example, you might want to change the pointer when the mouse is over a column whose data cannot be changed.

❖ To change the mouse pointer used during execution:

1 Position the mouse over the element of the DataWindow whose pointer you want to define. You can set a pointer for the entire DataWindow object, specific bands, and specific objects.

2 Select Pointer from the popup menu.

The Select Pointer dialog box displays.



3 Choose the pointer either from the Stock Pointers list or, if you have files containing pointer definitions (CUR files), from the Pointer Files list.

The chosen pointer is shown to the right of the Pointer Name box.

4 Click OK to return to the workspace.

Modifying text

When PowerBuilder initially generates the basic DataWindow object, it uses the following:

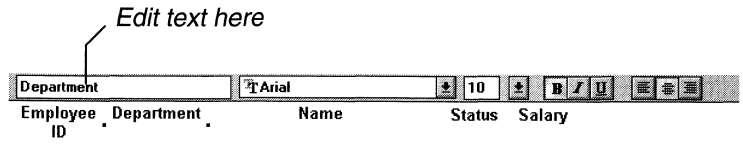
- For the text and alignment of column headings and labels, PowerBuilder uses the extended column attributes made in the Database painter—definitions made by selecting Header from a column's popup menu in the Database painter workspace.
- For fonts, PowerBuilder uses the definitions made in the Database painter for the table—definitions made by selecting Fonts from the table's popup menu in the Database painter workspace. If you didn't 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:

1 Select the text.

The first box in the StyleBar is now active.



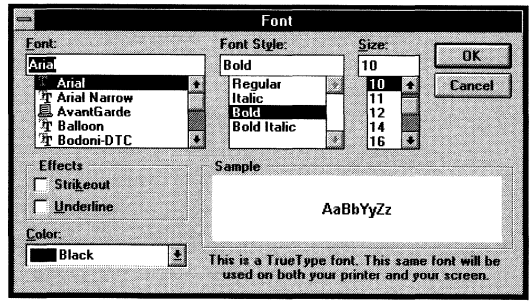
2 Type the new text. Use ~n~r to embed a newline character in the text.

❖ To change the text attributes for an object:

1 Select the object.

2 Do one of the following:

- Change the text attributes in the StyleBar, as shown above.
- Select Font from the object's popup menu and change the attributes from the Font dialog box.



Defining the tab order

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 during execution. PowerBuilder assigns tab values in increments of 10 in left-to-right and top-to-bottom order.

Tab order is not used in the workspace
Tab order is used when a DataWindow object is executed, but it is not used in the DataWindow painter workspace. In the workspace, the TAB key moves to the DataWindow objects in the order in which they were placed in the workspace.

If the DataWindow contains 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 "Controlling updates" on page 518.

How to change the tab order

❖ **To change the tab order:**

1. Select Design➤Tab Order from the menu bar.

   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 (0-9999); 0 removes the column from the tab order (the user cannot tab to the column). It doesn't 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 Design➤Tab Order from the menu bar 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 window and to allow space to insert new columns in the tab order.

Changing tab order during execution
To change tab order in a script, use the SetTabOrder function.

Naming objects

You use names to identify columns and other objects (such as graphs) in validation rules, filters, PowerScript functions, and DataWindow painter functions. PowerBuilder names objects it places in the generated DataWindow object (that is, columns, labels, and headings), but you need to name objects you place yourself if you want to refer to them anywhere.

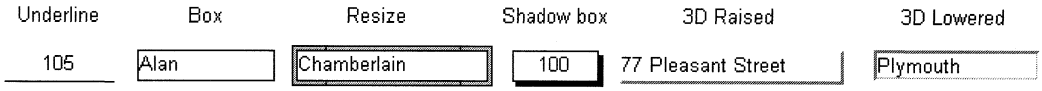
❖ To specify a name:

1 Select Name from the object's popup menu or double-click the object.

2 Type the name.

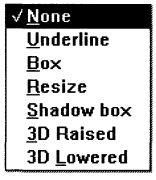
Using borders

You can place borders around text, columns, graphs, and crosstabs to enhance their appearance. PowerBuilder provides six types of borders.



If you specify the Resize border, users can resize the object during execution. Resize borders are particularly useful with graphs.

❖ To add a border to an object:



1 Select the objects you want to add a border to.

2 Select Border from one of the selected objects' popup menu.

The Border cascading menu displays.

3 Select the border you want.

PowerBuilder places the border around the selected objects.

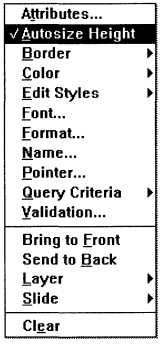
Specifying variable-height detail bands

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. You don't want to reserve space for that much information for the column in the detail band, since it would make the detail band's height very large, meaning the user could see few rows at a time.

Instead, you want the detail band to resize during execution, based on the data in the Memo column. If the Memo column has only one line of text, you want the detail band to be one line. If the Memo column has 20 lines of text, you want the detail band to be 20 lines high.

To provide a detail band that resizes as needed, you specify that the variable-length columns and the band have Autosize Height.

❖ To create a resizable detail band:



1 For each column that should resize based on the amount of data, select Autosize Height from the column's popup menu.

The menu item is now checked.

2 Turn off automatic horizontal scrolling for the resizable columns so that PowerBuilder wraps text during execution instead of displaying all text on one scrollable line:

1 Select Edit Styles➤Edit from the column's popup menu to open the Edit style dialog box.

2 Deselect the Auto H Scroll checkbox.

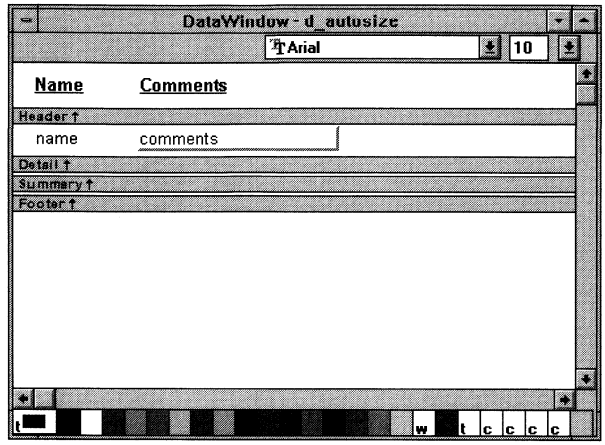
3 Position the mouse pointer on the bar that represents the detail band and select Autosize Height from the detail band's popup menu.

During execution, the detail band will resize based on the contents of the columns defined as having Autosize Height.

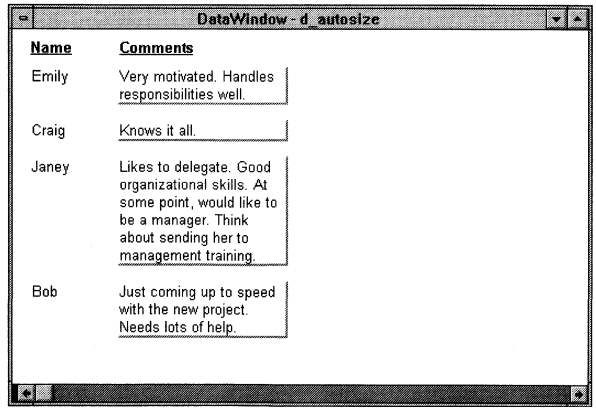
Tip
You can have Autosize Height columns without an Autosize Height detail band. If such a column expands beyond the size of the detail band during execution, it gets clipped.

Example

In the following DataWindow object, the Comments column is defined as having Autosize Height, as is the detail band. Horizontal scrolling is turned off for the Comments column.



During execution, the space allocated for a row depends on how much information is in Comments; the detail band expands to fit the text. In the executing DataWindow object below, the first row takes two lines, the second row takes one line, and so on.



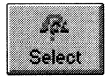
Modifying the data source

When modifying a DataWindow object, you might realize that you haven't included all the columns you need. Or you might need to define retrieval arguments. You can modify the data source from the DataWindow painter workspace. 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 Click the Select button in the PainterBar.
or
Select Design➤Edit 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.)

For more information, see "Defining the data source" in Chapter 13, "Defining DataWindow Objects."

2 Modify the SELECT statement graphically using the same techniques as when creating it. Select Convert to Syntax from the Options menu to modify the SELECT statement syntactically.

3 Click the Design button to return to the workspace.

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 "Controlling updates" on page 518.

Changing the table
If you change the table referenced in the SELECT statement, PowerBuilder maintains the columns in the workspace (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 Objects➤Retrieval Arguments from the menu bar.

The Specify Retrieval Arguments dialog box displays 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 13, "Defining DataWindow Objects."

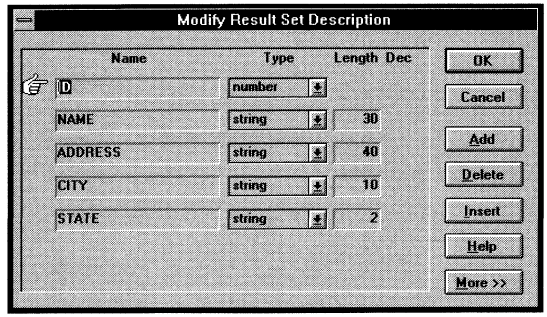
Modifying the result set

If the data source is External or Stored Procedure, you can modify the result set description.

❖ **To modify a result set:**

1 Click the Select button in the PainterBar.
or
Select Edit Data Source from the Design menu.

The Modify Result Set Description dialog box displays.



2 Review the description and make any necessary changes.

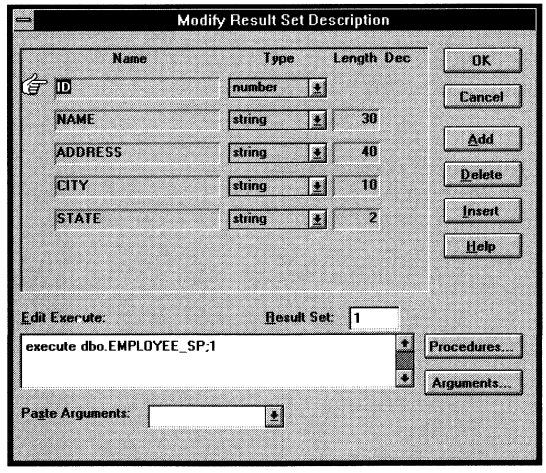
3 Click OK.

You return to the workspace.

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 Modify Result Set window contains a More button.

Click More to edit the Execute statement, select another stored procedure, or add retrieval arguments.



Reorganizing objects in the DataWindow

You can change the layout of any of the objects in a DataWindow object.

Displaying boundaries for objects

When reorganizing objects in the workspace, it is sometimes helpful to see how large all the objects are. That way, you can easily check for overlapping objects and make sure that the spacing around objects is what you want.

❖ To display object boundaries:

- Select Design➤Show Edges from the menu bar.

  PowerBuilder displays the boundaries of each object.

Boundaries are for screen display only
The boundaries displayed for objects are for use only in the painter workspace. They do not display during execution.

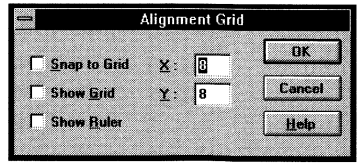
Using the grid and the ruler

The DataWindow painter provides a grid and a ruler to help you align objects.

❖ To use the grid and the ruler:

1 Select Design➤Grid/Ruler from the menu bar.

  The Alignment Grid dialog box displays.



2 Use the options to:

- Make objects snap to a grid position when you place them or move them in a DataWindow object.
- Show or hide the grid when the workspace displays.
- Specify the size (height and width) of the grid cells.
- Show a ruler. The ruler uses the units of measurement specified in the DataWindow Style window.

For more information, see "Changing the DataWindow object style" on page 478.

The options are:

Option	Meaning
Snap to Grid	If selected, controls snap to the grid when placed or moved
Show Grid	If selected, the grid is displayed in the workspace
X	The width of each cell in the grid in pixels
Y	The height of each cell in the grid in pixels
Show Ruler	If selected, the ruler is displayed in the workspace

Performance
Screen painting is slower when the grid is displayed, so you might want to display the grid only when necessary.

Your choices are saved

Your choices for the grid and the ruler are saved in the PB.INI file and used the next time you start PowerBuilder.

Deleting objects

❖ **To delete objects:**

1 Select the objects you want to delete.

2 Click the Delete button.
or
Select Edit➤Clear from the menu bar.
or
Press the DEL key.

All selected objects are deleted.

Moving objects

In all presentation styles except Grid

In all presentation styles except Grid, you can move all the objects (such as headings, labels, columns, drawing objects, and graphs) anywhere you want.

❖ To move objects:

1 Select the objects you want to move.

2 Do one of the following:

- Drag the objects with the mouse.
- Press an arrow key to move the objects in one direction.

In grid DataWindow objects

You can reorder columns in grid DataWindow objects when previewing.

See "Working in a grid DataWindow" on page 475.

Copying objects

You can copy objects within a DataWindow object and to other DataWindows. All attributes of the object are copied.

❖ To copy an object:

1 Select the object in the DataWindow painter workspace.

2 Select Edit➤Copy from the menu bar.
or
Press CTRL+C.

The object is copied to a private PowerBuilder clipboard.

3 To copy the object within the same DataWindow, select Edit➤Paste from the menu bar or press CTRL+V. To copy the object to another DataWindow object, open another instance of the DataWindow painter and open the desired DataWindow object in it. Make that DataWindow active and paste the object.

PowerBuilder pastes the object at the same location as in the source DataWindow object (if you are pasting into the same DataWindow, you should move the pasted object so it doesn't cover the original object). PowerBuilder displays a message box if the object you are pasting is not valid in the destination DataWindow object.

Resizing objects

You can resize an object using the mouse or the keyboard.

Using the mouse

To resize an object, select it, then grab an edge and drag it with the mouse.

Using the keyboard

To resize an object, select it, then do the following:

To make the object	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:**

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 objects

Often you want to align several objects or make them all the same size. You can use the grid to align the objects or have PowerBuilder align them for you.

❖ **To align objects:**

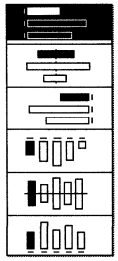
1. Select the object whose position you want to use to align the others.

   PowerBuilder displays handles around the selected object.

2. Extend the selection by pressing and holding the CTRL key and clicking the objects you want to align with the first one.

   All the objects have handles on them.

3 Select Edit➤Align Objects from the menu bar.



4 Pull the menu right and select the dimension along which you want to align the controls. For example, to align the objects along the left side, select the first choice on the cascading menu.

PowerBuilder moves all the selected objects to align with the first one.

Equalizing the space between objects

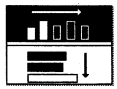
If you have a series of objects and the spacing is fine between two of them but the spacing is wrong for the rest, you can easily equalize the spacing around all the objects.

❖ To equalize the space between objects:

1 Select the two objects whose spacing is correct (click one object, then press CTRL and click the second object).

2 Select the other objects whose spacing you want to be the same as the first two objects by pressing CTRL and clicking.

3 Select Edit➤Space Objects from the menu bar.



4 Pull the menu right and select the dimension whose spacing you want to equalize.

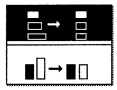
Equalizing the size of objects

Say you have several objects in a DataWindow object and want their sizes to be the same. You can accomplish this manually or by using the Edit menu.

❖ To equalize the size of objects:

1 Select the object whose size is correct.

2 Select the other objects whose size you want to match the first object by pressing CTRL and clicking.



3 Select Edit➤Size Objects from the menu bar and pull the menu right to select the dimension whose size you want to equalize.

Sliding objects to remove blank space

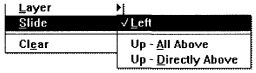
You can specify that you want to eliminate blank lines or spaces in a DataWindow object at execution time by sliding columns and other DataWindow objects to the left or up if there is blank space. You can use this feature to remove blank lines in mailing labels or to remove extra spaces between fields (such as first name 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.

For more information, see Chapter 17, "Using Nested Reports."

❖ To use sliding columns or objects:

1 Display the popup menu for the column or object you want to slide.



2 Select Slide and choose the option you want.

Option	Description
Left	Slide the column or object to the left if there is nothing to the left
Up -All Above	Slide the column or object up if there is nothing in the row above (the row above must be completely empty for the column or object to slide up)
Up - Directly Above	Slide the column or object up if there is nothing *directly above it* in the row above

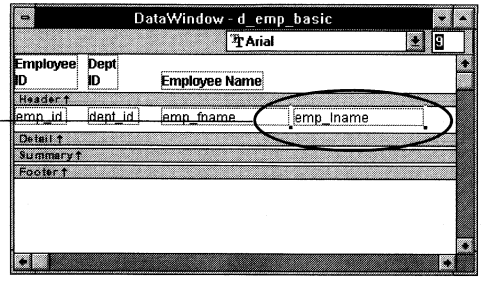
If you are sliding columns up
Even blank columns have height, so if you want columns to slide up, you need to specify as Autosize Height all columns above that might be blank and that you want to slide other columns up through.

Example

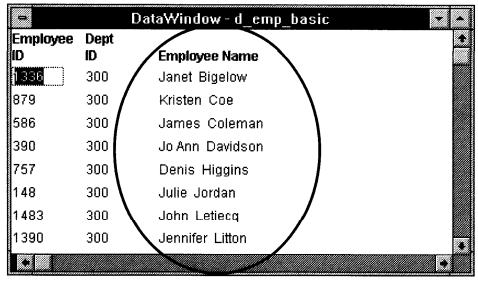
In a tabular or label DataWindow that includes first and last names, you might want to specify the last name column as slide left so it displays next to the first name (be sure to leave some space between the two columns so there is space between the names after sliding).

In the following DataWindow, Emp_Lname is specified as slide left (edges are shown in the screen to indicate the spacing between the columns):

Emp_Lname is defined as slide left



During execution, the last names slide left to remove the blank space:



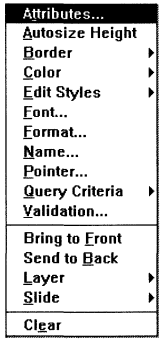
Conditionally modifying attributes at execution time

You define the appearance and behavior of a DataWindow in the DataWindow painter. As you do that, you are specifying the DataWindow object's attributes. For example, when you place a border around a column in the painter workspace, you are setting that column's Border attribute.

In most cases, the appearance and behavior are fixed; you don't want them to change in a running application. But you might want some attributes of the DataWindow (such as the use of borders and colors) to be driven by the data, which is not known when you are defining the DataWindow in the painter. For these situations, you can define **attribute conditional expressions**, which are expressions that are evaluated at execution time. The results of the expressions set the values of attributes of objects in the DataWindow. You can use these attribute conditional expressions to conditionally modify the appearance and behavior of your DataWindow during execution.

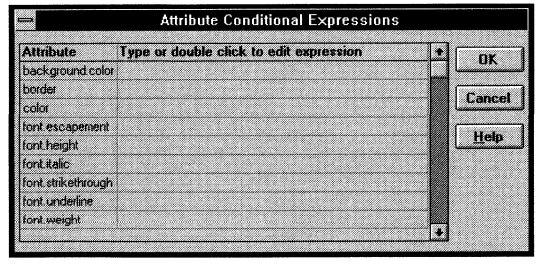
❖ To conditionally modify attributes at execution time:

1 Display the popup menu for the object whose attributes you want to conditionally modify. You can do this for bands and for any object in the DataWindow (columns, graphs, drawing objects, and so on).



2 Select Attributes from the popup menu.

The Attribute Conditional Expressions dialog box displays the attributes that you can conditionally set for the selected object.



3 Type an expression that will be evaluated at execution time.
or
Double-click to enter the expression in the Modify Expression dialog box (you can paste functions into the expression in the Modify Expression dialog box).

4 Specify as many attribute conditional expressions as you want.

5 Click OK.

At execution time, the expressions will be evaluated and the corresponding attributes will be assigned values based on the evaluations.

Another way
You can also programmatically modify the attributes of a DataWindow object in a script during execution using the Modify function.

For more information, see the discussion of Modify in the *Function Reference* and the discussion of dynamic DataWindows in *Building Applications*.

Specifying the attribute conditional expression

In nearly all attribute conditional expressions, you will use the If function to test for one or more conditions. If the condition is true, one value is set for the attribute. If the condition is false, another value is set for the attribute.

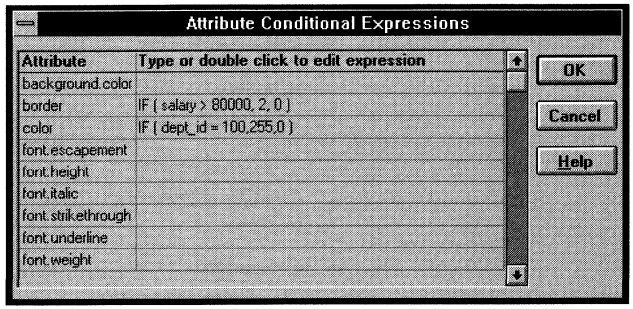
For complete information about what the valid values are for the various DataWindow object attributes, see the discussion of DataWindow object attributes in the *Function Reference* or online Help.

See the next section for examples.

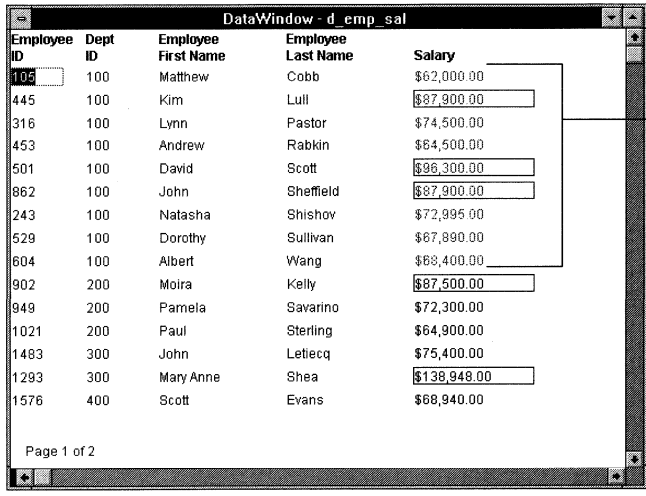
Examples

The following dialog box, which applies to the Salary column in a DataWindow showing employee information, specifies that:

- If Salary > $80,000, then the value should display with a rectangular border (the column's Border attribute equals 2, which means rectangular border).
- If the Department ID is 100, then the salary should display in red (the column's Color attribute equals 255, which means red).



Here is the resulting DataWindow at execution time:



Displayed in red because Dept ID is 100

The boxed salaries are greater than $80,000

Here are more examples of using attribute expressions to conditionally define attributes of a DataWindow object:

Attribute	Expression	Result
Protect for a column	If (IsRowNew( ), 0 , 1)	Protect the column except for new rows (user can only modify new data)
Color for a column	If (IsRowNew(), 255, 0)	Display value in red for new rows only
Visible for a picture object	If (State = "TX", 1, 0)	Display the picture in rows only where State is Texas

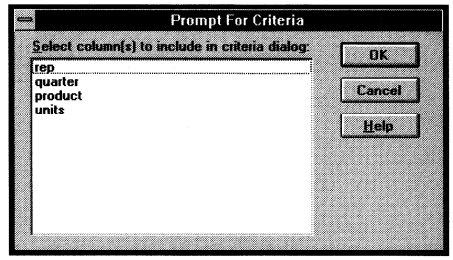
Prompting for retrieval criteria

You can define your DataWindow object so that it always prompts the user for retrieval criteria just before it retrieves data.

❖ To prompt for retrieval criteria:

1 Select Rows➤Prompt for Criteria from the menu bar.

The Prompt For Criteria dialog box displays listing all columns in the DataWindow.



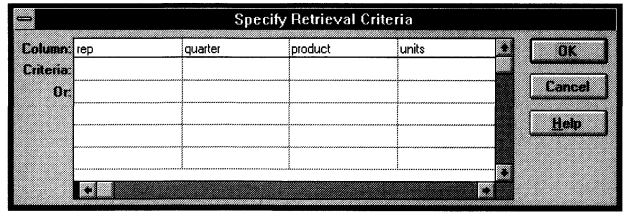
2 Select the columns you want the user to be able to specify retrieval criteria for during execution.

Selected columns are highlighted.

3 Click OK.

What happens

When you specify prompting for criteria, during execution PowerBuilder displays the following dialog box to the user 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 Prompt for Criteria dialog box displays in the grid. Users can specify criteria here exactly as you do in the grid with the Quick Select DataWindow data source.

Criteria specified here are added to the WHERE clause for the SQL SELECT statement defined for the DataWindow object.

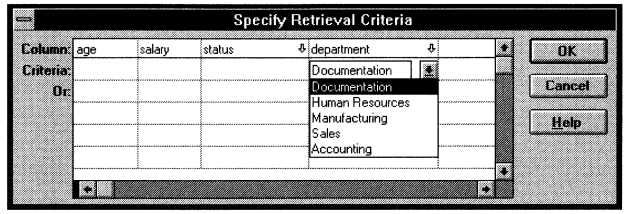
Testing in PowerBuilder

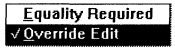
You can test the prompting for criteria by previewing the DataWindow. Note that by default, after doing the first retrieval, PowerBuilder stores the data internally (caches it) and doesn't re-retrieve it the next time you go to preview. So you won't be prompted for criteria each time you go to preview. You can explicitly retrieve by clicking the Retrieve button.

For more about data caching, see page 459.

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 dropdown listbox during execution.





If you don't want the dropdown listbox used for a column when the user specifies retrieval criteria, select Query Criteria➤Override Edit from the column's popup menu in the DataWindow painter workspace. Doing so provides a standard edit control for the column.

Forcing users to enter criteria

If you have specified prompting for criteria for a column, you can force users to enter criteria for the column by selecting Query Criteria➤Equality Required from the column's popup menu in the workspace; PowerBuilder will underline the column header in the grid during execution. Users have to specify selection criteria for the specified column, and they must use the = operator.

For more information

The section "Using Quick Select" in Chapter 13, "Defining DataWindow Objects," describes in detail how you and your users can specify selection criteria in the grid.

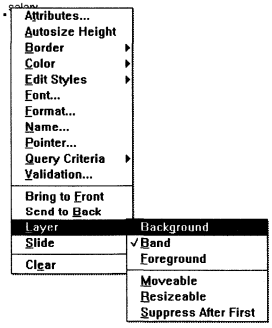
The chapter on dynamic DataWindow objects in *Building Applications* describes how to write scripts to dynamically allow users to specify retrieval criteria during execution.

Adding objects

You can add the following objects to enhance your DataWindow object:

- Columns
- Text
- Drawing objects
- Pictures (bitmaps)
- Computed fields
- Graphs
- OLE 1.0 binary large database objects (blobs)
- Reports

About the Layer attribute



Each object has a specified **layering**—how it is positioned within the DataWindow object. An object's layering is specified in the Layer cascading menu in the object's popup menu:

Attribute	Meaning
Background	Object is behind other objects. It is not restricted to one band.
Band	Object is placed within one band. It cannot extend beyond the band's border.
Foreground	Object is in front of other objects. It is not restricted to one band.
Moveable	Object can be moved during execution.
Resizeable	Object can be resized during execution.
Suppress After First	Don't repeat the object after the first column in a newspaper-style DataWindow. For more about newspaper columns, see *Building Applications*.

The default layering

PowerBuilder uses the following defaults when placing objects:

Object	Default layering
Graph	Foreground, movable, resizable
All other objects	Band, not movable, not resizable

Changing an object's layering

Usually the defaults chosen by PowerBuilder are what you want. But by changing selections on the Layer menu, you can change the layering for an object. For example, if you want a graph to be fixed in the detail band, you can change the layering to Band and deselect Moveable and Resizeable.

Adding columns

You can add columns to a DataWindow object. You can restore columns you have deleted, or have columns display more than once in the DataWindow.

❖ **To add a column:**

1 Click the Column button.
or
Select Objects➤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 DataWindow object.

3 Select the column and click OK.

Adding text

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:

1 Click the Text button in the PainterBar.
or
select Text from the Objects menu.

2 Click where you want the text.

PowerBuilder places the text object in the workspace and displays the word *text*.

3 Type the text in the text box at the top of the workspace.

4 (Optional) Change the font, size, style, and alignment for the text using the StyleBar.

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 attributes for any text in a DataWindow object.

For more about changing the application's default text font and style, see Chapter 2, "Working with Applications."

Adding drawing objects

You can add the following drawing objects to a DataWindow object to enhance its appearance:

- Rectangle
- RoundRectangle
- Line
- Oval

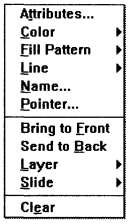
Drawing objects are useful in grouping objects in a DataWindow object or providing design highlights. For example, you can place a colored rectangle behind a group of objects to group them.

❖ To place a drawing object:

1 Select the drawing object from the PowerBar or from the Objects menu.

2 Click where you want the object to display.

3 Resize or move the drawing object as needed.



4 Use the drawing object's popup menu to change its attributes as needed. For example, you might want to specify a fill color for a rectangle or thickness for a line.

Adding pictures

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 contents of that band displays. If you place the picture in the detail band of the DataWindow object, it displays in each row.

Tips for using pictures

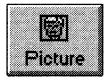
To display a different picture for each row of data, retrieve a column containing picture filenames from the database.

For more information, see the section on character columns in "Specifying extended column attributes" in Chapter 12, "Managing the Database."

To compute a picture name during execution, use the Bitmap function in the expression defining a computed field. To use a picture to indicate that a row has focus during execution, use the SetRowFocusIndicator function.

For information about these functions, see the *Function Reference*.

❖ To place a picture in a DataWindow object:



1 Click the Picture button in the PainterBar.
or
Select Objects➤Picture from the menu bar.

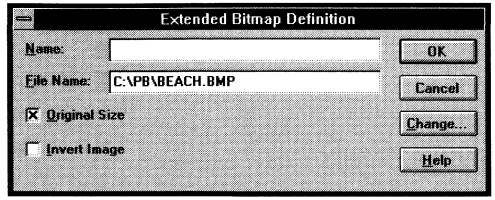
2 Click where you want the picture to display.

The Select a File Name dialog box displays a list of bitmap (BMP) files in the current directory.

3 Select a picture from the list or enter a filename in the File Name box. The picture must be a bitmap (BMP), runlength-encoded (RLE), or Windows metafile (WMF) file. To change the current directory, select another directory from the list.

4 Click OK.

The Extended Bitmap Definition dialog box displays.



5 Name the picture by typing a name in the Name box.

6 Click the Original Size checkbox to display the bitmap in its original size. You can use the mouse to change the size of the bitmap in the DataWindow painter.

7 Click the Invert Image checkbox to display the picture with its colors inverted.

8 Click the Change button to specify a different picture.

9 Click OK.

You return to the workspace with the picture in place.

Adding computed fields

You can use **computed fields** in any band of the DataWindow object. Typical uses of computed fields include:

- Calculations based on column data that change for each retrieved row. For example, if you are retrieving yearly salary, you can define a computed field in the detail band that displays monthly salary (defined as Salary / 12).
- Summary statistics of the data. For example, if you have a grouped DataWindow object, you can use a computed field to calculate the totals of a column for each group.

- Concatenated fields. For example, if you are retrieving first name and last name, you can define a computed field that concatenates the values so they appear with only one space between them (defined as Fname + " " + Lname).
- System information. For example, you can place the current date and time in a DataWindow object's header by using computed fields (defined as Today() and Now()).

About defining computed columns and 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 workspace, 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 as part of the SELECT statement. 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 workspace, 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 these columns in a DataWindow object. 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 part 101 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. So the user gets this display, with the incorrect cost:

Part #	Quantity	Price	Cost
101	100	2.50	**125.00**

On the other hand, if Cost is defined as a computed field in the DataWindow painter workspace, the SELECT statement looks like the following statement. It does *not* have a computed column:

```
SELECT part.part_num,
   part.part_qty,
   part.part_price,
FROM part;
```

The computed field is defined in the workspace as Quantity * Price.

In this scenario, if the user changes the price or part 101 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 don't care about dynamically updating the computed values, define the computed column as part of the SELECT statement.

If you want computed values to change dynamically, define your computed fields in the DataWindow painter workspace, as described next.

Defining a computed field

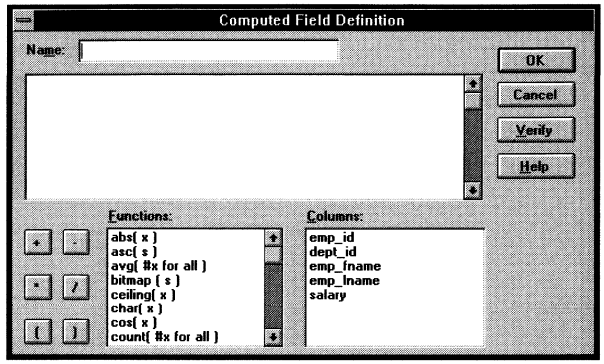
❖ To define a computed field in the workspace:

1 Click the Compute button in the PainterBar.
or
Select Objects➤Computed Field from the menu bar.

2 Click where you want 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 Computed Field Definition dialog box displays and lists:

- Built-in PowerScript functions you can use in the computed field
- The columns in the DataWindow object
- Operators and parentheses



3 Name the computed field.

4 Enter the expression that defines the computed field (see below).

5 (Optional) Click Verify to test the expression.

PowerBuilder analyzes the expression.

6 Click OK.

PowerBuilder returns you to the workspace with the computed field in place.

Entering the expression

You can enter any valid PowerScript expression when defining a computed field. You can paste operators, columns, and PowerScript functions into the expression from information in the Computed Field Definition window.

> **Expression is PowerScript**
> The expression you are entering is a *PowerScript* expression; it is not a SQL expression processed by the DBMS. So the expression follows PowerScript rules.
>
> For complete information about PowerScript expressions, see *PowerScript Language*.

You can use any non-object-level PowerScript function (built-in or user-defined) in an expression.

You can use the + operator to concatenate strings.

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

Here are some examples of computed fields and columns:

To display	Enter this expression	In this band
Current date at top of each page	Today( ) (a built-in PowerScript function)	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
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 more information

For complete information about the PowerScript functions that you can use in computed fields in the DataWindow painter, see the *Function Reference*.

A shortcut for doing summary statistics

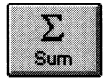
PowerBuilder provides a quick way to create computed fields that summarize values in the detail band.

❖ To summarize values:

1 Select one or more columns in the DataWindow object's detail band.

2 Do one of the following:

To place this computed field	Do this
Average	Select Objects➤Average - Computed Field from the menu bar.
Count	Select Objects➤Count- Computed Field from the menu bar.
Sum	Click the Sum button or select Objects➤Sum - Computed Field from the menu bar.



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.

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:

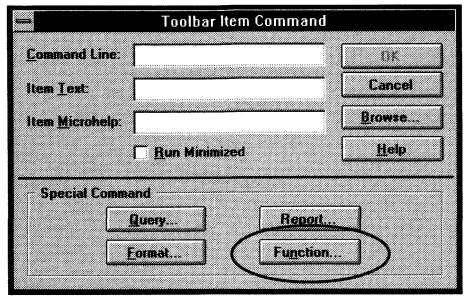
1 Place the mouse pointer over the PainterBar and select Customize from the popup menu.

The Customize dialog box displays.

2 Click Custom in the Select palette group to display the set of custom buttons.

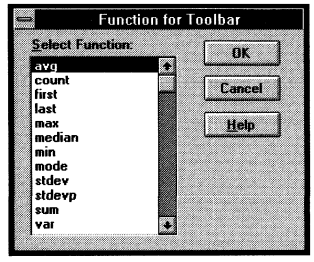
3 Drag a custom button into the Current toolbar group and release it.

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.

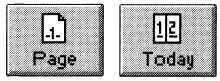
You return to the Toolbar Item Command dialog box.

6 Specify text 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.

A shortcut for placing page numbers and date

You can click buttons in the PainterBar to place a computed field for the current page number and date anywhere in the DataWindow object.



To place this computed field	Do this
'Page ' + page() + ' of ' + pageCount()	Click the Page button or select Objects➤Page n of n Computed Field from the menu bar
today()	Click the Today button or select Objects➤Today() - Computed Field from the menu bar

Adding graphs

Graphs are one of the best ways to present information. For example, if your application displays sales information over the course of a year, instead of displaying rows and columns of data that is difficult to interpret, 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 18, "Working with Graphs."

Adding OLE 1.0 blob objects

You can add a column to a DataWindow that contains a database binary large object (a blob object) using OLE 1.0.

For information on using blob columns, see the chapter on OLE in *Building Applications*.

Adding reports

You can nest reports (nonupdatable DataWindow objects) in a DataWindow object.

For information on nesting reports, see Chapter 17, "Using Nested Reports."

Storing data in a DataWindow object

Usually you retrieve data into a DataWindow object during execution—the data is changeable and you want the latest. But sometimes the data you display in a DataWindow object is static—it never changes. In these situations, you might want to store the data in the DataWindow object itself. That way, you don't need to go out to the database or other data source to display the data during execution.

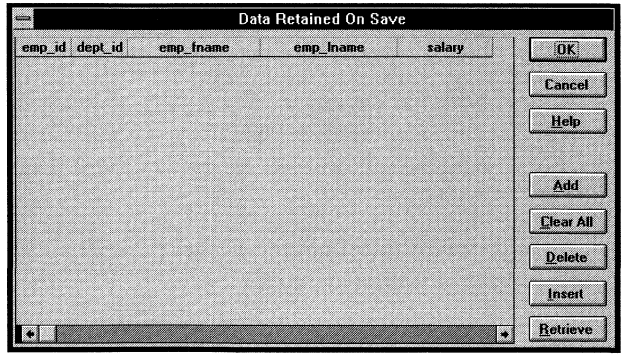
The most common reason to store data in a DataWindow object is for use as a dropdown DataWindow where the data is not coming from a database. For example, you might want to display a list of state postal codes for a user entering values in a State column in a DataWindow object. You can store those codes in a DataWindow and use the dropdown DataWindow edit style for the State column.

For more information about using the dropdown DataWindow edit style, see Chapter 15, "Displaying and Validating Data."

❖ To store data in a DataWindow object:

1 Select Rows➤Data from the menu bar.

The Data Retained On Save dialog box displays. All columns defined for the DataWindow object are listed at the top.



2 Do one of the following:

- Click Add to create an empty row and type a row of data into the window. You can enter as many rows as you want.

- Click Retrieve to retrieve all the rows of data from the database. If you want, you can delete rows you don't want to save or manually add new 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.

3 Click OK.

PowerBuilder saves the rows.

When you save the DataWindow object, the data is stored in the DataWindow object in the PowerBuilder library.

> **Sharing DataWindow 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 computer to get the data and all its attributes.

What happens during execution

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.

With one exception, if you *do* issue Retrieve for a DataWindow stored with data, PowerBuilder handles it the same as a DataWindow that is not stored with data: PowerBuilder gets the latest data by retrieving rows from the database.

The exception

PowerBuilder never retrieves data into a dropdown DataWindow that already contains data.

Retrieving only as many rows as needed

If your DataWindow object retrieves an enormous number of rows, there can be a noticeable delay during execution while all the rows are retrieved and before control returns to the user. In these DataWindow objects, you can specify that PowerBuilder retrieves only as many rows as it has to before displaying data and returning control to the user.

For example, if your DataWindow displays only 10 rows at a time, it might make sense to have PowerBuilder retrieve only a small number of 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 the pauses might be worth it if the user doesn't have to wait 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 Only 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.

Limitations

Retrieve Only 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.)

Controlling updates

When PowerBuilder generates the basic DataWindow object, it defines whether the data is updatable by default as follows:

- If the DataWindow 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 the user to tab to the columns.
- If the DataWindow 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 function to write your own update script.

For more information, see the *Function 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 the user modifies the values in a key column

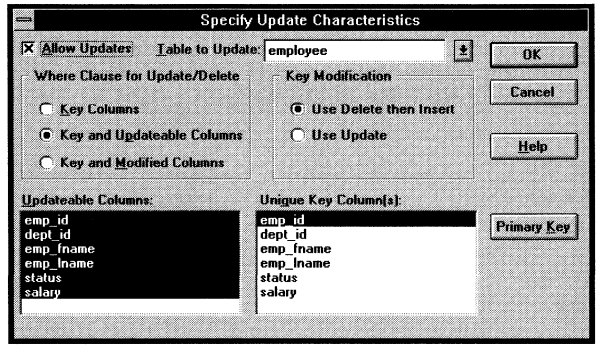
Updatability of views
Some views are logically updatable; some are not.

For the rules your DBMS follows regarding updating of views, see your DBMS's documentation.

❖ To specify update characteristics for a DataWindow object:

1 Select Rows➤Update from the menu bar.

The Specify Update Characteristics dialog box displays.



2 To prevent updates to the data, make sure the Allow Updates box is not selected. Click OK to return to the workspace.

To allow updates, select the Allow Updates box and specify the other settings as described below.

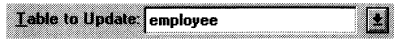
3 Click OK to return to the workspace.

Changing tab values

PowerBuilder does not change the tab values associated with columns after you change the update characteristics of the DataWindow object. So if you have allowed updates to a table in a multitable DataWindow object, you should change the tab values for the updatable columns so users can tab to them.

For more information, see "Defining the tab order" on page 483.

Specifying the table to update

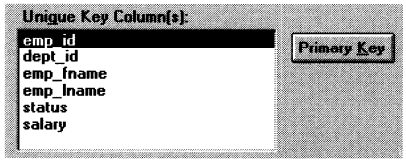


Each DataWindow object can update one table.

❖ To specify the table that can be updated:

- Select the table from the Table to Update box.

Specifying the unique key columns

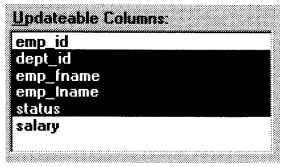


The Unique Key Columns box specifies which columns PowerBuilder uses to identify a row being updated. PowerBuilder uses the column or columns you specify here as the key columns when generating the WHERE clause to update the database, as described below.

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 updatable columns



You can specify that all or some of the columns in a table are updatable.

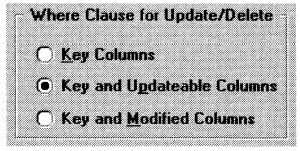
Updatable columns are displayed highlighted. In the preceding screen, Dept_ID, Emp_Fname, Emp_Lname, and Status are updatable. The other columns are not.

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 in the workspace. For example, if you have allowed a column to be updated, you should change its tab value to a nonzero number so the user 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...*	DELETE
SET *column = newvalue*	FROM *table*
WHERE *col1 = value1*	**WHERE *col1 = value1***
AND *col2 = value2 ...*	**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 following in the Where Clause for Update/Delete box. The results are illustrated by an example following the table:

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, if 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 Updateable 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.

Option	Result
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. Say your 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
```

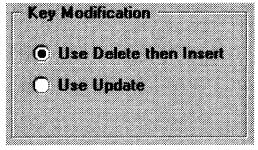
  Here the UPDATE statement is also checking the original value of the modified column in the WHERE clause. The statement will fail if another user changed the salary of employee 1001 since your application retrieved the row.

- If you choose Key and Updateable 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 attribute 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 will fail 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, you should select Use Delete then Insert.
- If only one row can be modified by the user before the database is updated, use UPDATE; it is faster.

CHAPTER 15

Displaying and Validating Data

About this chapter

This chapter describes how to specify how you want values for columns to be displayed and validated in DataWindow objects.

Contents

Overview

You can specify how to display the values for each column and validate data entered by users in a DataWindow object. You do this by defining display formats, edit styles, and validation rules for columns.

When PowerBuilder generates a basic DataWindow object given a data source and presentation style, it uses the extended attributes defined for a column in the Database painter and stored in the repository. These definitions can include a specification for a column's display format, edit style, and validation rule.

You can override the defaults for columns in any particular DataWindow object. The changes you make in the DataWindow painter do not change the information stored with the column definition in the repository.

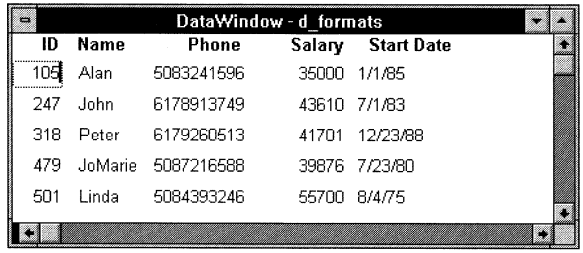
This chapter describes:

- Display formats
- Edit styles
- Validation rules

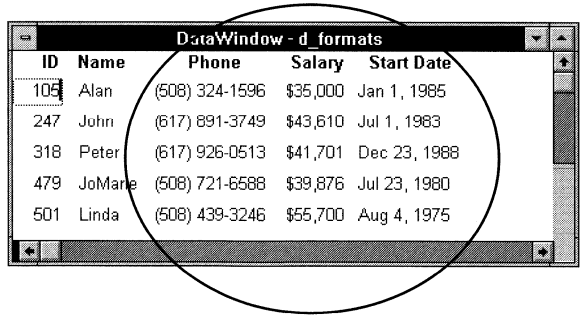
Working with display formats

You can use display formats to customize the display of column data in a DataWindow object. 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 is a DataWindow without any display formats; all values display as they are stored in the database:



Here the Phone, Salary, and StartDate columns are using display formats so the data is easier to interpret:



Display formats not used for data entry

When a user tabs to a column containing a display format, PowerBuilder removes the display format and displays the raw value for the user to edit.

If you want to provide formatting used for data entry, specify edit masks, as described in "Working with edit styles" on page 541.

Using 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 formats are stored in the repository. Once you define a display format, it can be used by any column of the appropriate data type in the database.

- Assign display formats to 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 repository
- Create an ad hoc, unnamed format to use with one specific column

Coupling between Database and DataWindow painters

Once 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 repository.

If the definition of the display format later changes in the repository, the format for the column in a DataWindow object will not change. If you want to use the modified format, you can reapply it to the column in the DataWindow painter.

Working 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.

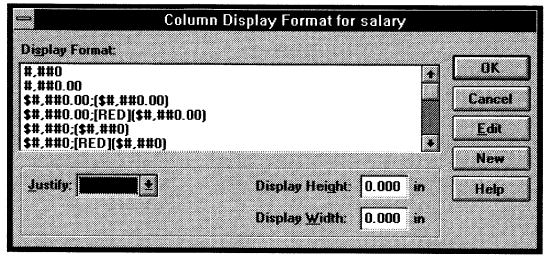
❖ To create or modify a display format:

1 In the Database painter, open the table containing the column you want to apply a display format to.

2 Position the mouse over the column and click the right mouse button to display the column's popup menu.

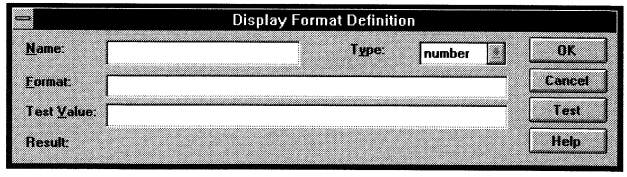
3 Select Display from the popup menu.

A dialog box displays listing all defined display formats for the corresponding data type.



4 Click New to create a new format, or click Edit to modify an existing one.

The Display Format Definition dialog box displays.



5 If creating a new format, name it.

6 Define the display format using masks, as described in "Defining display formats" on page 533.

7 If you want, you can enter a test value and click the Test button to test the format.

8 Click OK.

You return to the dialog box. The new or modified display format is highlighted.

9 Click OK.

You return to the workspace. The new or modified format is stored in the repository and applied to the column.

You can use this format with any column of the appropriate data type in the database.

Another way to create and modify a display format
You can also create and modify display formats by selecting Objects➤Display Format Maintenance from the menu bar.

The only difference is that when you create a display format this way, you must specify the appropriate data type in the Type dropdown listbox (because you are not creating the display format for a specific column, you need to tell PowerBuilder what type of column the display format is meant for).

❖ To apply an existing display format to a column:

1 Open the table containing the column you want to apply a display format to.

2 Position the mouse over the column and click the right mouse button to display the column's popup menu.

3 Select Display from the popup menu.

A dialog box displays listing all defined display formats for the corresponding data type.

4 Select a format.

5 Click OK.

The column now has the selected format associated with it in the repository.

Another way to apply display formats
You can also apply display formats by selecting Definition from a table's popup menu to display the Alter Table dialog box. Then select a column and pick a display format from the Format dropdown listbox in the Extended Attributes section at the bottom of the dialog box.

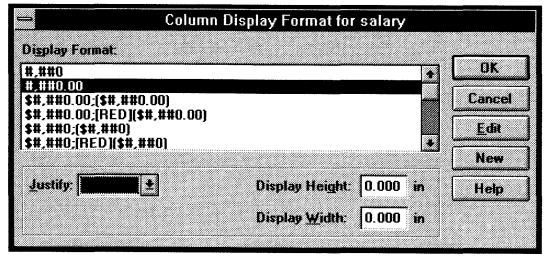
❖ To remove a display format from a column:

1 Open the table containing the column you want to apply a display format to.

2 Position the mouse over the column and click the right mouse button to display the column's popup menu.

3 Select Display from the popup menu.

A dialog box displays listing all defined display formats for the corresponding data type. The currently used format is highlighted.



4 Click the highlighted format.

PowerBuilder removes the highlighting.

5 Click OK.

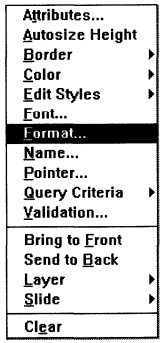
You return to the workspace. The display format is no longer associated with the column.

Working 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 repository or defining an ad hoc format for one specific column.

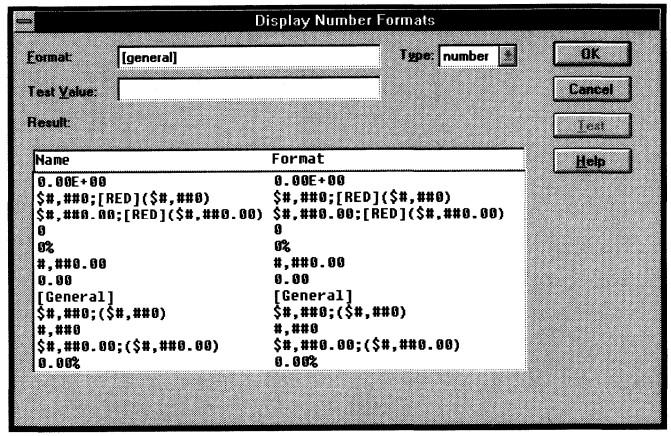
About computed fields
You can assign display formats to computed fields using the same techniques as for columns in a table.

❖ To specify a display format for a column:



1 In the DataWindow painter, select Format from the column's popup menu.

A dialog box appropriate to the data type of the selected column displays. The currently used format displays in the Format box. All formats for the data type defined in the repository are listed at the bottom. Here is the dialog box for numeric columns:



2 Do one of the following:

- Select a format in the repository from the list at the bottom of the dialog box.
- Create a new, ad hoc format for the column by typing it in the Format box. Use masks as described in "Defining display formats" on page 533.

> **Format not saved in repository**
> If you create a format here, it is used only for the current column. The format is not saved in the repository for use with other columns.

3 (Optional) Test the format by typing a value in the Test Value box and clicking Test.

4 Click OK.

Shortcuts
To assign the currency or percent display format to a numeric column, select the column, then click the Currency or Percent button in the PainterBar or select Edit➤Format from the menu bar and choose the format from the cascading menu.

Customizing the toolbar

You can add buttons to the PainterBar that assign a specified display format to selected columns.

For more information, see "Customizing toolbars" in Chapter 1, "The World of PowerBuilder."

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. So you can use the following mask to display phone numbers (this is the mask used for Phone in the DataWindow on page 527):

```
(@@@) @@@-@@@@
```

Using sections

Each type of display format can have two or more sections, with each section corresponding to a form of the number, string, date, or time. All sections except the first in a format are optional. When you define a display format with more than one section, separate the sections with semicolons (;).

For example, number display formats can have four sections: the format for positive numbers, the format for negative numbers, the format to represent zero, and the format to represent NULL. The following format specifies different displays for positive and negative numbers (negative numbers should be in parentheses):

```
$#,##0;($#,##0)
```

Combining types

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 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 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 (\).

Using colors

You can specify a color for each section of the display format 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]	[MAGENTA]
[BLUE]	[RED]
[CYAN]	[WHITE]
[GREEN]	[YELLOW]

The formula for combining color values into a number is:

256*256**blue** + 256***green*** + ***red***

where the amount of each primary color is specified as a value from 0 to 255.

For example:

Number	Color
255	Red (no green or blue component)
256*255 = 65280	Green (no blue or red component)
256*128 = 32768	Dark green (no blue or red component)

Number	Color
256*256*255 = 16711680	Blue (no green or red component)
255 + 256*255 = 65535	Yellow (combination of red and green)
128 + 256*128 = 32896	Brown (combination of red and green)
256*256*255+256*255 = 16776960	Cyan (combination of green and blue)
256*256*192 + 256*192 + 192 = 12632256	Light gray (combination of all primaries)

Setting display formats during execution

In scripts, you can use the GetFormat function to get the current format for a column and the SetFormat function to change the format for a column during execution.

For more about the GetFormat and SetFormat functions, see the *Function Reference.*

Number display formats

A number display format can have up to four masks, with only the first being required.

```
positive-format;negative-format;zero-format;null-format
```

Special characters

Characters that have special meaning in number display formats are:

Character	Meaning
#	A number
0	A required number; a number will display for every 0 in the mask

Dollar signs, percent signs, decimal points, parentheses, and spaces display as entered in the mask.

International considerations

So that an application you build will run the same no matter in which country it is deployed, masks (used in display formats and edit masks) and DataWindow expressions require U.S. notation for numbers. That is, when you specify a number in a DataWindow expression or in a number mask, comma always represents the thousands delimiter and period always represents the decimal place.

During execution, the locally correct symbols are displayed for numbers. For example, in countries where comma represents the decimal place and period represents thousands, users will see numbers in those formats during execution.

Number keyword

You can use the following keyword as a number display format:

Keyword	Meaning
[General]	PowerBuilder determines an appropriate format to use

Examples

The following table shows how the values 5, –5, and .5 display when different format masks are applied:

Sample format	5	-5	.5
[General]	5	-5	0.5
0	5	-5	1
0.00	5.00	-5.00	0.05
#,##0	5	-5	1
#,##0.00	5.00	-5.00	0.50
$#,##0;($#,##0)	$5	($5)	$1
$#,##0;-$#,##0	$5	-$5	$1
$#,##0;[RED]($#,##0)	$5	($5)	$1
$#,##0.00;($#,##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

Characters that have special meaning in date display formats are:

Character	Meaning
d	Day number with no leading zero (for example, 9)
dd	Day number with leading zero if appropriate (for example, 09)
ddd	Day name abbreviation (Sun, Mon, and so on)
dddd	Day name (Sunday, Monday, and so on)
m	Month number with no leading zero
mm	Month number with leading zero if appropriate
mmm	Month name abbreviation (Jan, Feb, and so on)
mmmm	Month name (January, February, and so on)
yy	Two-digit year
yyyy	Four-digit year

About two-digit years
If a user specifies a two-digit year in a DataWindow, 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 a date display format:

Keyword	Meaning
[General]	The short date format specified by the user in the Microsoft Windows Control Panel
[LongDate]	The long date format specified by the user in the Microsoft Windows Control Panel
[ShortDate]	The short date format specified by the user in the Microsoft Windows Control Panel

Examples

The following table shows how the date Friday, Jan. 30, 1998, displays when different format masks are applied:

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

Characters that have special meaning in time display formats are:

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 if appropriate (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	AM or PM as appropriate.
am/pm	am or pm as appropriate.
A/P	A or P as appropriate.
a/p	a or p as appropriate.

Colons, slashes, and spaces display as entered in the mask.

24-hour times
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:

Keyword	Meaning
[Time]	The time format specified by the user in the Microsoft Windows Control Panel

Examples

The following table shows how the time 9:45:33:234567 PM displays when different format masks are applied:

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

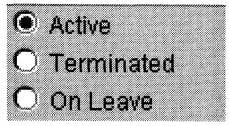
Working with edit styles

You can define edit styles for columns. Edit styles specify how column data is presented in DataWindow objects. Unlike display formats, edit styles don't only affect the display of data; they also affect how the user interacts with the data during execution.

You can choose from the following edit styles:

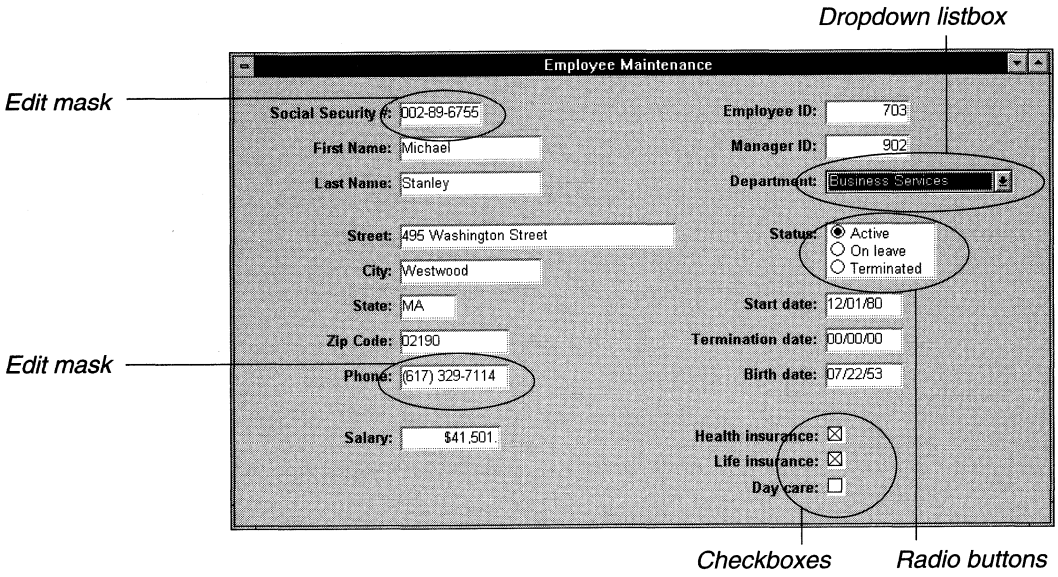
- An Edit box in which users can type values (the default)
- A DropDownListBox, in which users can select or enter a value
- A CheckBox that has specific values when it is on, off, or in the third state
- RadioButtons that display options users can select
- An Edit Mask that specifies allowable characters
- A DropDownDataWindow, in which users can select a value that comes from another DataWindow object

For example, if you have a column Status, which takes one of three values, A, T, or L, you might assign it the RadioButton edit style.



The Status data will be presented as radio buttons. Your user can simply click a button instead of having to type A, T, or L (and you don't have to create a validation rule to validate the typed input).

Here's a DataWindow that uses five edit styles.



Using edit styles

You work with edit styles 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 edit styles

  The edit styles are stored in the repository. Once you define an edit style, it can be used by any column of the appropriate data type 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 repository
- Create an ad hoc, unnamed edit style to use with one specific column

Coupling between Database and DataWindow painters

Once 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 repository, 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 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.

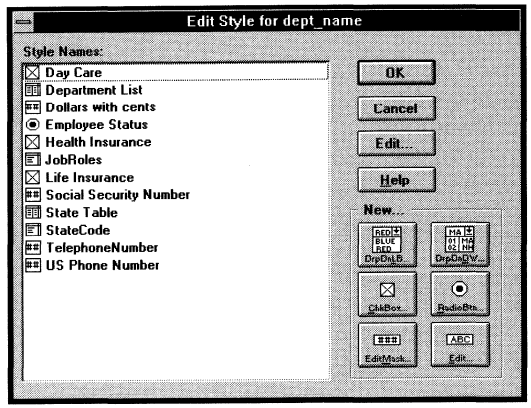
❖ To create or modify an edit style:

1 In the Database painter, open the table containing the column you want to apply an edit style to.

2 Position the mouse over the column and display the column's popup menu.

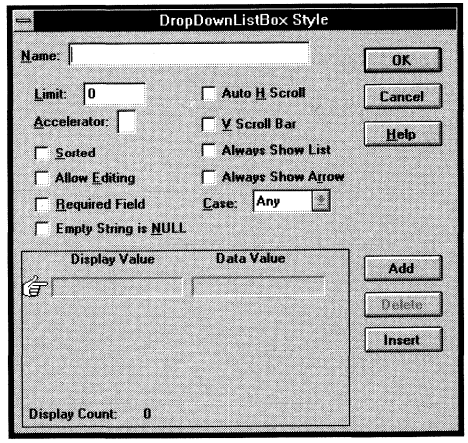
3 Select Edit Style from the popup menu.

The Edit Style dialog box displays listing the defined edit styles.



4 Click the button representing the type of edit style you want to create in the New group, or select an existing edit style and click Edit to modify one.

The dialog box where you define the edit style displays. Here is the dialog box for the DropDownListBox edit style:



5 If creating a new edit style, name it.

6 Specify the attributes of the edit style, as described in "Defining edit styles" on page 547.

7 Click OK.

You return to the Edit Style dialog box. The new or modified edit style is highlighted.

8 Click OK.

You return to the workspace. The new or modified edit style is stored in the repository and applied to the column.

Another way to create and modify an edit style
You can also maintain edit styles by selecting Objects➤Edit Style Maintenance from the menu bar.

You can use the new or modified edit style with any column of the appropriate data type in the database.

Adjusting size
After you specify an edit style for a column, you might want to adjust the width or height of the column to accommodate the edit style.

❖ To apply an existing edit style to a column:

1 Open the table containing the column you want to apply an edit style to.

2 Position the mouse over the column and click the right mouse button to display the column's popup menu.

3 Select Edit Style from the popup menu.

The Edit Style dialog box displays listing the defined edit styles.

4 Select a style.

5 Click OK.

The column now has the selected edit style associated with it in the repository.

Another way to apply edit styles
You can also apply edit styles by selecting Definition from a table's popup menu to display the Alter Table dialog box. Then select a column and pick an edit style from the Edit dropdown listbox in the Extended Attributes section at the bottom of the dialog box.

❖ To remove an edit style from a column:

1 Open the table containing the column you want to apply an edit style to.

2 Position the mouse over the column and click the right mouse button to display the column's popup menu.

3 Select Edit Styles from the popup menu.

The Edit Style dialog box displays with the currently used edit style highlighted.

4 Click the highlighted edit style.

PowerBuilder removes the highlighting.

5 Click OK.

You return to the workspace. The column is no longer associated with the edit style.

Working in the DataWindow painter

Edit styles 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 edit style in the DataWindow painter by choosing another edit style from the repository or defining an ad hoc style for one specific column.

❖ To specify an edit style for a column:

1 In the DataWindow painter, display the column's popup menu and choose the type of edit style from the Edit Style cascading menu.

The dialog box for the selected edit style type displays.

2 Select a stored edit style from the Name box.
or
Define your own edit style, as described next.

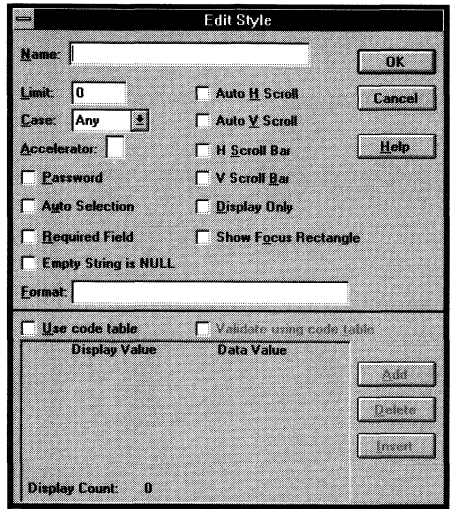
3 Click OK.

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 using the Edit Style dialog box:



For example, you can:

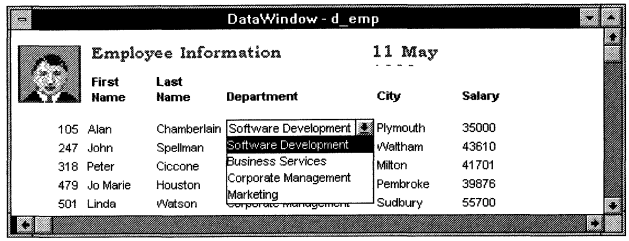
- Restrict the number of characters the user can enter (the Limit box)
- Convert the case of characters upon display (the Case box)
- Have entered values display as asterisks for sensitive data (the Password box)
- Allow users to tab to the column but not change the value (the Display Only box)
- Define a code table to determine which values are displayed to users and which values are stored in the database

See "Defining a code table" on page 557.

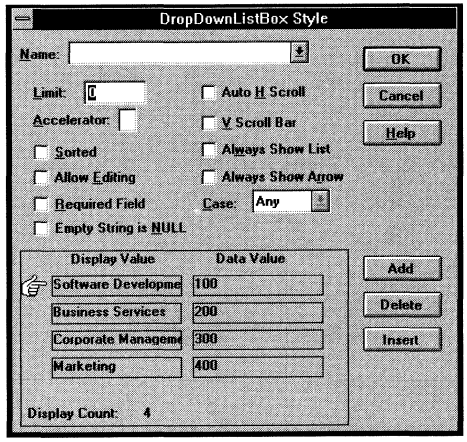
For more about Edit edit style attributes, click the Help button.

The DropDownListBox edit style

You can use the DropDownListBox edit style to have columns display as dropdown listboxes during execution.



You display column data as a dropdown listbox using the DropDownListBox Style dialog box.



Typically this edit style is used with code tables, where you can specify display values, which users see, and 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 dropdown listbox.

In the preceding example, users see the values Software Development, Business Services, Corporate Management, and Marketing. When one of these values is selected, the corresponding data value is 100, 200, 300, or 400.

During execution

You can define and modify a code table for a column in a script by using the SetValue function during execution. To obtain the value of a column during execution, use GetValue. To clear the code table of values, use the ClearValues function.

For information about the ClearValues, GetValue, and SetValue functions, see the *Function Reference*.

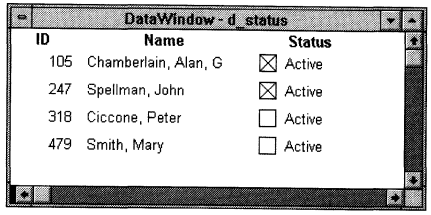
For more information

For information about other attributes of the DropDownListBox edit style, click Help.

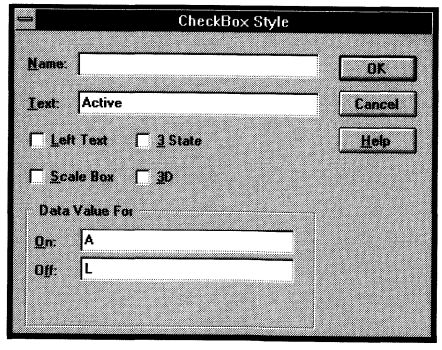
For more about code tables, see "Defining a code table" on page 557.

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 checkbox. Users can select or deselect the checkbox to specify a value. In the following DataWindow, users can simply check or uncheck a box to indicate an employee's status.



You display column data as checkboxes using the CheckBox Style dialog box.



❖ To use the CheckBox edit style:

1 Enter the text you want displayed next to the checkbox during execution in the Text 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 Enter the value you want put in the DataWindow buffer when the CheckBox is checked, unchecked, or optionally in the third state in the Data Value For box. A box labeled Other displays only if you select the 3 State option to indicate the checkbox can be in the third state (neither on nor off).

3 If necessary, widen the column to accommodate the checkbox. You can do that in the Database painter in the Alter Table dialog box or in the DataWindow painter workspace.

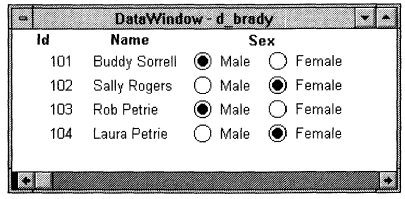
What happens

The value you enter in the Text box becomes the display value, and values entered for On, Off, and Other become the data values.

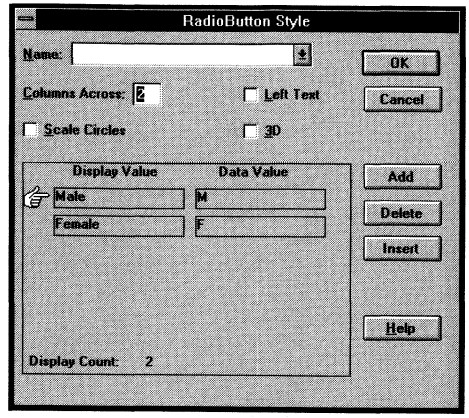
When the user checks or unchecks the checkbox during execution, PowerBuilder enters the appropriate data value in its buffer. When Update is issued, PowerBuilder sends the data values to the database.

The RadioButton edit style

If a column can take one of a small number of values, you might want to display the column as radio buttons.



You display column data as radio buttons using the RadioButton Style dialog box.



❖ To use the RadioButton edit style:

1 Specify how many radio buttons will display in a single line.

2 Enter a set of display and data values for each button you want to display. Each display value you enter becomes the text of the button; the data value is put in the DataWindow buffer when the button is clicked.

> **Using accelerator keys**
> If the radio button has an accelerator key, enter an ampersand (&) before the letter in the Display Value for the button that represents the accelerator key.

3 If necessary, widen and lengthen the column to accommodate the buttons. You can do that in the Database painter in the Alter Table dialog box or in the DataWindow painter workspace.

What happens

Users select values by clicking a radio button. When Update 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, phone numbers have a three-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.

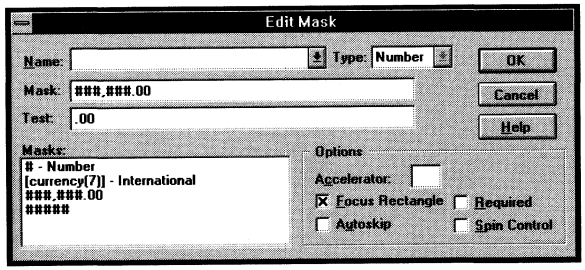
Edit masks consist of special characters that determine what can be entered in the column. They can also contain punctuation characters to aid the user.

For example, to make it easier for users to enter phone numbers in the proper format, specify this mask:

(###) ###-####

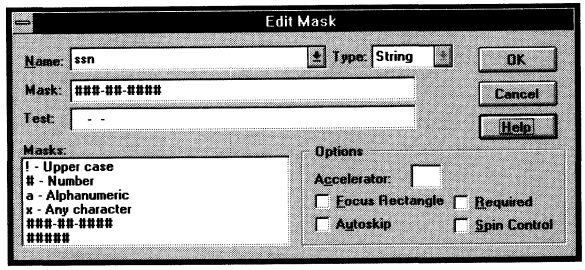
During execution, the punctuation characters display in the box and the cursor jumps over them as the user types.

You display column data with an edit mask using the Edit Mask dialog box.



❖ To use an edit mask:

1 Define the mask in the Mask box. The special characters used in the mask for the current data type display in the Masks box. Here is an edit mask for Social Security numbers:



2 (Optional) Test the mask by typing a value in the Test box.

3 Specify other attributes for the edit mask. For information on the other attributes, click the Help button.

4 Click OK.

Special characters

Special characters used in edit masks are the same ones used in display formats.

For information about special characters used in masks, see "Defining display formats" on page 533.

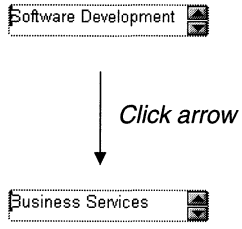
Keyboard behavior

Note the following about how certain keystrokes behave in edit masks.

- Pressing BACKSPACE always deletes the preceding character, even if SHIFT is pressed.
- Pressing DELETE deletes everything that is selected.
- Non-numeric edit masks treat any characters that don't match the mask pattern as delimiters.
- With Date, Datetime, and Time edit masks, if the user types a non-zero number in the first position of dd, mm, hh, ss, or mm (minutes) and then types the delimiter character, the typed digit is entered as the second position and the first position is set to 0. For example, typing **1/1/92** results in **01/01/92**.
- With Date, Datetime, and Time edit masks, if the user types a number in the first position that is greater than the maximum day, month, hour, and so on, the typed number is entered as the second position and the first position is set to 0. The cursor is positioned in front of the second position.
- With Date, Datetime, and Time edit masks, 00/00/00 or 00/00/0000 are interpreted as the NULL value for the column.

Using spin controls

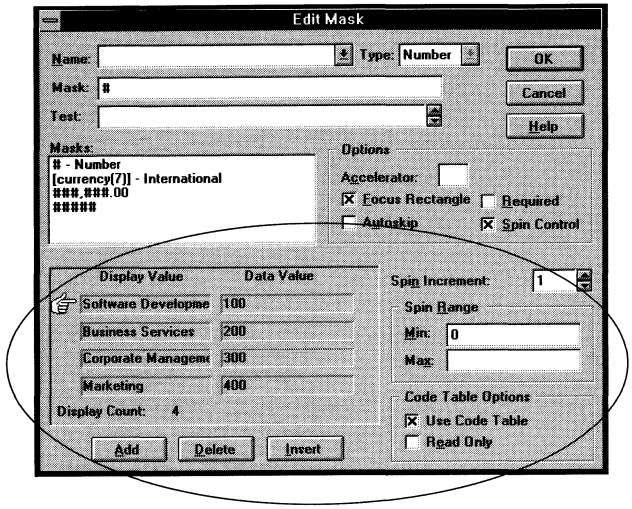
You can define an edit mask as a **spin control**, a box that contains up and down arrows that users can click to cycle through fixed values. For example, you can set up a code table that provides the valid entries in a column; the user simply clicks an arrow to select the one they want. Used this way, a spin control works like a dropdown listbox that doesn't drop down: users can display one value at a time by clicking an arrow.



❖ To define an edit mask as a spin control:

1 Select the Spin Control box in the Options group.

Options for the spin control display.



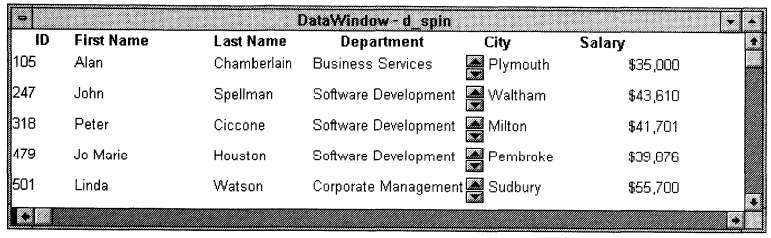
2 Specify the needed information. For example, to provide a code table with display values and data values, select the Use Code Table checkbox in the Code Table Options group and enter values for Display Value and Data Value.

For more about code tables, see "Defining a code table" on page 557. For more about the options for spin controls, click the Help button.

3 Click OK to return to the workspace.

4 If necessary, widen the column to accommodate the display values.

The following DataWindow uses the edit mask defined in the preceding dialog box. The DeptID column has been defined as having a spin control, with department names as display values and the numeric department ID as the data values. Users can click the arrows in the spin control to cycle through the valid values. Compare this edit style with the DropDownListBox edit style shown on page 548; the only difference is that values don't drop down with the spin control.

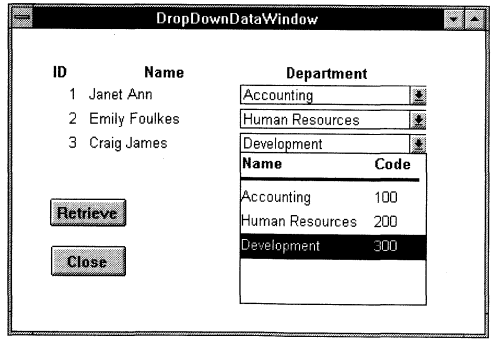


The DropDownDataWindow edit style

Sometimes another data source determines which data is valid for a column. Consider this situation: The Department table records your company's departments. It includes two columns, DeptID and DeptName. The Employee table records your employees. The Department column in the Employee table can have any of the values of the DeptID column in the Department table.

As new departments are added in your company, you want the DataWindow object containing the Employee table to automatically provide the new departments as choices for your user when entering values in the Department column.

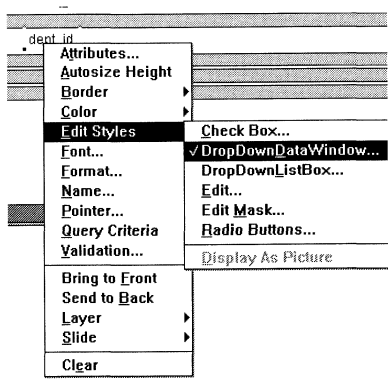
In situations such as these, you can specify that a column has the DropDownDataWindow edit style: it is populated from another DataWindow object. When the user goes to the column, the contents of the DropDownDataWindow object display, showing the latest data.



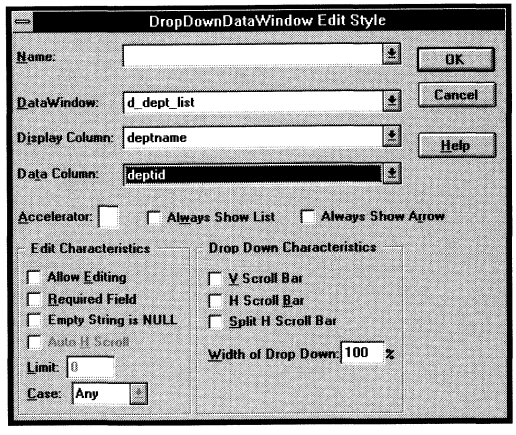
❖ To use the DropDownDataWindow edit style:

1 Create a DataWindow object that contains the columns 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 containing values to be stored in the database. In the example above, you would create a DataWindow object containing the DeptID and DeptName columns in the Department table. Assume this DataWindow object is named d_dept_list.

2 For the column getting its data from the DataWindow object, select the DropDownDataWindow edit style. In the example, you would specify the DropDownDataWindow edit style for the Department column in the Employee table.



The DropDownDataWindow Edit Style dialog box displays.



3 Name the DataWindow object that contains the data for the column in the DataWindow box (in the example, d_dept_list).

4 Select the column containing the values that will display to the user in the Display Column box (in the example, DeptName).

5 Select the column containing the values that will be stored in the database in the Data Column box (in the example, DeptID).

6 Specify other attributes for the edit style. For more information, click Help.

7 Click OK.

What happens

During execution, when data is retrieved into the DataWindow, the column whose edit style is DropDownDataWindow will itself be populated as data is retrieved into the DataWindow object serving as the dropdown DataWindow.

When the user goes to the column and drops it down, the contents of the dropdown 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.

Defining a code table

To reduce storage needs, you frequently want to store short, encoded values in the database. But these encoded values may not be meaningful to users. To make the application 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 the user sees during execution. The data values are those that are saved in the database.

How code tables are implemented

You use code tables through a column's edit style. Five edit styles support code tables:

- Edit
- DropDownListBox
- RadioButton
- DropDownDataWindow

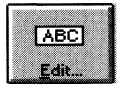
- EditMask, using spin control

This section describes how to define code tables in the Database painter.

Allowing NULL values
To allow NULL values for a column using a code table, specify NULL! as the data value. NULL! is an internal PowerBuilder code that indicates that NULL values are allowed. If you enter NULL! as a data value, you should specify a display format for NULLs for the column.

❖ To define a code table with the Edit edit style:

1 Select Edit Style from a column's popup menu.



2 Click the Edit button in the New group to create a new Edit edit style.

The Edit Style dialog box displays.

3 Select the Code Table checkbox.

4 Enter the display and data values for the code table.

5 If you want to restrict users to entering values in the code table, select the Validate Using Code Table checkbox.

For more information, see "Validating user input" on page 561.

6 Specify other attributes of the edit style (for more information, click Help).

7 Click OK.

❖ To define a code table with the DropDownListBox edit style:

1 Select Edit Style from a column's popup menu.

2 Click the DropDownListBox button in the New group to create a new edit style.

The DropDownListBox Style dialog box displays.

3 Enter the display and data values for the code table.

4 If you want to restrict users to entering values in the code table, deselect the Allow Editing checkbox. To allow users to type values, select Allow Editing.

For more information, see "Validating user input" on page 561.

5 Specify other attributes of the edit style (for more information, click Help).

6 Click OK.

❖ To define a code table with the RadioButton edit style:

1 Select Edit Style from a column's popup menu.

2 Click the RadioButton button in the New group to create a new edit style.

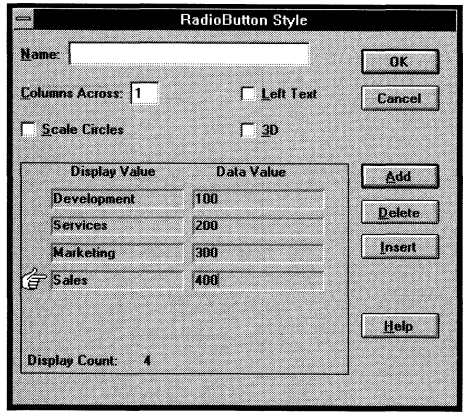
The RadioButton Style dialog box displays.

3 Enter the display and data values for the code table.

4 Specify other attributes of the edit style (for more information, click Help).

5 Click OK.

Here is an example.



❖ To define a code table with the DropDownDataWindow edit style:

1 Select Edit Style from a column's popup menu.

2 Click the DropDownDataWindow button in the New group to create a new edit style.

The DropDownDataWindow Style dialog box displays.

3 Specify the column that provides the display values in the Display Column box.

4 Specify the column that provides the data values in the Data Column box.

5 If you want to restrict users to entering values in the code table, deselect the Allow Editing checkbox. To allow users to type values, select Allow Editing.

6 Specify other attributes of the edit style (for more information, click Help).

7 Click OK.

❖ **To define a code table with the EditMask edit style:**

1 Select Edit Style from a column's popup menu.

2 Click the EditMask button in the New group to create a new edit style.

The EditMask Style dialog box displays.

3 Select the Spin Control checkbox in the Options group.

4 Select the Use Code Table checkbox in the Code Table Options group.

5 Enter the display and data values for the code table.

6 Specify other attributes of the edit style (for more information, click Help).

7 Click OK.

How code tables are processed

When data is retrieved into a DataWindow 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 is displayed. If there is no match, the actual value displays.

Consider this example:

Display values	Data values
Massachusetts	MA
Massachusetts	ma
ma	MA

Display values	Data values
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, if the column has a code table that contains duplicate display values, each value displays only once. So if this code table is defined for a column in the DataWindow object that has a DropDownListBox edit style, Massachusetts and Rhode Island display in the ListBox portion of the DropDownListBox.

Validating user input

When a user enters data into a DataWindow column, 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 DataWindow buffer. For each display value, the first data value is used. Using the sample code table, if the user enters Massachusetts, ma, or Mass, MA is the data value.

You can specify that *only* the values in the code table are acceptable:

- For a column using the Edit edit style, select the Validate Using Code Table checkbox.
- For the DropDownListBox and DropDownDataWindow edit styles, deselect the Allow Editing checkbox: users cannot type a value.

If you have selected Validate Using Code Table checkbox 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.

When the code table processing is complete, the ItemChanged or ItemError event is triggered.

Valid data
The data values in the code table must pass validation for the column. The data must have the same data type as the column.

Working with validation rules

When a user enters data in a DataWindow, you want to be sure the data is valid before you use it to update the database table associated with the DataWindow. One way to do this is through validation rules.

To use a validation rule, you define the rule and associate it with a column in the Database painter or the DataWindow painter.

Another technique
Another way to perform data validation is through code tables, which are implemented through a column's edit style.

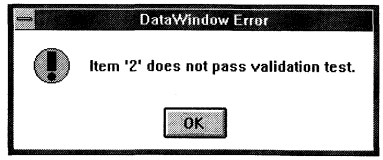
For more information, see "Working with edit styles" on page 541.

Understanding validation rules

Validation rules are criteria that a DataWindow uses to validate data entered into a column by a user. They are PowerBuilder-specific, used only in a DataWindow. They are 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 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, PowerBuilder does not accept the entry 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 an ItemError script to cause different processing to happen.

For more information, see the chapter on using DataWindow objects in *Building Applications*.

During execution
In scripts, you can use the GetValidate function to obtain the validation rule for a column and the SetValidate function to change the validation rule for a column.

For information about the GetValidate and SetValidate functions, see the *Function Reference*.

Working with validation rules

You work with validation rules 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 validation rules

  The validation rules are stored in the repository. Once you define a validation rule, it can be used by any column of the appropriate data type in the database.

- Assign validation rules to columns

  These rules 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 validation rule assigned to a column in the Database painter
- Create an ad hoc, unnamed rule to use with one specific column

Coupling between Database and DataWindow painters

Once you have placed a column that has a validation rule from the repository in a DataWindow object, there is no longer any link to the named rule in the repository.

If the definition of the validation rule later changes in the repository, the rule for the column in a DataWindow object will not change.

Working in the Database painter

Typically you will 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.

This section describes the ways you can manipulate validation rules in the Database painter.

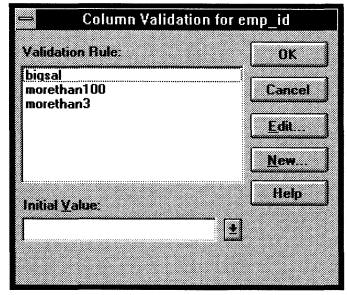
❖ To create or modify a validation rule:

1 In the Database painter, open the table containing the column you want to apply a validation rule to.

2 Position the mouse over the column and display the column's popup menu.

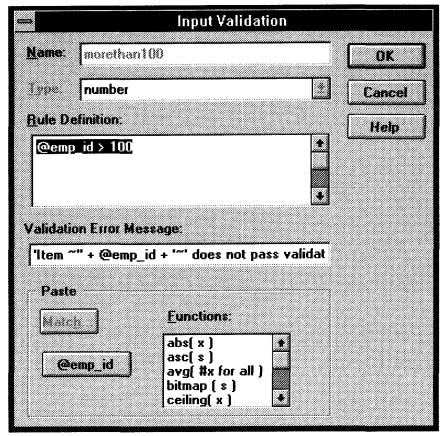
3 Select Validation from the popup menu.

A Column Validation dialog box displays listing all defined validation rules for the corresponding data type.



4 Click New to create a new rule, or click Edit to modify an existing one.

The Input Validation dialog box displays.



5 If creating a new rule, name it.

6 Define the expression for the validation rule, as described on page 568.

7 (Optional) Customize the error message, as described on page 570.

8 Click OK.

You return to the Column Validation dialog box. The new or modified validation rule is highlighted.

9 Click OK.

You return to the workspace. The new or modified rule is stored in the repository and applied to the column.

You can use this rule with any column of the appropriate data type in the database.

Another way to create and modify a validation rule

You can also maintain validation rules by selecting Objects➤Validation Maintenance from the menu bar.

The only difference is that when you create a validation rule this way, you must specify the appropriate data type in the Type dropdown listbox (because you are not creating the validation rule for a specific column, you need to tell PowerBuilder what type of column the rule is meant for).

❖ To apply an existing validation rule to a column:

1 Open the table containing the column you want apply a validation rule to.

2 Position the mouse over the column and display the column's popup menu.

3 Select Validation from the popup menu.

A Column Validation dialog box displays listing all defined validation rules for the corresponding data type.

4 Select a rule.

5 Click OK.

The column now has the selected rule associated with it in the repository. Whenever you use this column in a DataWindow object, it will use this validation rule by default.

Another way to apply validation rules
You can also apply validation rules by selecting Definition from a table's popup menu to display the Alter Table dialog box. Then select a column and pick a validation rule from the Valid dropdown listbox in the Extended Attributes section at the bottom of the dialog box.

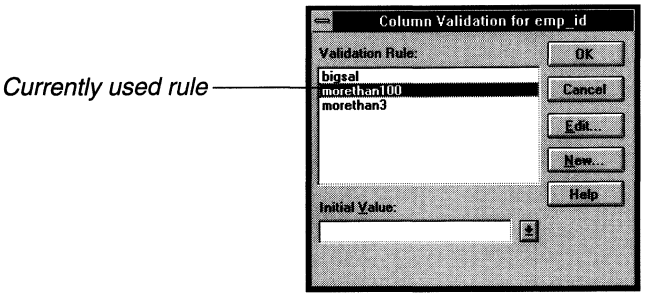
❖ To remove a validation rule from a column:

1 Open the table containing the column you want to apply a validation rule to.

2 Position the mouse over the column and display the column's popup menu.

3 Select Validation from the popup menu.

A Column Validation dialog box displays listing all defined validation rules for the corresponding data type. The currently used rule is highlighted.



4 Click the highlighted rule.

PowerBuilder removes the highlighting.

5 Click OK.

You return to the workspace. The column is no longer associated with the validation rule.

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 PowerScript expression in validation rules.

Validation rules can include any non-object-level PowerScript functions, including user-defined functions. The commonly used PowerScript functions are displayed in the Functions listbox and can be pasted into the definition.

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 repository with the placeholder name. When the validation rule is used for a column in a DataWindow object, PowerBuilder substitutes the value of the column for *placeholder*.

Pasting the placeholder

The name of the column for which you initially create the rule is frequently used as the placeholder. In the Input Validation dialog box, the current column name preceded by an @ displays just below the Match button. You can click the button to paste the placeholder 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 for one of the columns. If defining it for Salary, the expression would be:

```
@salary > 0
```

Then use the same validation rule for Age.

During execution, PowerBuilder substitutes the appropriate values for the column data when the rule is applied.

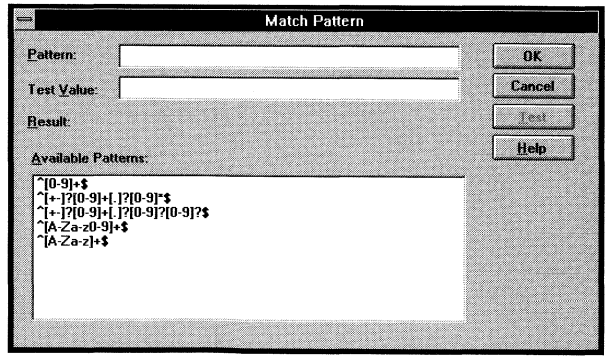
Using match values for character columns

If the column you are defining the validation rule for is a character column, the Match button is active. You use Match to match the contents of the 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.

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 When you are satisfied that the pattern is correct, click OK to return to the Input Validation dialog box with the match pattern.

5 Click OK again.

For more on the Match function and text patterns, see the *Function Reference*.

Customizing the error message

When you define a validation rule, PowerBuilder automatically creates the error message that displays by default when a user enters an invalid value during execution:

```
'Item ~" + @ColumnName + '~' does not pass validation test.'
```

You can edit the string expression to create a custom error message.

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:

1 Select Validation from the column's popup menu.

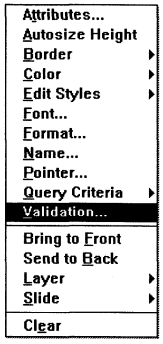
The Column Validation dialog box displays.

2 Specify a value in the Initial Value box.

Working in the DataWindow painter

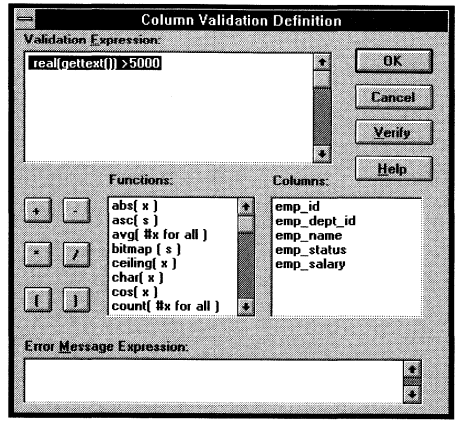
Validation rules you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the validation rule in the DataWindow painter by defining an ad hoc rule for one specific column.

❖ To specify a validation rule for a column:



1 In the DataWindow painter, select Validation from the column's popup menu.

The Column Validation Definition dialog box displays. The current validation rule displays in the Validation Expression box.



2 Create or modify the validation expression, as described next.

> **Used for current column only**
> If you create a validation rule here, it is used only for the current column. The rule is not saved in the repository for use with other columns. To save a rule in the repository, define it in the Database painter.

3 Enter a string or string expression if you want to customize the validation error message.

For more information, see "Customizing the error message" on page 570.

4 Click Verify to verify the syntax.

5 Click OK.

Specifying the expression

Since the user might just have entered a value in the column, validation rules refer to the current data value, which you can obtain through the GetText PowerScript 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 data type other than string.

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 by the user in the current column is a positive number, 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:

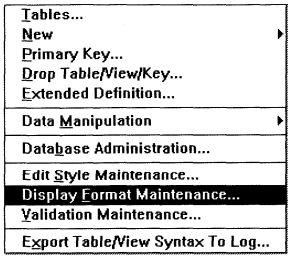
```
"Price is " +  if(price >= 1000,
"greater than or equal to","less than") +
" 1000. Commission must be between " +
If(price >= 1000,".10", ".04") +  " and " +
If(price >= 1000, ".20.", ".09.")
```

Maintaining the entities

PowerBuilder provides facilities you can use to create, modify, and delete display formats, edit styles, and validation rules independently of their association with columns.

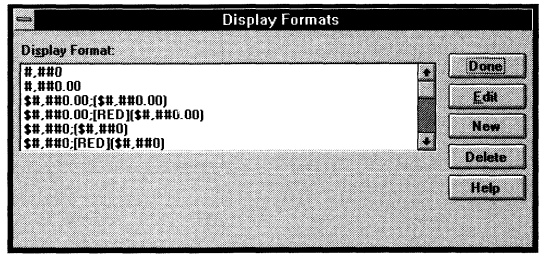
❖ To maintain display formats, edit styles, and validation rules:

1 Open the Database painter.



2 Select Edit Style Maintenance, Display Format Maintenance, or Validation Maintenance from the Objects menu.

A dialog box displays listing all the corresponding entities that are in the repository. Here is the dialog box listing display formats:



3 Do one of the following:

- To create a new entity, click New.
- To modify an existing entity, select it, then click Edit.
- To delete an existing entity, select it, then click Delete.

> ***Caution***
> *If you delete a display format, edit style, or validation rule, it is removed from the repository and from all columns that had used it.*

CHAPTER 16

Filtering, Sorting, and Grouping Rows

About this chapter

This chapter describes how you can customize your DataWindow object by doing the following in the DataWindow painter:

- Define filters to limit which of the retrieved rows are displayed in the DataWindow object
- Sort rows after they have been retrieved from the database
- Display the rows in groups and calculate statistics on each group

Contents

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 during execution.

However, you may want to further limit the data that displays in the DataWindow during execution. For example:

- You might want to retrieve many rows and display only a subset initially (and perhaps allow the user to specify a different subset of rows to display during execution).
- You might want to limit the data that is displayed using PowerScript functions (such as IF) that are not valid in the SELECT statement.

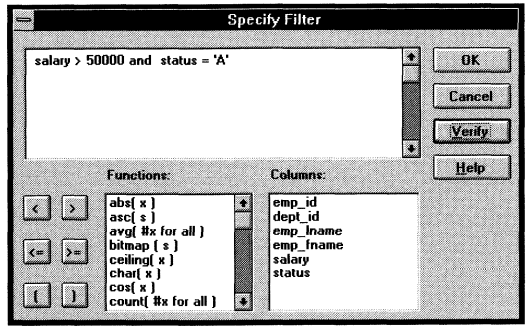
To limit the rows that display during execution, you use filters. You can define a filter in the DataWindow painter.

Filters don't affect which rows are retrieved
A filter operates against the retrieved data. It does not re-execute the SELECT statement.

❖ To define a filter:

1 In the DataWindow painter, select Rows➤Filter from the menu bar.

The Specify Filter dialog box displays.



2 Enter a boolean expression that PowerBuilder will test against each retrieved row. If the expression evaluates to TRUE, the row will be displayed. You can specify any valid PowerScript 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
So that an application you build will run the same no matter in which country it is deployed, filter expressions require U.S. notation for numbers. That is, comma always represents the thousands delimiter and period always represents the decimal place when you specify expressions in the development environment.

For information about PowerScript expressions, see *PowerScript Language*.

3 (Optional) Click Verify to make sure the expression is valid.

4 Click OK to return to the workspace.

5 (Optional) Test the filter by clicking Preview.

Only rows meeting the filter criteria will be displayed.

Filtered rows and updates
Filtered rows are updated when you update the database.

❖ To remove a 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.

Examples of filters

Assume that a DataWindow retrieves employee rows. Three of the columns are Salary, Status, and Emp_Lname.

The following filter displays only employees with salaries over $50,000:

```
Salary > 50000
```

The following filter displays only active employees:

```
Status = 'A'
```

The following filter displays active employees with salaries over $50,000:

```
Salary > 50000 AND Status = 'A'
```

The following filter displays employees whose last names begin with H:

```
left(Emp_Lname, 1) = 'H'
```

Setting filters in a script

You can use the SetFilter and Filter functions in a script to dynamically modify a filter that was set in the DataWindow painter.

For information about SetFilter and Filter, see the *Function Reference*.

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. The rows are brought into PowerBuilder already sorted.

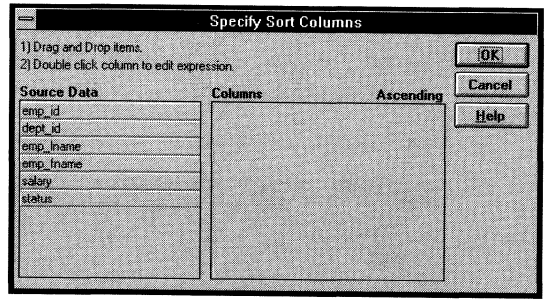
However, you might want to sort the rows after they are retrieved. For example:

- You might want to offload the processing from the DBMS.
- You might want to sort on an expression, which is not allowed in the SELECT statement but is allowed in PowerBuilder.

❖ To sort the rows:

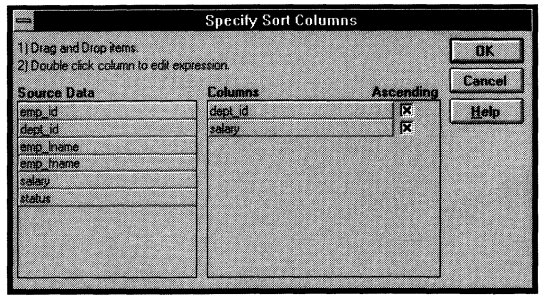
1 Select Rows➤Sort from the menu bar.

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 checkbox with an X in it displays under the Ascending heading to indicate that the values will be sorted in ascending order. To sort in descending order, uncheck the checkbox.



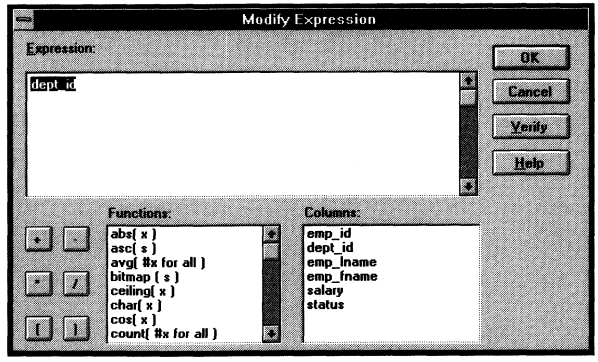
Precedence of sorting

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 will be sorted by dept ID. Within dept ID, rows will be sorted by salary.

To change the precedence order, drag the column names in the Column box to the order you want.

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 an item in the Columns box.

The Modify Expression dialog box displays.



4 Specify the expression 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.

5 When you have specified all the sort columns and expressions, click OK.

PowerBuilder sorts the rows.

Suppressing repeating values

You might want to suppress repeating values for a column. For example, if you have sorted employees by department ID, you might want to suppress the display of department ID in a row when it is the same as the ID in the preceding row.

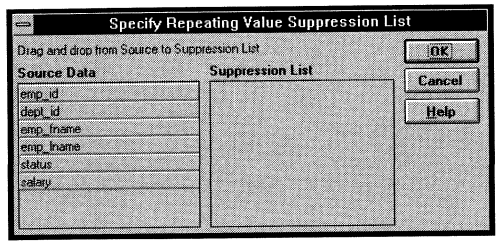
DataWindow - d_emp_40

ID	Dept	Name	Status	Salary
105	100	Matthew Cobb	A	$62,000.00
453		Andrew Rabkin	A	$64,500.00
529		Dorothy Sullivan	A	$67,890.00
604		Albert Wang	A	$68,400.00
243		Natasha Shishov	A	$72,995.00
316		Lynn Pastor	A	$74,500.00
445		Kim Lull	A	$87,900.00
862		John Sheffield	A	$87,900.00
501		David Scott	A	$96,300.00
1021	200	Paul Sterling	A	$64,900.00
949		Pamela Savarino	A	$72,300.00
902		Moira Kelly	A	$87,500.00
1483	300	John Letiecq	L	$75,400.00
1293		MaryAnne Shea	A	$138,948.00
1607	400	Benjamin Packard	A	$61,300.00
1576		Scott Evans	A	$68,940.00

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.

❖ To suppress repeating values:

1 Select Rows➤Suppress Repeating Values from the menu bar.



2 Drag the columns whose repeated values you want to suppress from the Source Data box to the Suppression List box.

3 Click OK.

> **If you change your mind**
> You can remove a column from the suppression list by simply 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.

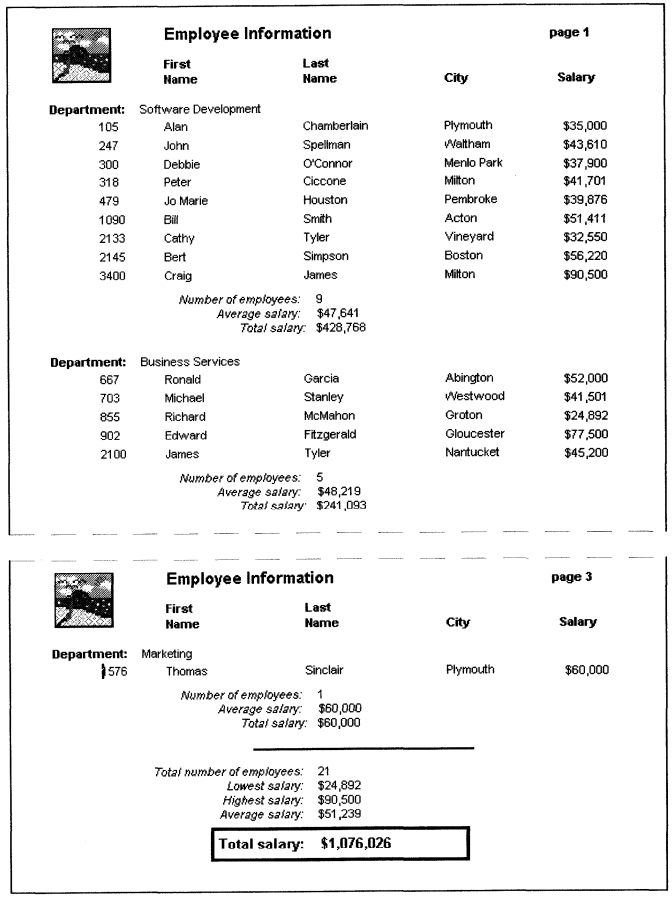
Each group is defined by one or more DataWindow columns. Each time the value in a grouping column changes, a **break** occurs and a new section begins.

For each group you can:

- Display the rows in each section
- Specify the information you want displayed at the beginning and end of each section
- Specify page breaks after each break in the data
- Reset the page number after each break

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 the department name before the first row for that department.
- It displays the number of employees, average salary, and total salary for each department.
- It displays totals for the company at the end.



Employee Information page 1

	First Name	Last Name	City	Salary
Department:	Software Development			
105	Alan	Chamberlain	Plymouth	$35,000
247	John	Spellman	Waltham	$43,610
300	Debbie	O'Connor	Menlo Park	$37,900
318	Peter	Ciccone	Milton	$41,701
479	Jo Marie	Houston	Pembroke	$39,876
1090	Bill	Smith	Acton	$51,411
2133	Cathy	Tyler	Vineyard	$32,550
2145	Bert	Simpson	Boston	$56,220
3400	Craig	James	Milton	$90,500

Number of employees: 9
Average salary: $47,641
Total salary: $428,768

Department:	Business Services			
667	Ronald	Garcia	Abington	$52,000
703	Michael	Stanley	Westwood	$41,501
855	Richard	McMahon	Groton	$24,892
902	Edward	Fitzgerald	Gloucester	$77,500
2100	James	Tyler	Nantucket	$45,200

Number of employees: 5
Average salary: $48,219
Total salary: $241,093

Employee Information page 3

	First Name	Last Name	City	Salary
Department:	Marketing			
1576	Thomas	Sinclair	Plymouth	$60,000

Number of employees: 1
Average salary: $60,000
Total salary: $60,000

Total number of employees: 21
Lowest salary: $24,892
Highest salary: $90,500
Average salary: $51,239

Total salary: $1,076,026

How to do it

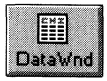
You can create a grouped DataWindow two ways:

- Use the Group presentation style to create a grouped DataWindow object from scratch (described next).
- Take an existing tabular DataWindow object and define grouping in the DataWindow painter workspace (page 589).

Using the Group presentation style

One of the DataWindow presentation styles is Group, which offers an easy way to create a grouped DataWindow object. It generates a tabular DataWindow that has one group level and some other grouping properties defined. You can then customize the DataWindow in the DataWindow painter workspace.

❖ To create a basic grouped DataWindow using the Group presentation style:

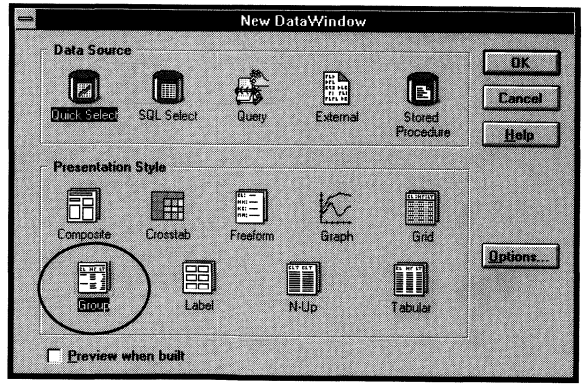


1 Click the DataWindow button in the PowerBar or PowerPanel.

The Select DataWindow dialog box lists the DataWindow objects in the current library.

2 Click the New button.

The New DataWindow dialog box displays.



3 Choose a data source.

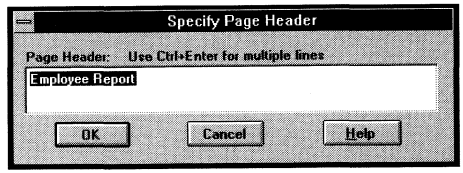
4 Choose the Group presentation style.

5 Click OK.

You are prompted to define the data for the DataWindow.

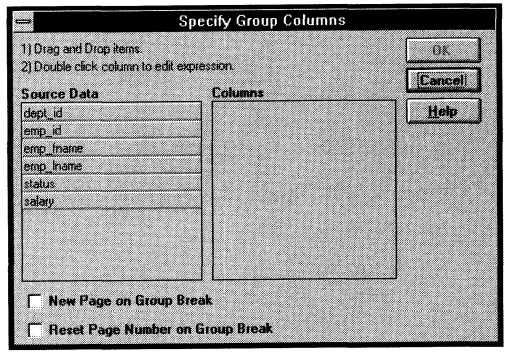
6 Define the data.

You are prompted to provide a page header for the DataWindow. The default header is the name of the table, followed by the word Report.

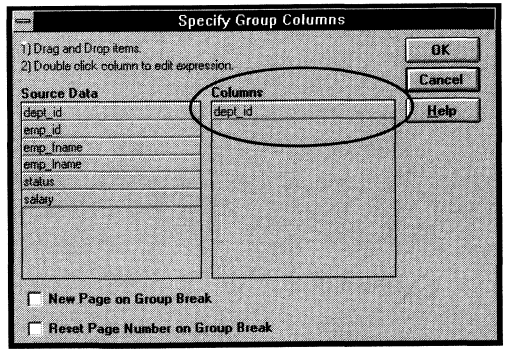


7 Specify the page header text.

You are prompted to define the grouping column.



8 Drag the column you want to group on from the Source Data box to the Columns box. You can define one group level at this point. You can define additional group levels later in the workspace.



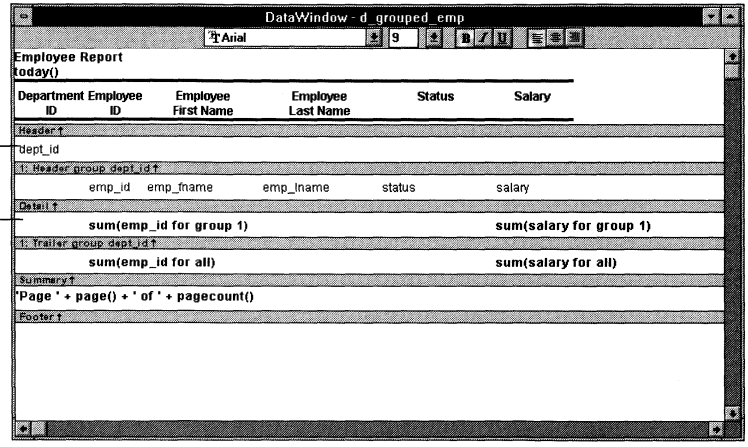
You can double-click a column name and type an expression as the grouping item. You can specify more than one grouping item expression for a group. A break occurs whenever the value concatenated from each column/expression changes.

9 If you want a page break each time a grouping value changes, select the New Page On Group Break box.

10 If you want page numbering to restart at 1 each time a grouping value changes, select the Reset Page Number On Group Break box.

11 Click OK.

The DataWindow painter workspace displays with the basic grouping properties set.



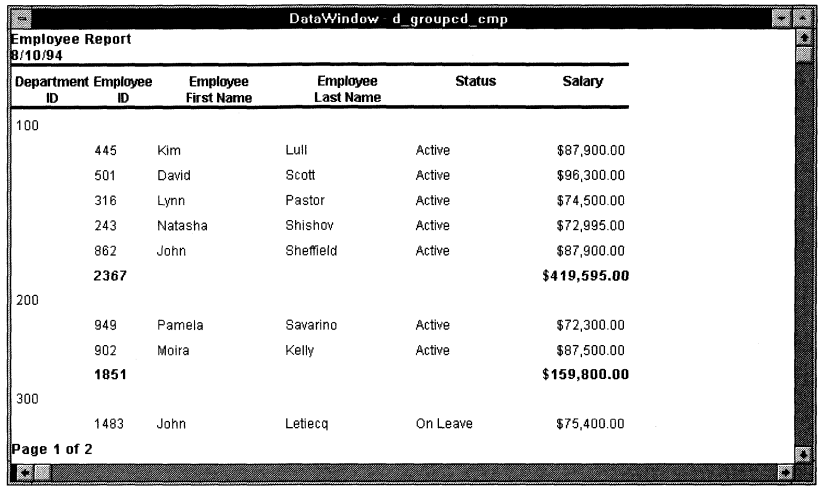
What PowerBuilder did

As a result of your specifications, PowerBuilder generates a tabular DataWindow 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 during preview:



What you can do

You can use any of the techniques available in a tabular DataWindow to modify and enhance the grouped DataWindow, such as moving objects, 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 and how to add features especially suited for grouped DataWindows (for example, add a second group level, define additional summary statistics, and so on).

DataWindow is not updatable by default

When you generate a DataWindow 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, you need to modify its update characteristics.

For more information, see "Controlling updates" in Chapter 14, "Enhancing DataWindow Objects."

Defining groups in an existing DataWindow object

Instead of using the Group presentation style to create a grouped DataWindow 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.

5 (Optional) Add summary statistics.

6 (Optional) Sort the groups.

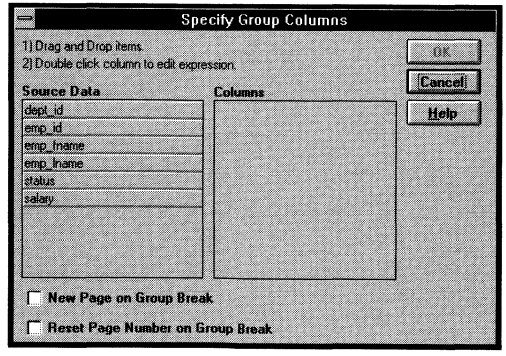
Steps 2 through 6 are described below.

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 585.

Creating subgroups

After defining your first group, you can define subgroups—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.

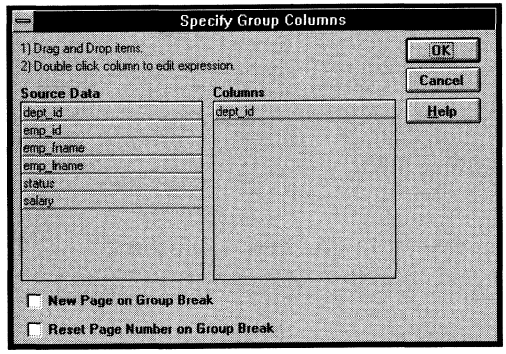
   You can specify as many levels of grouping as you need.

How groups are identified

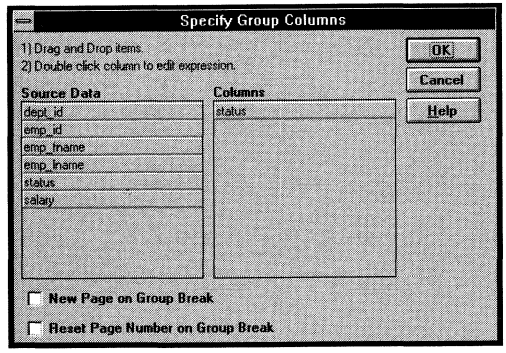
PowerBuilder assigns each group a number (or level) when you create the group. The first group you specify becomes group 1, the primary group. The second group becomes group 2, a subgroup within group 1, and so on.

For example, say you defined two groups in the following order:

Group 1



Group 2



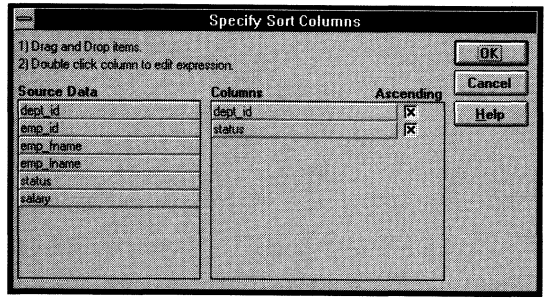
The rows in the DataWindow object will be grouped first by department (group 1). Within department, rows will be grouped by status (group 2). If you specify page breaks for the groups, a page break will occur when any of these values changes.

You use the group's number to identify it when defining summary statistics for the group, as described on page 593.

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 as shown above, select Rows➤Sort from the menu bar and specify Dept_ID and then Status as sorting columns.



You can also sort on additional rows if you want. 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 579.

Rearranging the DataWindow

When you create a group, PowerBuilder creates two new bands for each group:

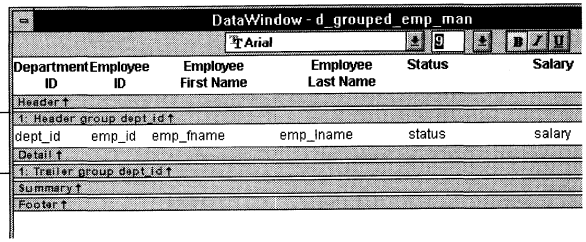
- A group header band
- A group trailer band

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

Header band for Group 1

Trailer band for Group 1



You can include any object in the DataWindow (such as columns, text, and computed fields) in the header and trailer bands of a group.

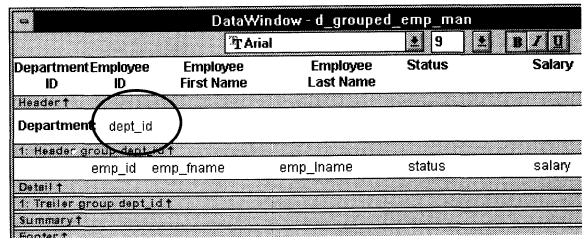
Using the group header band

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. For example, 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.

Dept_ID column has been moved to the group header band so it displays before each department's rows



During execution, you see this:

DataWindow - d_grouped_emp_man

Department ID	Employee ID	Employee First Name	Employee Last Name	Status	Salary
Department: 100					
	862	John	Sheffield	Active	$87,900.00
	243	Natasha	Shishov	Active	$72,995.00
	316	Lynn	Pastor	Active	$74,500.00
	445	Kim	Lull	Active	$87,900.00
	501	David	Scott	Active	$96,300.00
Department: 200					
	949	Pamela	Savarino	Active	$72,300.00
	902	Moira	Kelly	Active	$87,500.00
Department: 300					
	1293	Mary Anne	Shea	Active	$138,948.00
	1483	John	Letiecq	On Leave	$75,400.00

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 Click the Computed Field button.

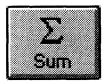
2 Click the workspace where you want the statistic.

The Computed Field Definition dialog box displays.

3 (Optional) Name the computed field (this allows you to reference the computed field later if you need to).

4 Specify the expression that defines the computed field (see below).

5 Click OK.



A shortcut to sum values

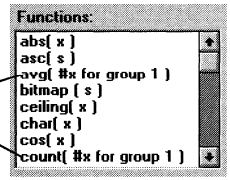
If you want to sum a numeric column, select the column in the workspace and click the Sum button in the PainterBar.

PowerBuilder automatically places a computed field in the appropriate band in the workspace.

Specifying the expression

Typically, you will use aggregate and other functions in your summary statistic. PowerBuilder lists functions you can use in the Functions box in the Computed Field Definition 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.

These functions reference group 1



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:

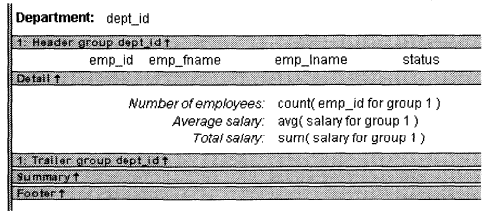
```
Count( Emp_Id for group 1 )
```

To get the average salary of employees in a department, specify:

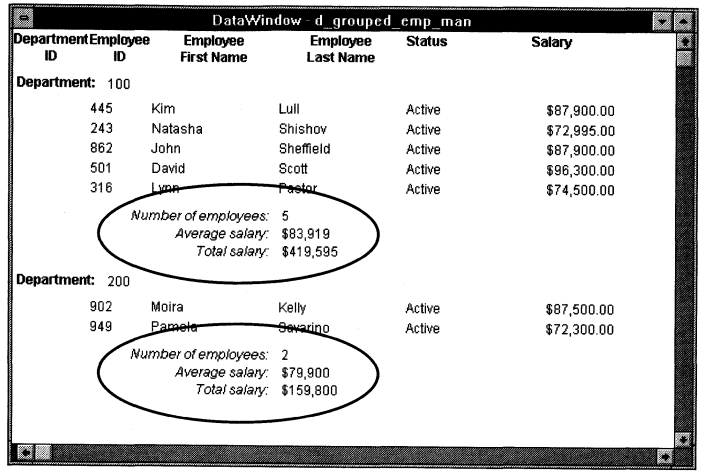
```
Avg( Salary for group 1 )
```

To get the total salary of employees in a department, specify:

```
Sum( Salary for group 1 )
```



During execution, you see this:



Sorting the groups

You can sort the groups in your DataWindow. For example, in a DataWindow showing employee information grouped by department, you might want to sort the departments (the groups) by total salary.

Typically this involves the use of aggregate functions, as described in "Adding summary statistics" on page 593. In the example in the preceding paragraph, you would sort the groups using the aggregate function Sum to calculate total salary in each department.

❖ To sort the groups:

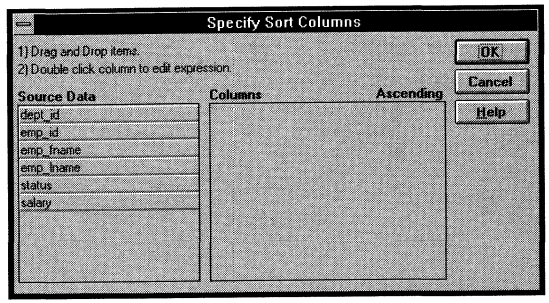
1 Place the mouse pointer on the group header bar (not inside the band).

The pointer turns into a double-headed arrow.

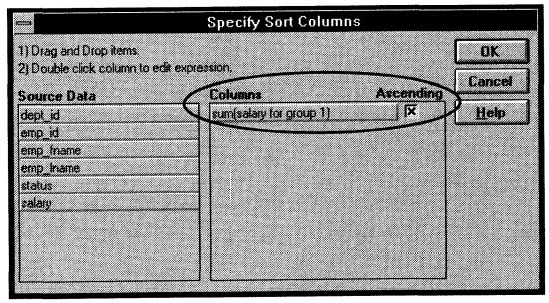
2 Display the popup menu for the group header.

3 Select Group Sort from the popup menu.

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)**.

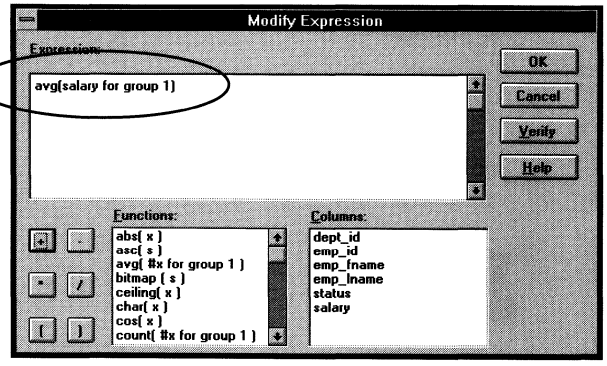


5 Select ascending or descending sort as appropriate.

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.

During execution, the groups will be sorted on the expression you specified.

CHAPTER 17

Using Nested Reports

About this chapter

This chapter provides information about creating reports that have other reports nested in them.

Contents

About reports and DataWindow objects

A report is the same as a nonupdatable DataWindow object. You can create reports in either the DataWindow painter or the Report painter.

This chapter shows the process of nesting reports using the Report painter, but you can do the same things in the DataWindow painter—the results are exactly the same.

For more about reports and DataWindow objects, see Chapter 13, "Defining DataWindow Objects."

About nested reports

A **nested report** is a report in another report.

There are two ways to create reports that have nested reports:

- Create a composite report using the Composite presentation style
- Place a nested report in another report

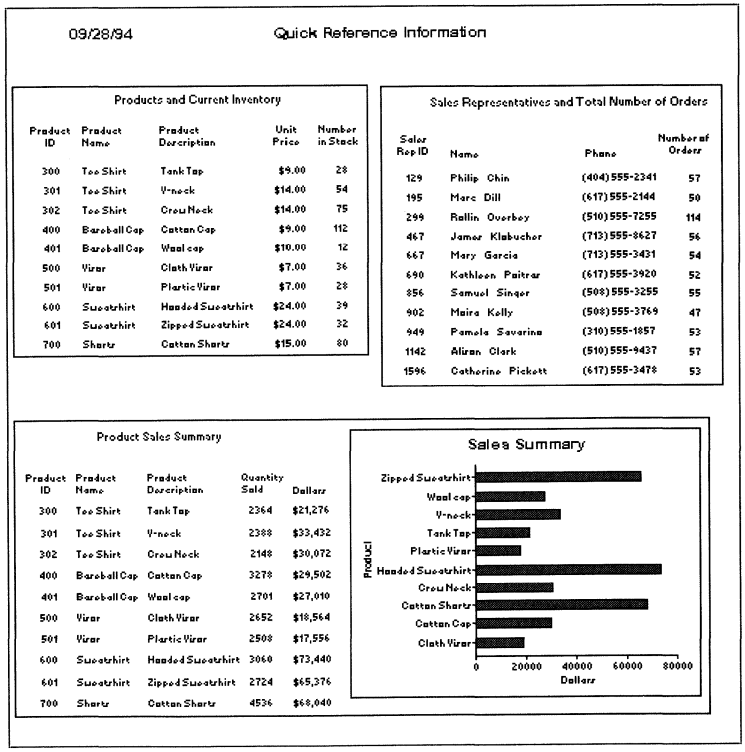
Creating a composite report

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 (DataWindow) on a page.

Composite report

For example, the following composite report consists of three tabular reports. One of the tabular reports includes a graph.



09/28/94 Quick Reference Information

Products and Current Inventory

Product ID	Product Name	Product Description	Unit Price	Number in Stock
300	Tee Shirt	Tank Top	$9.00	28
301	Tee Shirt	V-neck	$14.00	54
302	Tee Shirt	Crew Neck	$14.00	75
400	Baseball Cap	Cotton Cap	$9.00	112
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
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 Puitrar	(617)555-3920	52
856	Samuel Singer	(508)555-3255	55
902	Moira Kelly	(508)555-3769	47
949	Pamela Savarino	(310)555-1857	53
1142	Alinn Clark	(510)555-9437	57
1596	Catherine Pickett	(617)555-3478	53

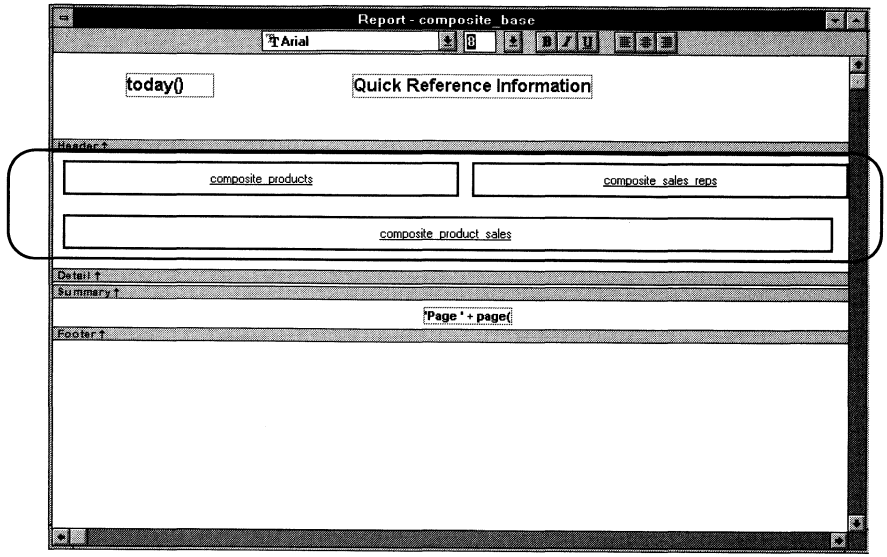
Product Sales Summary

Product ID	Product Name	Product Description	Quantity Sold	Dollars
300	Tee Shirt	Tank Top	2364	$21,276
301	Tee Shirt	V-neck	2388	$33,432
302	Tee Shirt	Crew Neck	2148	$30,072
400	Baseball Cap	Cotton Cap	3278	$29,502
401	Baseball Cap	Wool cap	2701	$27,010
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	2724	$65,376
700	Shorts	Cotton Shorts	4536	$68,040

Sales Summary

Composite report in the workspace

In the workspace, you see three boxes that represent the individual tabular reports that are included in the composite report. The only additional objects in this example are a title, date, and page number.



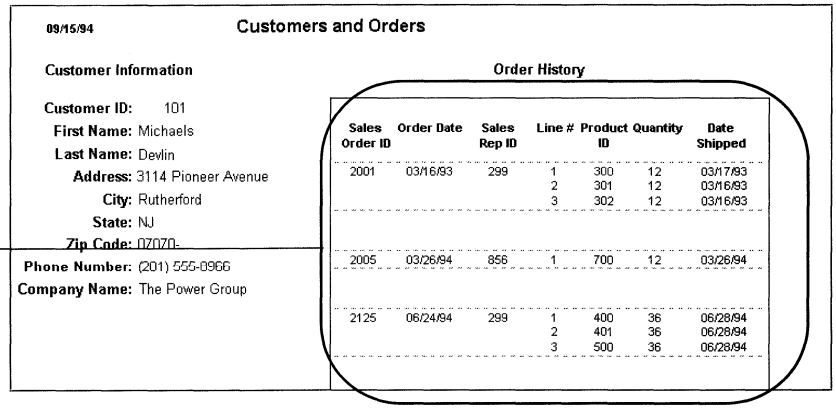
Placing a nested report within another report

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 that it is placed in (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

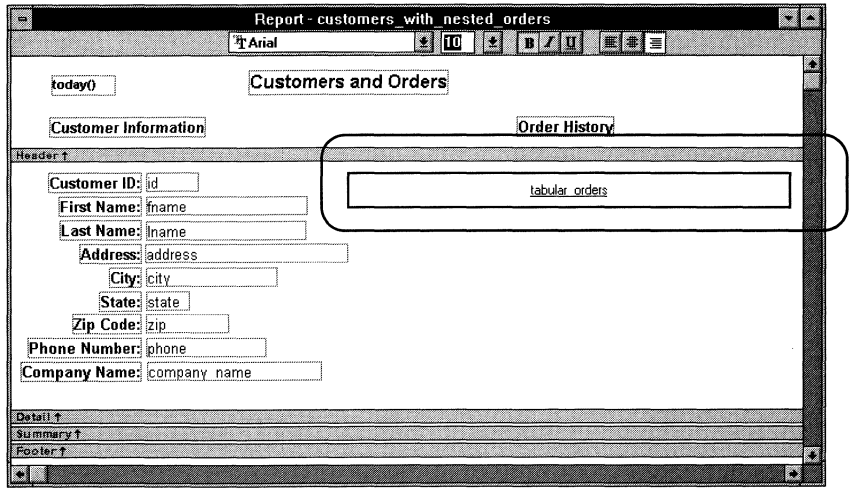
For example, the following freeform report lists all information about a customer and then includes a related nested report (it happens to be a tabular report). The related nested report lists every order that the customer has ever placed. The base report supplies the customer ID to the nested report, which requires a customer ID as a retrieval argument. This is an example of a master/detail relationship—one customer has many orders.



Related nested report lists all the orders for the current customer

What you see in the workspace

In the workspace, you see everything in the base report plus a box that represents the related nested report.



What's the difference?

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.

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 618.

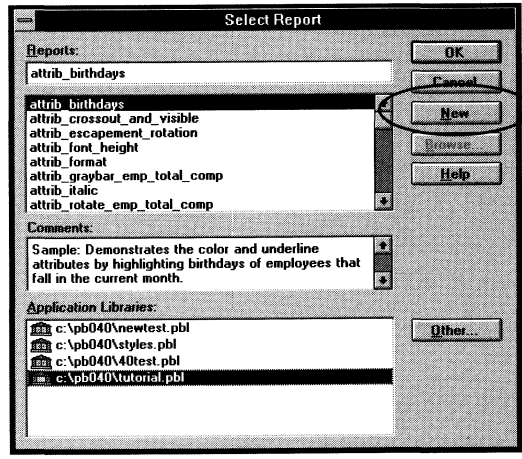
Creating a report using the Composite presentation style

❖ To create a report using the Composite presentation style:

1 Click the Report button in the PowerBar or the PowerPanel.

> **Customizing the PowerBar**
> If the Report button is not on your PowerBar, you can add it.
>
> For more information, see "Customizing toolbars" in Chapter 1, "The World of PowerBuilder."

The Select Report dialog box displays. It lists reports (DataWindow objects) in the current library.

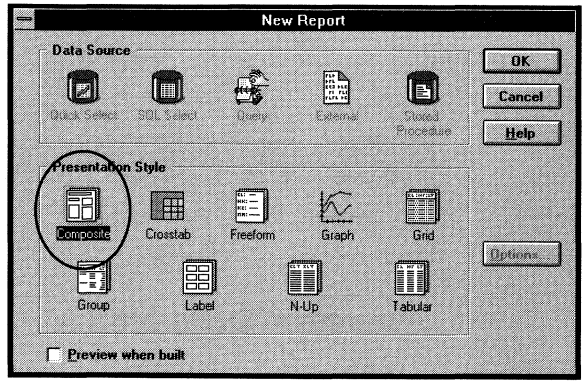


2 Click the New button.

The New Report dialog box displays.

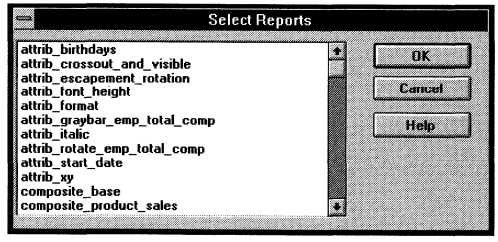
3 Click the Composite presentation style.

Notice that all the data sources become unavailable (grayed out) when you click Composite. Composite reports do not have data sources. They include other reports, which have their own data sources.

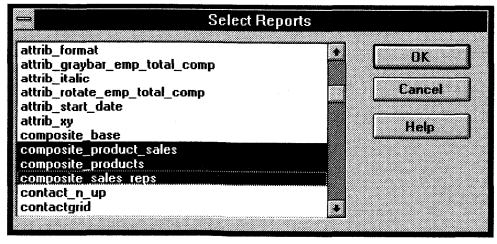


4 Click OK in the New Report dialog box.

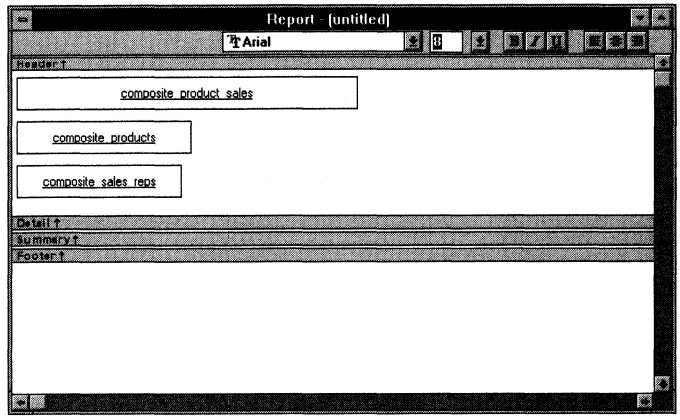
The Select Reports dialog box displays listing all reports (DataWindows) in the libraries that are in the current application's library search path.



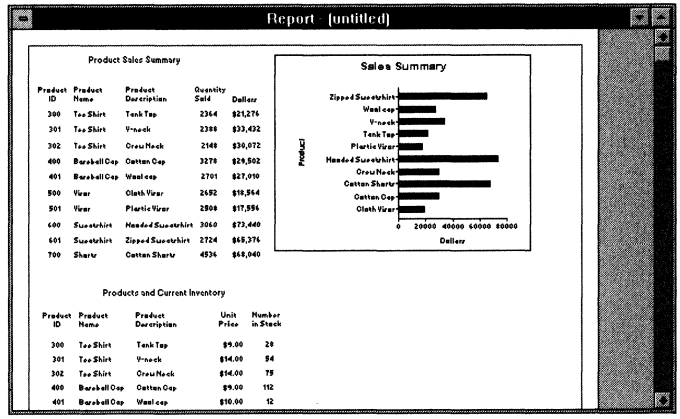
5 Click the reports you want to include in the composite report and then click OK.



6 PowerBuilder places boxes for the selected reports in the workspace. In this example, you see three reports.



7 Click the Preview button to see what your report looks like.

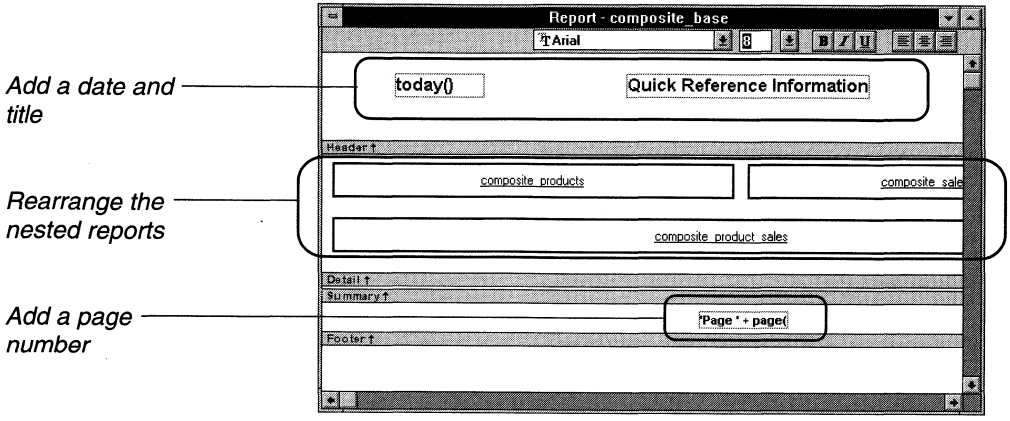


Notice that when you preview reports, you go to print preview (which is read-only) instead of standard preview, where you can update data in an updatable DataWindow object.

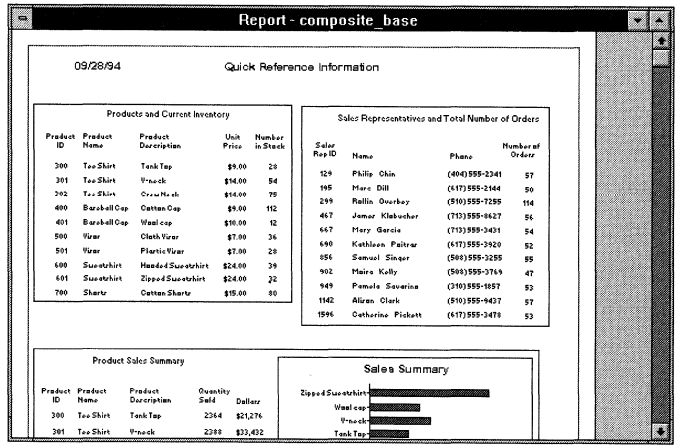
Previewing composite reports is different from previewing other types of reports

The Rows➤Filter, Rows➤Import, and Rows➤Sort menu items are grayed when you preview a composite report. You cannot choose these options. If you want to use any of these options, you can go back to the workspace and access the nested report(s), where these options are available in preview.

8 Click the Design button to return to the workspace and continue to enhance the composite report. The following picture of the workspace shows some enhancements to the example.



This is the enhanced report in preview.



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:

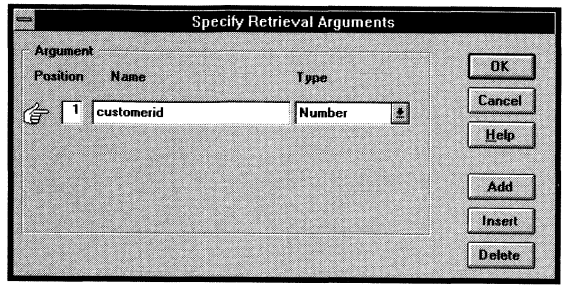
1 Create the nested report (DataWindow object) that you plan to place in the base report.

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:

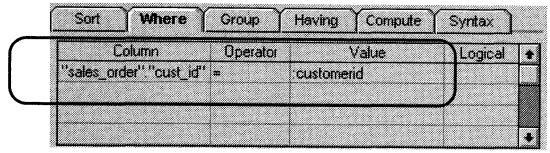
1 Select Objects➤Retrieval Arguments from the menu bar in the Select painter.

2 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.

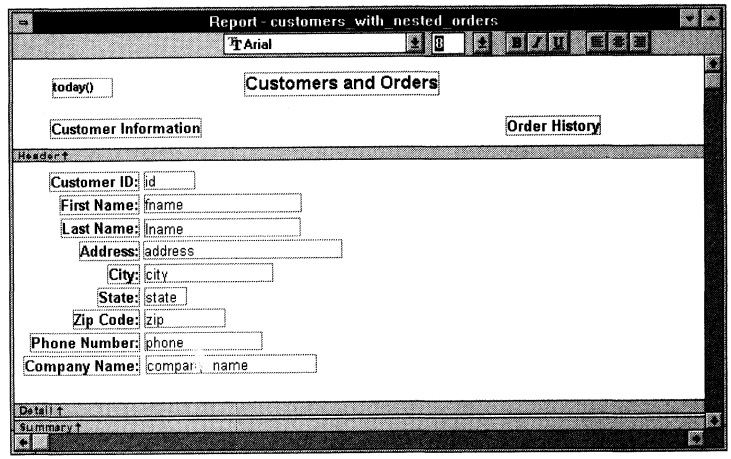
The WHERE clause in this example tells the DBMS to retrieve rows where the value in the column **cust_id** equals the value of the argument **:customerid**.



At this point, when you preview (run) the report, 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 be the base report.

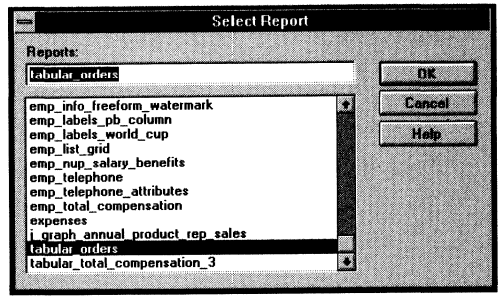
In the example, the base report is one that lists customers and has a place for the order history of each customer.



5 Click the Report button *in the PainterBar* (not the PowerBar or PowerPanel).
or
Select Objects➤Report from the menu bar.

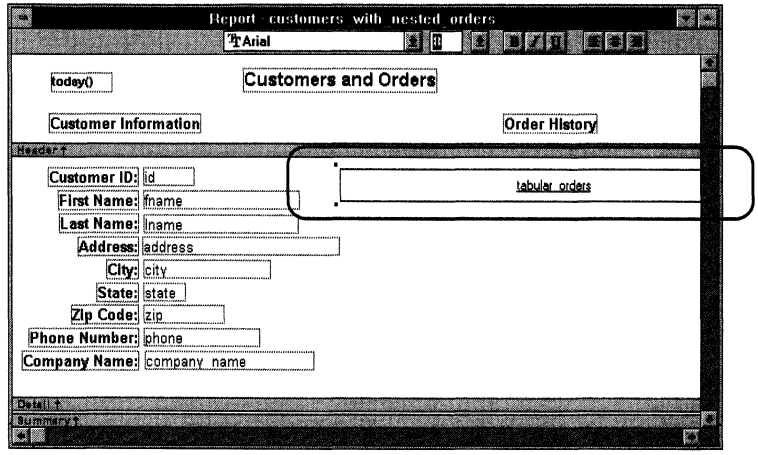
6 Click in the workspace where you want to place the report.

The Select Report dialog box displays. It lists reports (DataWindows) in the current application's search path.



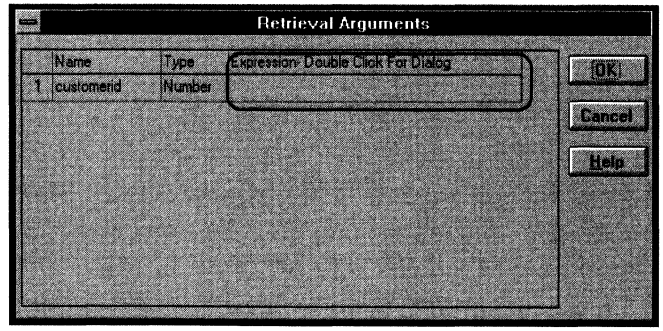
7 Select the report you want and click OK.

A box representing the report displays in the workspace, as shown in this example.



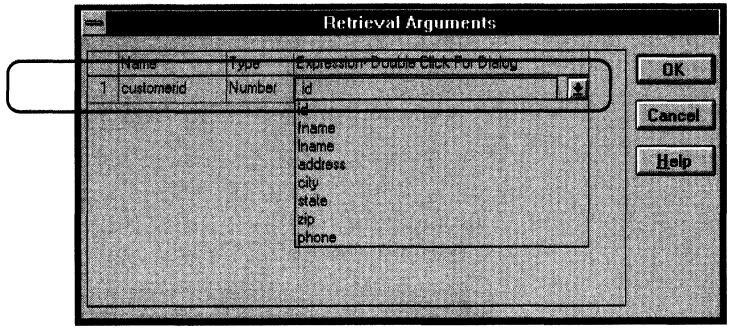
8 Display the popup menu for the nested report and select Retrieval Arguments.

The Retrieval Arguments dialog box displays. It lists arguments defined for the nested report and provides a place for you to specify how information from the base report will be used to supply the value of the argument to the nested report.

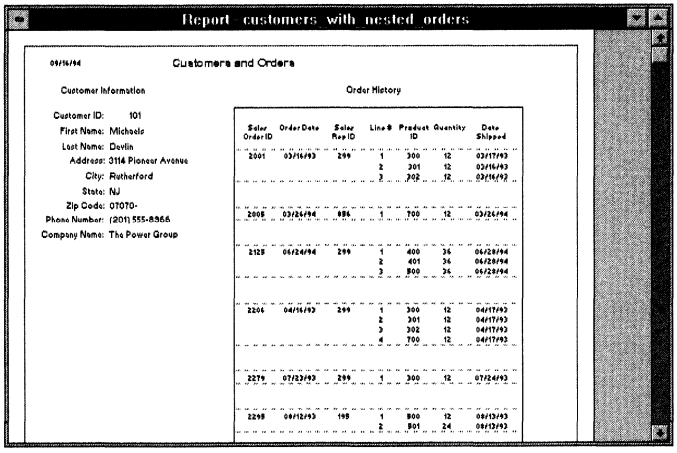


9 Enter the source of the value for the argument and click OK. You can double-click to enter an expression that will be used to supply a value for the argument.

In the example, the column named **id** from the base report will supply the value for the argument **:customerid** in the nested report.



10 Click the Preview button to see what your report looks like.



For more about what you can do while previewing, see Chapter 14, "Enhancing DataWindow Objects."

11 Click the Design button to return to the workspace and continue to enhance the report.

Placing an unrelated nested report in another report

When you place an unrelated nested report in a base report, the entire nested report will appear with *each* row of the base report.

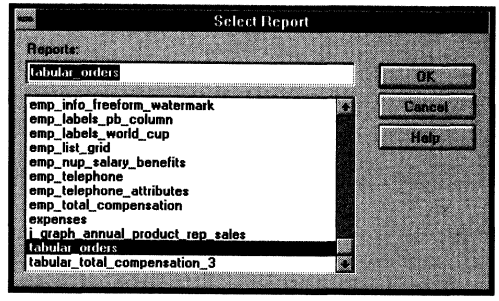
❖ To place an unrelated nested report in another report:

1 Open the report you want to be the base report.

2 Click the Report button *in the PainterBar* (not the PowerBar or PowerPanel).
or
Select Objects➤Report from the menu bar.

3 Click in the workspace where you want to place the report.

The Select Report dialog box displays. It lists reports (DataWindows) in the current application's search path.



4 Select the report you want and click OK.

A box representing the report displays in the workspace.

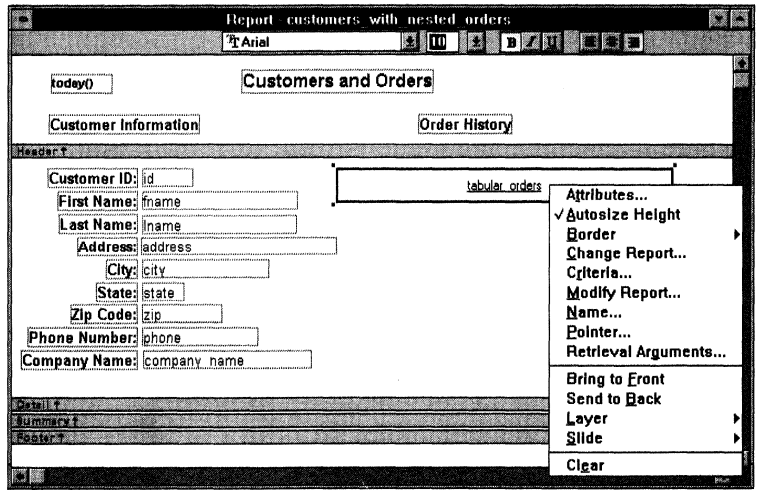
5 Click the Preview button to see what your report looks like.

For more about what you can do while previewing, see Chapter 14, "Enhancing DataWindow Objects."

6 Click the Design button to return to the workspace and continue enhancing the report.

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 display the popup menu for the nested report. The following screen shows the popup menu for a related nested report.



Many of the options on the popup menu are described in Chapter 14, "Enhancing DataWindow Objects," not here. For example, using borders on nested reports is like using borders on any object.

This section describes activities that apply only to nested reports or that have special meaning for nested reports.

Activity	See page
Using the Start On New Page option (composite only)	622
Using the Trail the Footer option (composite only)	622

Adjusting nested report width

When you preview a report with nested reports, you may find that the width of the nested report is 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.

❖ **To adjust report width:**

1 In the workspace, 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 Click the Preview button to check the new width.

Using the Autosize Height option for nested reports

Autosize Height must 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 the detail band also has an Autosize Height option. The option is on by default and must be on for the Autosize Height option for the nested report to work properly.

❖ To change the Autosize Height option:

- Select Autosize Height from the popup menu for the nested report.

If you put a nested report in a band other than the detail band
Bands other than the detail band do not have an Autosize Height option. If you place a nested report in a band other than the detail band, you must size the band itself to hold the report.

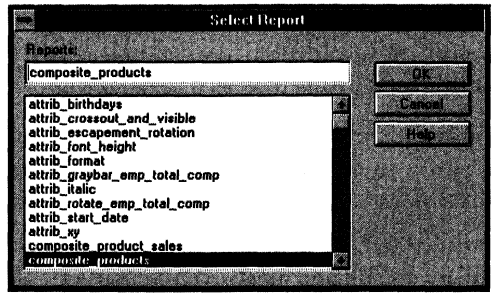
Changing a nested report from one report to another

You may want to 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:

1. Position the pointer on the nested report you want to change and display the popup menu.
2. Select Change Report from the popup menu.

   The Select Report dialog box displays. It lists all reports (DataWindows) in the current application's search path.



3. Select the report you want to use and click OK.

   The name of the report that displays in the box in the workspace changes to the new one.

Modifying the contents of a nested report

While you are working with nested reports, you may want to modify the contents of the nested report. You can do this directly from the composite report or base report that contains the nested report.

❖ To modify the contents of a nested report from the composite report or base report:

1 Position the pointer on the nested report whose contents you want to modify and display the popup menu.

2 Select Modify Report from the popup menu.

The nested report opens and displays in the Report painter (or DataWindow painter) workspace. Both the composite or base report and the nested report are now 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 may want to add another report.

For information on adding a nested report to a report (not a composite report), see "Placing a related nested report in another report" on page 608 or "Placing an unrelated nested report in another report" on page 613.

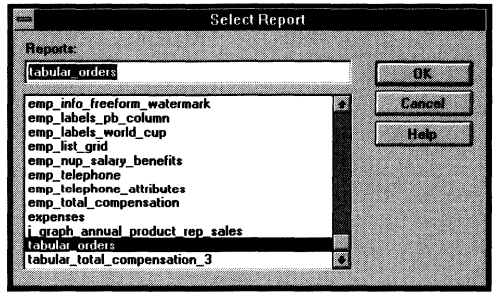
❖ To add another nested report to a composite report:

1 Open the composite report.

2 Click the Report button *in the PainterBar* (not the PowerBar or PowerPanel).
or
Select Objects➤Report from the menu bar.

3 Click in the workspace where you want to place the report.

The Select Report dialog box displays. It lists all reports (DataWindows) in the current application's search path.



4 Select the report you want and click OK.

A box representing the report displays in the workspace.

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 retrieval arguments defined, you use the process described in this section to supply the retrieval argument value from the base report to the nested report. (The process described is part of the whole process covered in "Placing a related nested report in another report" on page 608.)

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:

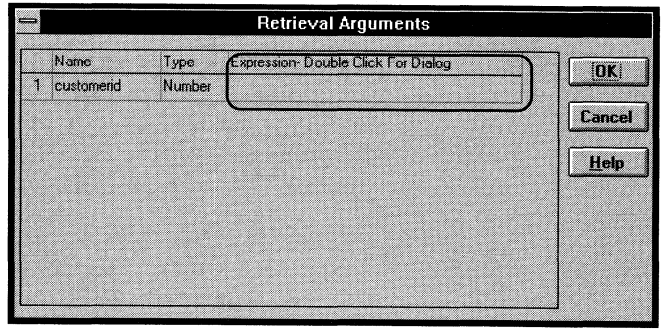
- The database interface must support SQL caching.
- You must enable caching in the database profile. You should set the SQLCache DBParm parameter to the number of levels of nesting plus 5.

  For more information, see the description of the SQLCache DBParm parameter in *Connecting to Your Database*.

❖ To supply a retrieval argument value from the base report to the nested report:

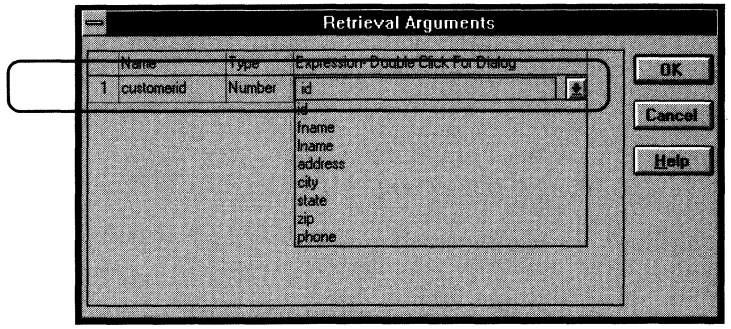
1. Make sure that the nested report has been set up to take one or more retrieval arguments.
2. Display the popup menu for the nested report and select Retrieval Arguments.

   The Retrieval Arguments dialog box displays. It lists arguments defined for the nested report and provides a place for you to specify how information from the base report will supply the value of the argument to the nested report.



3 Enter the source of the value for the argument and click OK. You can click the dropdown arrow to display a list of the columns in the base report. You can double-click to enter an expression that will supply a value for the argument.

In the example, the column named **id** from the base report will supply the value for the argument **:customerid** in the nested report.



You return to the workspace. When you run the report now, you will not be prompted for retrieval argument values for the nested report. The base report will supply 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.

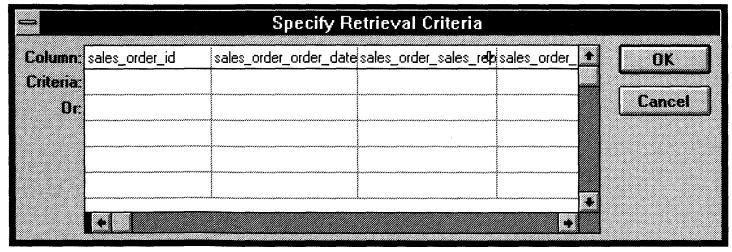
How the DBMS processes SQL if you use the specify criteria technique

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 Display the popup menu for the nested report and select Criteria.

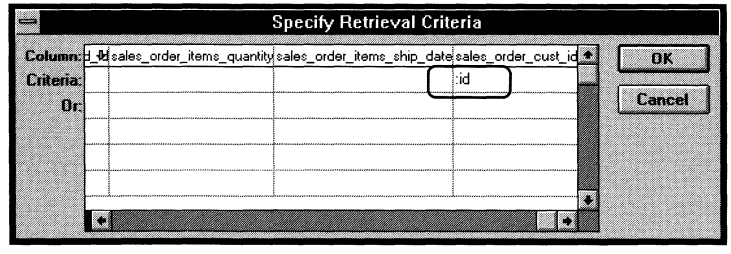
The Specify Retrieval Criteria dialog box displays. It lists all the columns retrieved for the nested report and provides a place for you to specify how information from the base report will supply the retrieval criteria to the nested report.



2 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 criteria line are ANDed together. Criteria entered on separate lines are ORed together.

In this example, the customer ID (the **id** column) is the retrieval criteria being supplied to the nested report. Notice that the id column is preceded by a colon (:), which is required because it serves as a SQL host variable.



For more information about specifying retrieval arguments, see Chapter 13, "Defining DataWindow Objects."

You return to the workspace. 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 the Slide option for a nested report

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 reports and off by default for all others. Having Slide Left on for grid and crosstab reports ensures that they break horizontally on whole columns, not in the middle of a column.
- The Slide Up All Above/Up 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 objects to remove blank space" in Chapter 14, "Enhancing DataWindow Objects."

Using the Start On New Page option (composite only)

The Start On 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:

- Select Start On New Page from the popup menu for the nested report.

  A checkmark displays next to the option indicating the option is selected.

Using the Trail the Footer option (composite only)

The Trail the 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, not at the bottom of the page.

❖ To specify that the footer should appear at the bottom of the page:

- Select Trail the Footer from the popup menu for the nested report.

  The checkmark next to the option disappears, indicating the option is no longer selected. The footer will appear 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.

Using nested reports in an application

When you have completed the report that contains the nested reports, you associate the report with a DataWindow control in a window, resize the control as needed, then write scripts to manipulate the control. Minimally, you will use SetTransObject to associate the control with a transaction object and Retrieve to retrieve data into the report.

For general information about using DataWindow objects (reports) in applications, see *Building Applications*.

Following is information specifically about using nested reports in an application.

Printing multiple updatable DataWindows on a page

One of the advantages of composite reports is that you can print multiple reports (DataWindows) on a page. One of the limitations of composite reports is that they are not updatable, so you cannot *directly* print several updatable DataWindows on one page. However, there is an *indirect* way to do that, as follows.

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

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

❖ To print multiple DataWindows on a page:

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

Naming the nested report
To use GetChild on a nested report, the nested report must have a name. To name a nested report, double-click it in the workspace and enter a name in the Report Name dialog box.

3 Add the composite report to the window (it can be hidden).

4 In your application, do the following:

1 Retrieve data into the report.

2 Use GetChild to get a reference to the nested reports in the composite report.

3 Use ShareData to share data between the updatable DataWindows and the nested reports.

4 When appropriate, print the composite report.

The report contains the information from the updatable DataWindows.

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

For more about GetChild and ShareData, see the *Function Reference*.

Creating and destroying nested reports during execution

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

Creating nested reports

To create a nested report, use the CREATE keyword with the Modify function. When creating the nested report, supply the appropriate values for the nested report's attributes.

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

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

Destroying nested reports

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

For more information

For more about creating and destroying objects in a DataWindow (report), see the chapter on dynamic DataWindow objects in *Building Applications*.

For a list of attributes of nested reports, see the chapter on DataWindow object attributes in the *Function Reference*.

For more about exporting object syntax, see Chapter 23, "Managing Libraries."

Running nested reports in a PSR file

When you save a report that contains nested reports to a PSR (Powersoft Report) file, the main report definition and all of the data is saved, but the definitions of the nested reports are not saved. Only the names of the nested reports are saved. This means that in order to run the PSR file, you need to make sure that the nested reports themselves are available in the application's library search path.

For more information about PSR files, see Chapter 14, "Enhancing DataWindow Objects."

CHAPTER 18

Working with Graphs

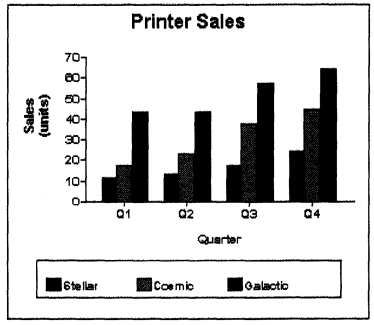
About this chapter

This chapter describes how to build and use graphs in applications.

Contents

Overview of graphs

Often the best way to display information is graphically. Instead of showing your user 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.



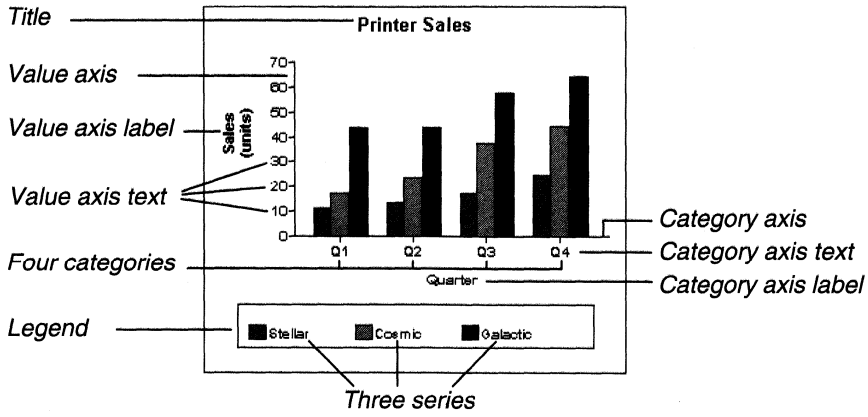
PowerBuilder provides many types of graphs and allows you to customize your graphs in many ways. Probably most of your use of graphs will be in a DataWindow object—the source of the data for your graphs will be the database. You can also use graphs as standalone controls in windows (and user objects) 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. But the way you manipulate graphs is different in a DataWindow than 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. (The data comes from the Printer table, which is in the Powersoft Demo Database.)



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:

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).

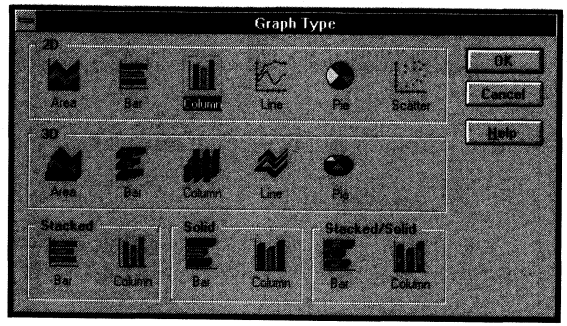
Organization of a graph

The following table lists the parts of a typical 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. But 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.
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

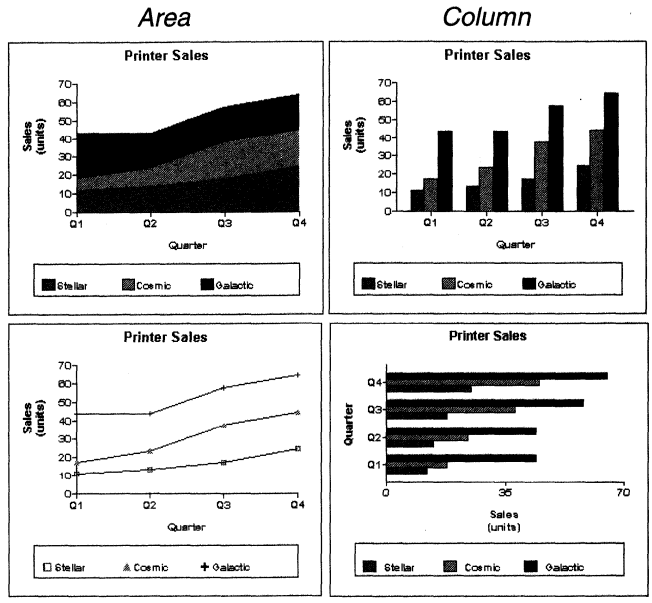
PowerBuilder provides many graph types, which are divided into groups. You choose a graph's type from the Graph Type dialog box:



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.

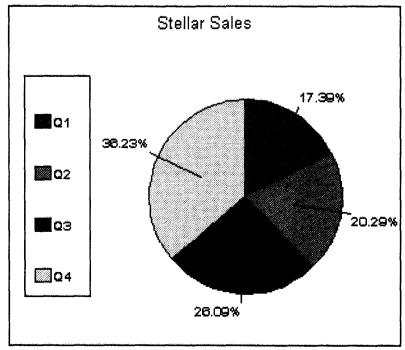
Following are the graph types, all showing the same data. The *only* difference between these graphs is the type; all other attributes of the graphs (such as the series, categories, data values, and legends) are identical.



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

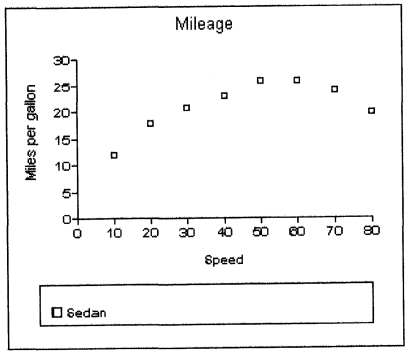
Scatter graphs show XY data points. Typically you use scatter graphs to show the relationship between two sets of numeric values.

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

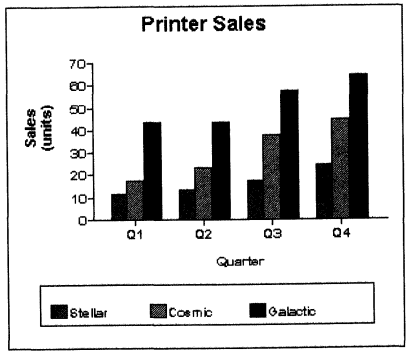
Here is the data in a scatter graph:



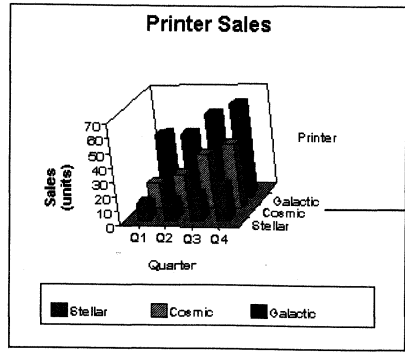
You can have multiple series of data in a scatter graph. In the above example, you might want to plot mileage versus speed for several makes of cars in the same graph.

Three-dimensional graphs

You can also create three-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. For example, here is a two-dimensional graph:



Here is the same graph, this time specified as 3D (the only difference between this graph and the preceding one is the type):



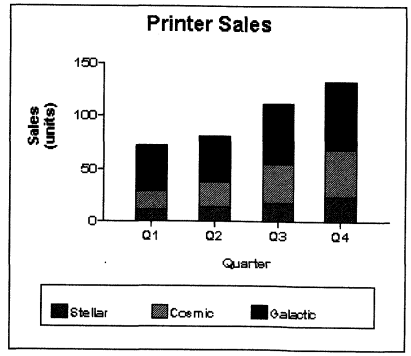
The three series are plotted along the Series axis, instead of being next to each other along the Category axis.

In 3D graphs, 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.

Here is the stacked version of the preceding two-dimensional column chart:



Each category has one column, which is divided into the different 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 during execution or you can place a graph in a band and prevent movement. In windows, graphs are placed like all other window controls.

What's ahead

Section	Page	Covers
"Using graphs in DataWindow objects"	637	Topics specific to graphs in a DataWindow, such as how to place a graph in a DataWindow object and how to specify the columns in your database that will populate the graph during execution
"Defining a graph's attributes"	655	How to define the basic properties of graphs, such as the graph's name, type, legend location, and text properties
"Using graphs in windows"	672	Topics unique to graphs in a window, such how to write scripts to populate the graph during execution
"Accessing graphs at execution time"	677	How to access and modify graph attributes during execution

Using graphs in DataWindow objects

You will probably use most of your graphs in DataWindow objects—the data for the graph comes from tables in the database.

Graphs in DataWindows are dynamic

DataWindow graphs are tied directly to the data that is in the DataWindow. 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 an object in a DataWindow object

  Here you use a graph to enhance the display of information in a DataWindow object, such as a tabular or freeform DataWindow. This technique is described next.

- By using the Graph presentation style

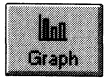
  Here the entire DataWindow object is a graph. The user doesn't see the underlying data. This technique is described in "Using the Graph presentation style" on page 653.

Placing a graph in a DataWindow

❖ **To place a graph in a DataWindow object:**

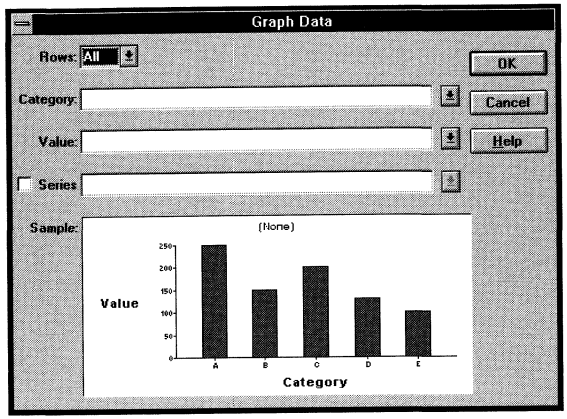
1. Open the DataWindow painter and select the DataWindow object that will contain the graph.

   The DataWindow painter workspace displays the DataWindow object.



2. Click the Graph button in the PainterBar.
3. Click where you want the graph.

PowerBuilder displays the Graph Data window.



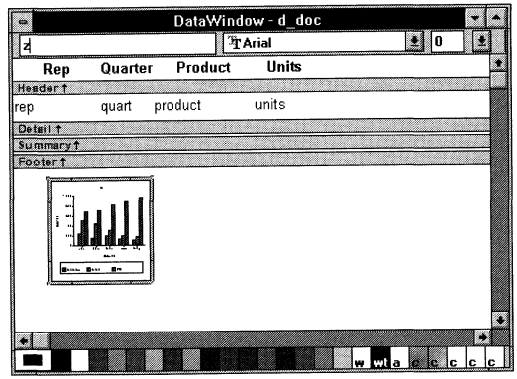
Here you specify which columns contain the data for the graph.

4 Specify which columns contain the data now, or leave the boxes empty and do it later.

For more information, see "Associating data with a graph" on page 641.

5 Click OK.

You return to the painter workspace with a representation of the graph in place.



6 Specify the graph's attributes.

Using popup menus

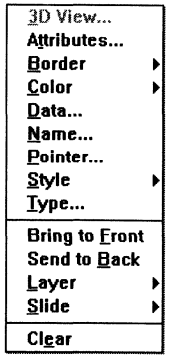
Like other controls, graphs have popup menus from which you can specify attributes. The graph itself has a popup menu as does each of the following parts of a graph:

- The title
- Axes
- Axis labels
- The legend

❖ To display a popup menu:

- Position the mouse pointer over part of the graph and display the popup menu as usual. To display the popup menu for the graph as a whole, position the mouse pointer in the body of the graph (where the data is being represented).

  A popup menu specific to the selected part of the graph displays. Here is the popup menu for a graph in the DataWindow painter:



Menu item	Action
3D View	Defines the perspective used in 3D graphs (is grayed if you are not working with a 3D graph)
Attributes	Specifies an expression that is evaluated at execution time to determine the value of an attribute of the graph
Border	Specifies the type of border around the graph
Color	Specifies the color for the graph's background and text
Data	Specifies where to get the graph's data
Name	Specifies the graph's name, title, type, and other properties
Pointer	Specifies the pointer to use when the mouse is positioned over the graph
Style	Specifies general properties for the graph
Type	Specifies the type of graph
Bring to Front	Places the graph in front of other elements
Send to Back	Places the graph behind other elements

Menu item	Action
Layer	Specifies how to position the graph in the DataWindow object
Slide	Specifies whether the graph object in the DataWindow will slide up or to the left to fill blank space
Clear	Removes the graph from the DataWindow

Changing a graph's position

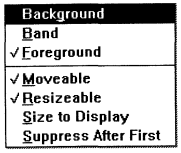
When you initially place a graph in a DataWindow object, it is in the foreground—it sits above the bands in the DataWindow. Unless you change this setting, the graph displays in front of any retrieved data.

The initial graph is also movable and resizable, so users have complete flexibility as to the size and location of a graph at execution time.

You can change these attributes if you want.

❖ To specify a graph's position:

1. Position the mouse pointer over the body of the graph and display the graph's popup menu.
2. Select Layer and display the cascading menu.



3. Select the settings you want for the graph:

Setting	Meaning
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 you scroll through rows, 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 won't obscure data during execution.

Setting	Meaning
Moveable	The user can move the graph during execution.
Resizeable	The user can resize the graph during execution.
Size to Display	The graph fills the DataWindow control and resizes when the DataWindow control is resized during execution.
Suppress After First	Don't repeat graph after the first column in a DataWindow using newspaper-style columns For more about newspaper-style columns in a DataWindow, see *Building Applications*.

Associating data with a graph

When using a graph in a DataWindow object, you associate axes of the graph with columns in the DataWindow.

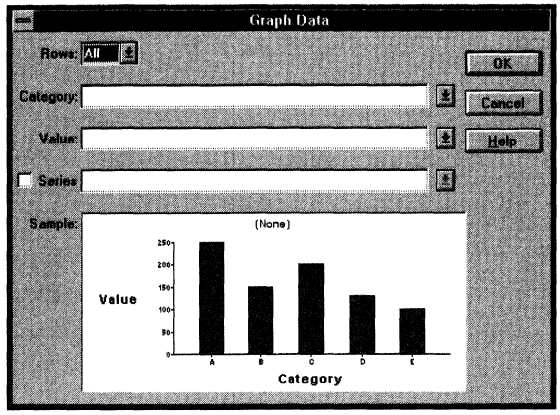
In fact, the only way to get data into a DataWindow graph is through columns in the DataWindow. You cannot add, modify, or delete data in the graph except by adding, modifying, or deleting data in the DataWindow.

You can graph data from any columns retrieved into the DataWindow object. The columns don't have to be displayed.

About the examples
The process of specifying data for a graph is illustrated below using the Printer table in the Powersoft Demo Database.

❖ To specify data for a graph:

1 If you have just placed a graph in the DataWindow painter, the Graph Data dialog box is displayed. Otherwise, select Data from the graph's popup menu.



2 Fill in the boxes as described below.

3 Click OK.

Specifying which rows to include in a graph

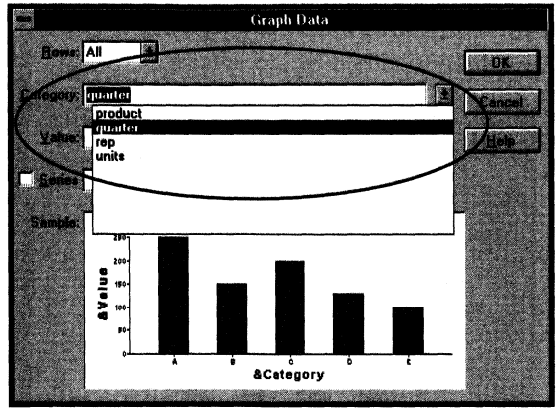
The Rows dropdown listbox allows you to specify which rows of data are graphed at any one time.

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)
Page	Graphs only the data from the rows that are currently displayed on the screen
Group *n*	Graphs only the data in the specified group (in a grouped DataWindow)

If you select Group
If you are graphing data in the current group in a grouped DataWindow object and might have several groups displayed at the same time, you should localize the graph in a group-related band in the workspace to ensure that it is clear which group the graph represents. Usually the group header band is the most appropriate band.

Specifying the categories

Specify the column or expression whose values determine the categories. You can select a column name from the dropdown listbox or type an expression.



There will be an entry along the Category axis for each different value of the column or expression you specify here.

Using display values of data

If you are graphing columns that use code tables—data is stored with a data value but displayed to the user with more meaningful display values—by default the graph will use the column's data values. To have the graph use a column's display values, use the LookupDisplay function when specifying Category or Series. LookupDisplay returns a string that matches the display value for a column:

LookupDisplay (*column*)

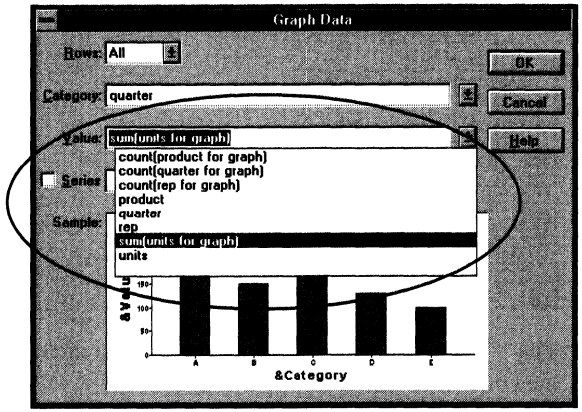
For more about code tables, see "Defining a code table" in Chapter 15, "Displaying and Validating Data."

For more about LookupDisplay, see the *Function Reference.*

Specifying the values

PowerBuilder populates the Value dropdown listbox in the Graph Data dialog box with the names of all the columns as well as the following aggregate functions:

- Count for all non-numeric columns
- Sum for all numeric columns



Select an item from the dropdown listbox or type an expression. 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.

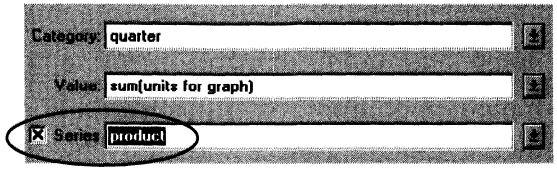
Single-series graphs

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 checkbox and specify the column that will provide the series values.

You can select column names from the dropdown listbox.



There will be 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. 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 need to 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 "Graphing actual and projected sales" on page 650 illustrates this technique.

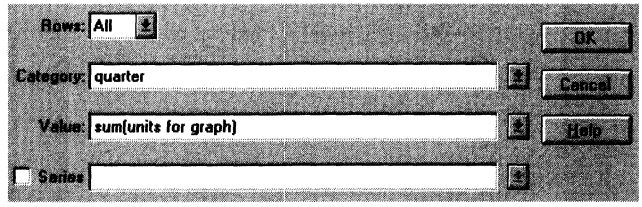
Examples

This section shows how to specify the data for several different graphs of the data in the Printer table in the Powersoft Demo Database. The table records quarterly unit sales of three printers by three sales representatives:

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

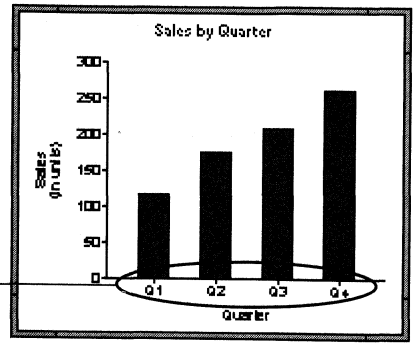
Rep	Quarter	Product	Units
Perez	Q1	Galactic	1
...	...	...	...
Simpson	Q4	Stellar	30
Jones	Q4	Stellar	24
Perez	Q4	Stellar	36
Simpson	Q4	Cosmic	60
Jones	Q4	Cosmic	52
Perez	Q4	Cosmic	48
Simpson	Q4	Galactic	3
Jones	Q4	Galactic	3
Perez	Q4	Galactic	6

Graphing total sales Say you want to graph total sales of printers in each quarter. Retrieve all the columns into a DataWindow object and create a graph with the following data specification:



You want to graph sales by quarter, so 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 as done in the preceding dialog box, or type a string literal to identify the series in a legend. You want to graph total sales in each quarter, so the Value box is specified as sum(units for graph).

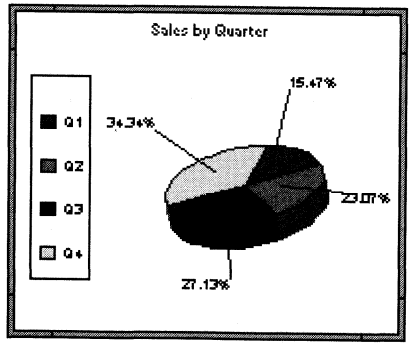
Here is the resulting column graph. PowerBuilder automatically generates the category text based on the data in the table.



Four categories

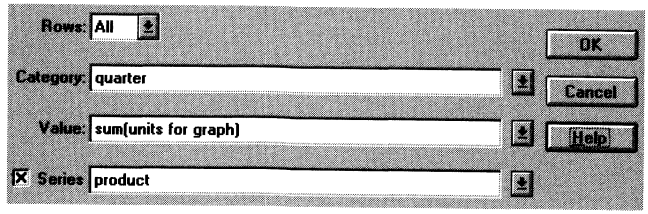
In the preceding graph, there is one set of data points (one series) across four quarters (the category values).

The following is a pie graph, which has exactly the same attributes as the column graph above except for the type, which is 3D Pie:



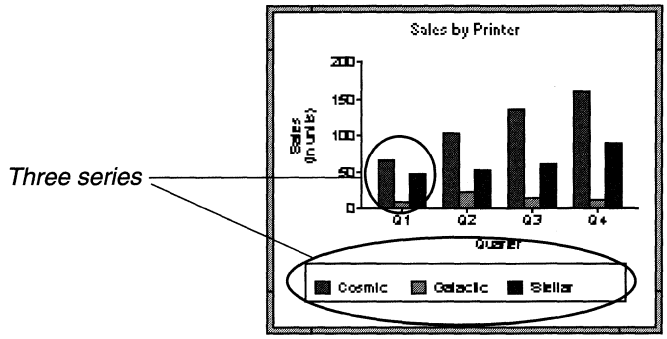
In pie graphs, categories are shown in the legend.

Graphing unit sales of each printer Say you want to graph total quarterly sales of each printer. Create a graph with the following data specification:

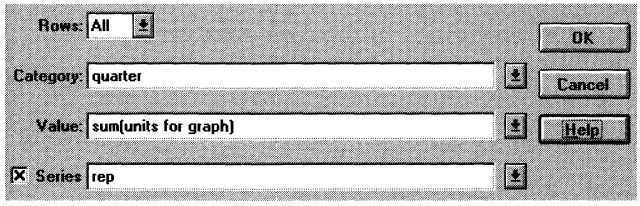


You want a different series for each printer, so the column Product serves as the series. Because the Product column has three values (Cosmic, Galactic, and Stellar), there will be three series in the graph. As in the first example, you want a value for each quarter, so the Quarter column serves as the category. And you want to graph total sales in each quarter, so the Value box is specified as sum(units for graph).

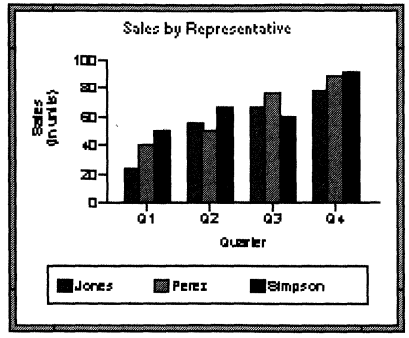
Here is the resulting graph. PowerBuilder automatically generates the category and series labels based on the data in the table. The series labels display in the graph's legend.



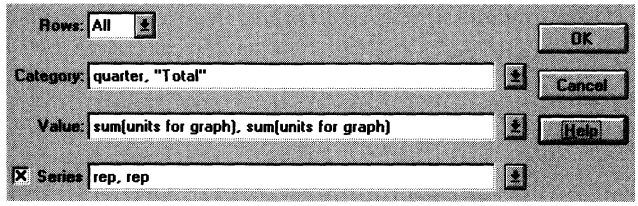
Graphing unit sales by representative Say you want to graph the quarterly sales made by each representative. Here you want a different series for each representative, so the column Rep serves as the series. Your category is still the Quarter column. And you still want to graph total sales in each quarter for each series, so the Value box is still sum(units for graph).



Here is the resulting graph:



Graphing unit sales by representative and total sales Say you want to graph sales by representative, as shown above, plus total sales for each printer. Here is how to specify the data:



Here you have two types of categories: the first is Quarter, which shows quarterly sales, as in the preceding graph. You also want a category for total sales. There is no corresponding column in the DataWindow, so you can simply type the literal "Total" to identify the category. The Category box looks like this:

```
quarter, "Total"
```

You separate multiple entries with a comma.

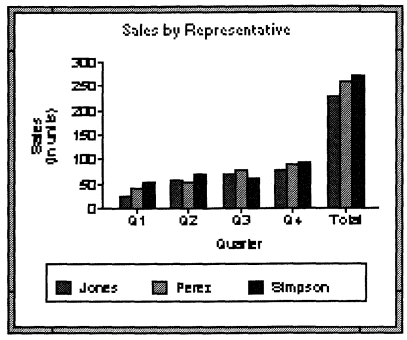
For each of these category types you want to graph the sum of units sold for each representative. So the Value entry is:

```
sum(units for graph), sum(units for graph)
```

And the Series entry is:

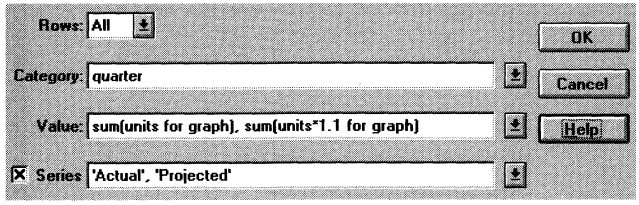
```
rep, rep
```

Here is the resulting graph:



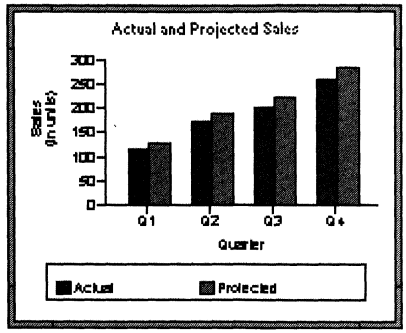
Notice that PowerBuilder uses the literal "Total" supplied in the Category box in the Graph Data window as a value in the Category axis.

Graphing actual and projected sales Say you want to graph total quarterly sales of all printers and also want to graph projected sales for next year. You figure that next year you will sell about 10 percent more printers. Here is how to specify the data:



You are using labels to identify two series, Actual and Projected. Note the 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 the series in the Graph Data window as follows:

- If specifying a column name to identify the series, specify this for the series:

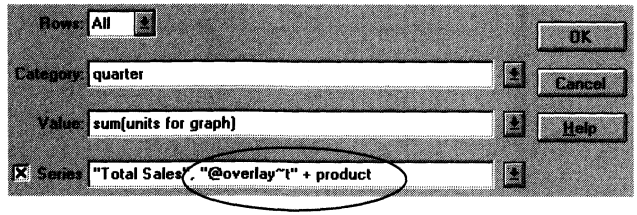
  "@overlay~t" + *ColumnName*

- If using a label to identify the series, specify this for the series:

  "@overlay~t*SeriesLabel*"

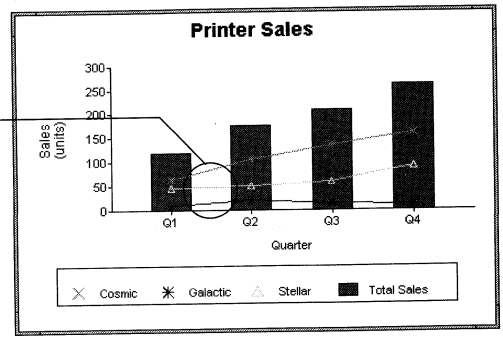
Examples

Say you want to graph sales in each quarter and overlay the sales of each individual printer. Specify the graph's data like this:

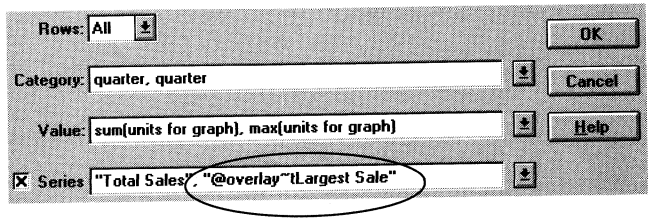


Here is the resulting graph:

Individual printer sales are overlaid as lines over total quarterly sales

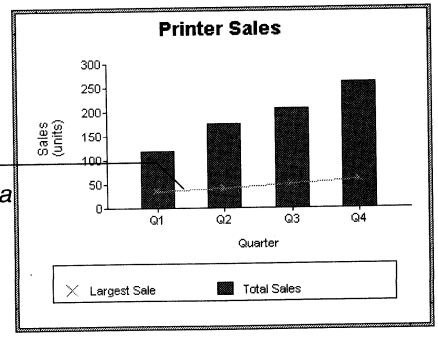


To graph unit sales of printers by quarter and overlay the largest sale made in each quarter, specify the graph's data like this:



Here is the resulting graph:

The largest sales in a quarter are overlaid as a line on the total sales



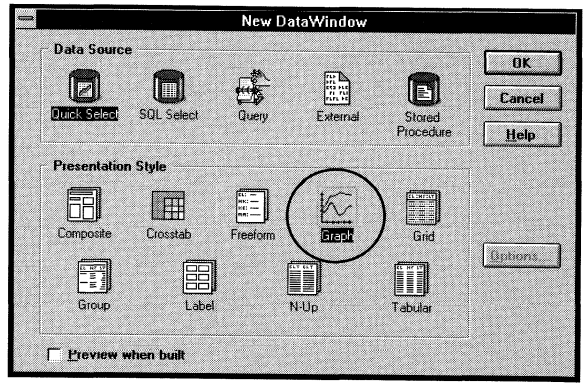
Using the Graph presentation style

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 the user resizes the DataWindow control associated with the graph DataWindow object during execution.

❖ To use the Graph presentation style:

1 Open the DataWindow painter and select New in the Select DataWindow dialog box.

The New DataWindow dialog box displays.



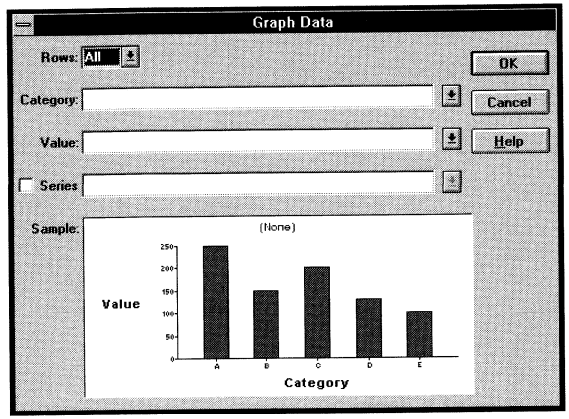
2 Select a data source and the Graph presentation style, then click OK.

You are prompted to specify the data.

3 Specify the data you want retrieved into the DataWindow object.

For more information, see Chapter 13, "Defining DataWindow Objects."

The DataWindow painter workspace displays, and you are prompted to specify the data for the graph.



4 Enter the definitions for the series, categories, and values, as described in "Associating data with a graph" on page 641. (Note that the Rows box is protected. When using the Graph presentation style, the graph always graphs all rows; you cannot specify page or group.)

5 Click OK.

A model of the graph displays in the entire workspace.

6 Specify the attributes of the graph, as described in "Defining a graph's attributes" on page 655.

7 Save the DataWindow object in a library.

8 Associate the graph DataWindow object with a DataWindow control in a window or user object.

During execution, the graph fills the entire control and resizes when the control is resized.

Defining a graph's attributes

This section describes attributes of a graph that are used regardless of whether the graph is in a DataWindow object or in a window.

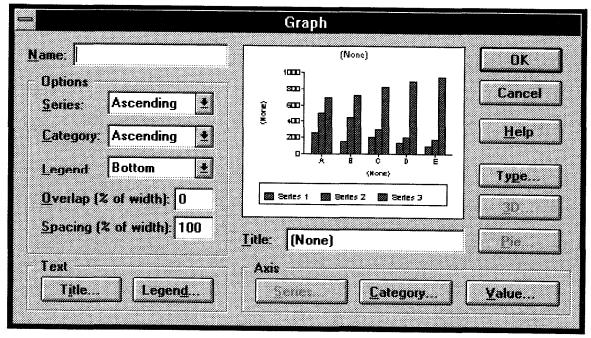
Using the Graph dialog box

One of first things you will probably do with a graph is name it and define its basic attributes. You can do that in the Graph dialog box.

❖ To open the Graph dialog box:

- Do one of the following:
  - Position the mouse pointer over the body of the graph and display the popup menu. Select Name from the popup menu.
  - Double-click the graph.

  The Graph dialog box displays.



About what is shown in the graph model

As you modify the graph's attributes, PowerBuilder updates the model graph shown in the Graph dialog box so you can get an idea of the graph's basic layout, as follows:

- PowerBuilder uses the graph title and axis labels you specify.
- PowerBuilder uses sample data (not data from your DataWindow) to illustrate series, categories, and values.

Naming a graph

You can modify graphs in scripts during execution. To reference a graph during execution, you use its name.

❖ To name a graph:

1 Open the Graph dialog box by selecting Name from the popup menu or by double-clicking the graph.

2 Assign a meaningful name to the graph.

3 Click OK.

Defining a graph's title

You specify a graph's title in the Graph dialog box. The title displays at the top of the graph.

Multiline titles
You can force a new line in a title by embedding ~n.

For information on specifying properties for the title text, see "Specifying text attributes of titles, legends, and axes" on page 658.

Specifying the type of graph

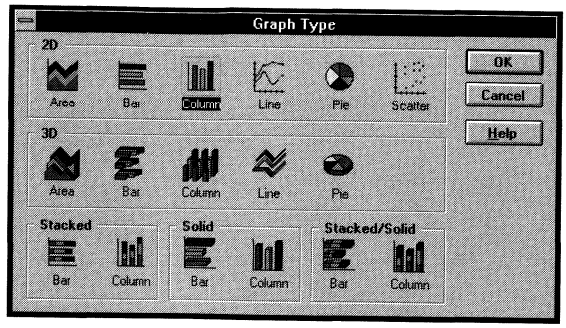
You can change the graph type anytime in the development environment (you can also change the type during execution by modifying a graph's GraphType attribute).

❖ To specify the graph type:

1 Do one of the following:

- Click the Type button in the Graph dialog box.
- Select Type from the graph's popup menu.

The Graph Type dialog box displays.



2 Click the button representing the graph type and click OK.

Using legends

A legend provides a key to your graph's series.

❖ To include a legend:

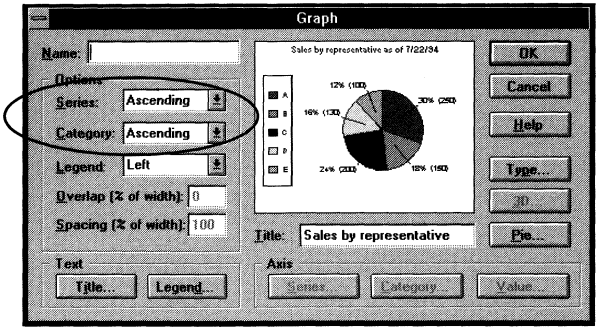
1 Display the Graph dialog box by double-clicking the graph.

2 Specify where you want the legend to appear by selecting a value in the Legend dropdown listbox.

For information on specifying text properties for the legend, see "Specifying text attributes of titles, legends, and axes" on page 658.

Sorting data

You can specify how to sort the data for series and categories. By default, the data is sorted in ascending order.



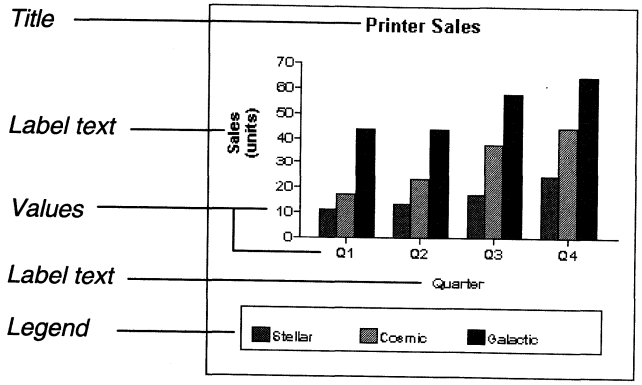
You can choose unsorted, sort in ascending order, or sort in descending order.

For upgraders
The ability to specify sorting of series and categories is new in Version 4.0. When you migrate from PowerBuilder Version 3.0 to 4.0, your graphs will be set to sort in ascending order, so they might display differently than in Version 3.0. If you don't want your data sorted, you need to display the Graph dialog box and specify Unsorted for the axes.

Specifying text attributes of titles, legends, and axes

A graph can have four text elements:

- Title
- Labels for the axes
- Text that shows the values along the axes
- Legend



You can specify attributes for each text element. You can do this using popup menus or from the Graph dialog box.

Using popup menus

❖ To specify text attributes using a popup menu:

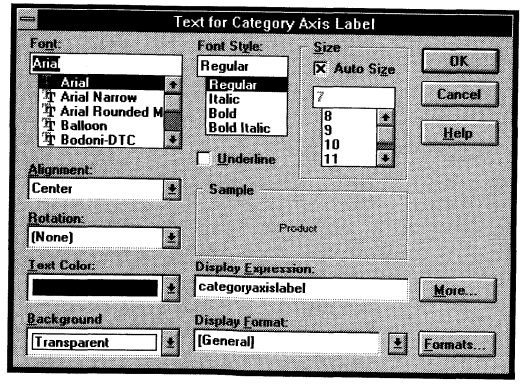
1 Position the mouse pointer over a text element in the model graph displayed in the workspace.

2 Display the popup menu.

A popup menu specific to the text element displays.

3 Select Font from the menu.

A Text For *Element* dialog box displays. The title bar indicates the text element you are modifying. Other than the title bar, all the Text For *Element* windows are the same.



4 Specify the font and its characteristics.

Using the Graph dialog box

❖ **To specify text attributes using the Graph dialog box:**

1 Do one of the following:

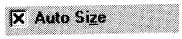
To modify attributes for	Do this
The title	Click the Title button in the Graph dialog box.
The legend	Click the Legend button in the Graph dialog box.
Labels for an axis	Display the Axis window by clicking the appropriate button in the Graph dialog box. Then click the Label Text button.
The text that shows values along an axis	Display the Axis window by clicking the appropriate button in the Graph dialog box. Then click the Text button.
Pie graph labels	Click the Pie button in the Graph dialog box.

The appropriate Text For *Element* dialog box displays.

2 Specify the font and its characteristics.

Using Auto Size

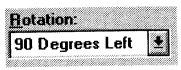
To have PowerBuilder automatically size text, select the Auto Size checkbox (this is the default) in the Text For *Element* dialog box. With Auto Size in effect, PowerBuilder resizes the text appropriately whenever the graph is resized.



To specify a font size, deselect Auto Size, then select the size in the Size dropdown listbox.

Rotating text

For all the text elements, you can specify the number of degrees by which you want to rotate the text in the Text For *Element* dialog box.



Changes you make here are shown in the model graph in the workspace and Graph dialog box.

Using display formats

You can choose an existing display format for the text from the Display Format dropdown listbox in the Text For *Element* dialog box, or define a new display format by clicking the Formats button.

The formats listed in the dropdown listbox are the ones appropriate for the data type of the text element.

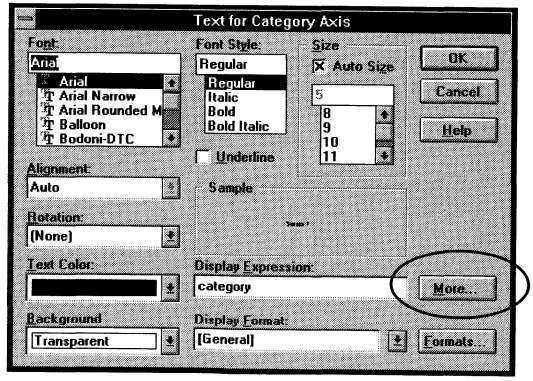
Modifying display expressions

You can specify an expression for the text that is used for each graph element. The expression is evaluated at execution time.

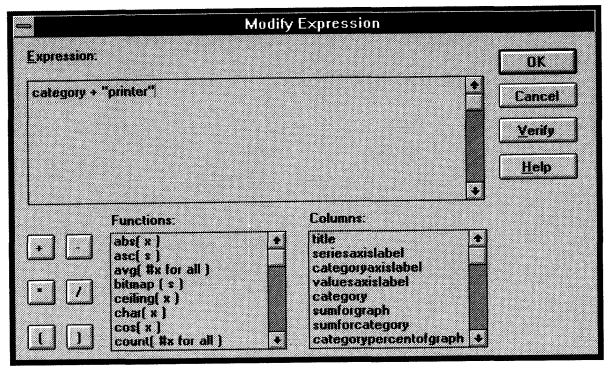
❖ To specify an expression for text:

1 Display the Text For *Element* dialog box for the text element, as described above.

2 Click the More button.



The Modify Expression dialog box displays.



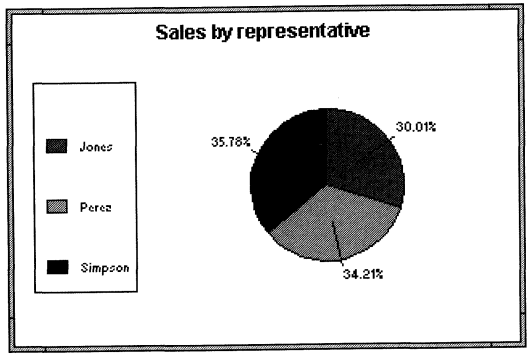
3 Specify the expression. You can paste DataWindow functions, column names, and operators. Included with column names in the Columns box are statistics about the columns, such as counts and sums.

4 Click OK.

Example

Here's an example of using expressions to enhance a graph.

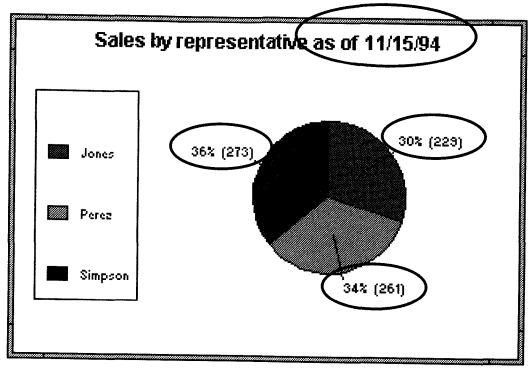
By default, when you generate a pie graph, PowerBuilder puts the title at the top and labels each slice of the pie with the percentage each slice is of the whole, out to two decimal places.



The following graph has been enhanced as follows:

- The current date displays in the title
- The percentages are rounded to integers

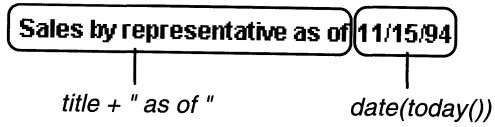
- The raw data for each slice is shown in addition to the percentages.



To accomplish this, the display expressions were modified for the title and pie graph labels.

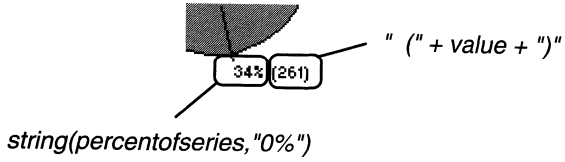
For the title:

Original expression	Modified expression
title	title + " as of " + date(today())



For the pie graph labels:

Original expression	Modified expression
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 following attributes:

Attribute	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.

You can specify these properties in the Style choice of the popup menu or in the Graph dialog box.

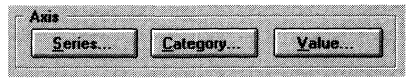
Specifying axis properties

Graphs have two or three axes. You specify the axes' properties from the Axis dialog box.

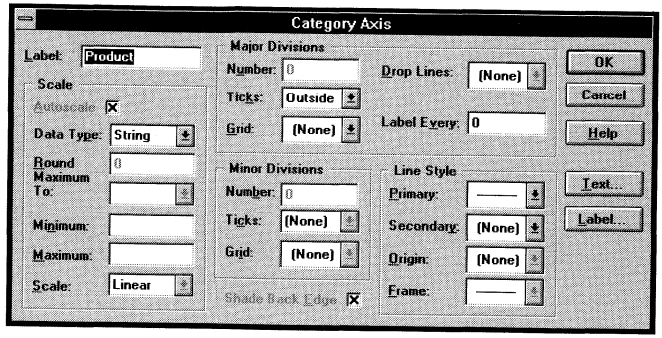
❖ To specify properties for an axis:

1 Do one of the following:

- Position the mouse pointer over the axis you want to modify in the model graph in the workspace and display the popup menu. Select Axis from the menu.
- Open the Graph dialog box and click the appropriate button. (The Series axis button is disabled if you are not working with a 3D graph.)



The Axis dialog box displays.



2 Specify the attributes, as described next.

3 Click OK.

Specifying text attributes

You can specify the characteristics of text that displays for each axis. There are two kinds of text associated with an axis.

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 attributes of titles, legends, and axes" on page 658.

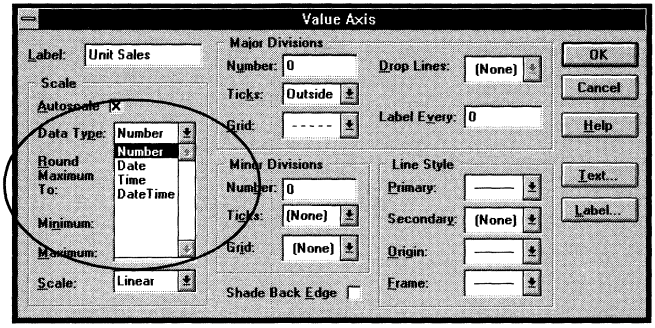
Specifying data types

The data graphed along the Value, Category, and Series axes has an assigned data type. The Series axis always has the data type String. The Value and Category axes can have the following data types:

Axis	Possible data types
Value	Number, Date, DateTime, Time
Category (for scatter graph)	Number, Date, DateTime, Time
Category (other graph types)	String, Number, Date, DateTime, Time

For graphs in DataWindows, PowerBuilder automatically assigns the data types based on the data type of the corresponding column; you do not specify them.

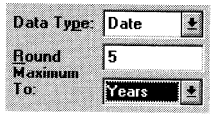
For graphs in windows, you specify the data types yourself. Be sure you specify the appropriate data types, so when you populate the graph (using the AddData function), the data matches the data type.



Scaling axes

You can specify several attributes that define the scaling used along numeric axes:

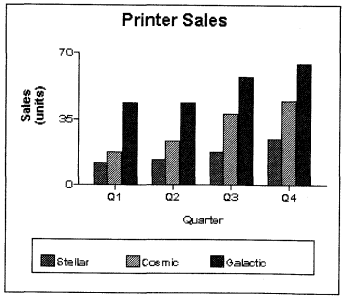
Attribute	Meaning
Autoscale	If selected (the default), PowerBuilder automatically assigns a scaling for the numbers along the axis.

Attribute	Meaning
Round Maximum To	Specifies how to round the end points of the axis. (Note that this just rounds the range displayed along the axis; it doesn't round the data itself.) You can specify a number and a unit. The unit is based on the data type; 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 will extend up to 1995, which is 1993 rounded to the next highest five-year interval. 
Minimum, maximum	The smallest and largest numbers to appear on the axis (disabled if you have selected Autoscale).
Scale	Specifies linear or logarithmic scaling (common or natural).

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.

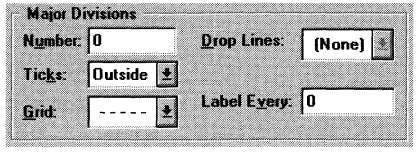
This graph's Value axis is divided into two major divisions. One goes from 0 to 35. The other goes from 35 to 70.



By default, PowerBuilder divides the axes automatically into major divisions.

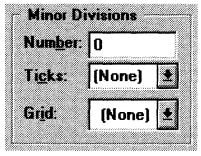
❖ To define divisions for an axis:

1 To divide an axis into a specific number of major divisions, type the number of divisions you want in the Number box in the Major Divisions group. To have PowerBuilder automatically create divisions, leave the number 0.



2 By default, PowerBuilder labels each tick mark in major divisions. If you don't want each tick mark labeled, enter a value in the Label Every box. For example, if you enter 2, PowerBuilder will label every second tick mark for the major divisions.

3 To use minor divisions, which are divisions within each major division, type the appropriate number in the Number box in the Minor Divisions group. Leave the number 0 to not use minor divisions.

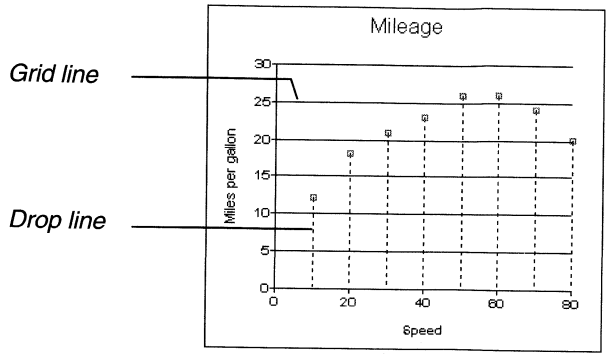


When using common logarithmic axes
If you want minor divisions, specify 1; otherwise, specify 0.

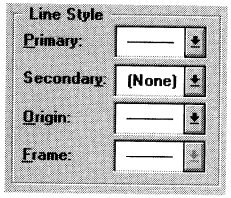
Specifying division lines

You can specify lines to represent the divisions:

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 from a data point to its axis (not available for all graph types).



Using line styles



You can define line styles for the following components of an axis:

Component	Meaning
Primary	The axis itself
Secondary	The axis parallel to and opposite the primary axis
Frame	The frame for the axis in 3D graphs (disabled for two-dimensional graphs)
Origin	A grid line that represents the value zero

Specifying a border

You can specify the border that PowerBuilder places around a graph.

❖ **To specify a border:**

1 Select Border from the graph's popup menu.

A cascading menu displays.

2 Select the type of border to use.

Specifying a pointer

You can specify a pointer to use when the mouse is over a graph during execution.

❖ To specify a pointer:

1 Select Pointer from the graph's popup menu.

The Select Pointer dialog box displays.

2 Select a stock pointer from the list, or select a CUR file containing a pointer.

3 Click OK.

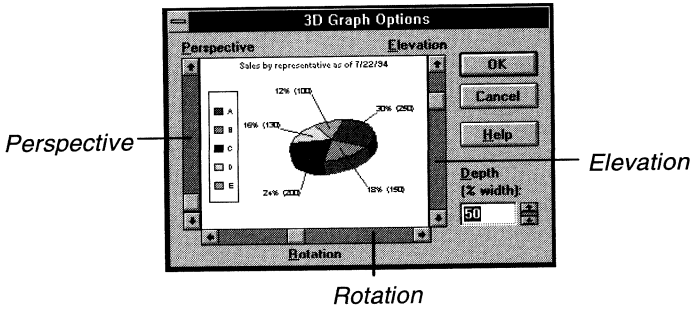
Specifying point of view in 3D graphs

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 Select 3D View from the graph's popup menu.
or
Click the 3D button in the Graph dialog box.

The 3D Graph Options dialog box displays.



2 Adjust the point of view along the three dimensions:

- To change the perspective, move the scroll box in the left scrollbar
- To rotate the graph, move the scroll box in the bottom scrollbar

- To change the elevation, move the scroll box in the right scrollbar

3 Define the depth of the graph (the percent the depth is of the width of the graph) by entering a number in the Depth box or using the scrollbars on the graph representation to increase or decrease the depth.

4 Click OK.

Using graphs in windows

In addition to using graphs in DataWindow objects, you can also place graphs directly in windows. You define attributes for a graph control in the Window painter and use scripts to populate the graph with data and to modify attributes for the graph during execution.

This section describes procedures unique to using graphs in windows.

For general graph properties, see "Defining a graph's attributes" on page 655.

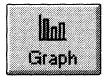
About user objects
You can also place graphs in user objects. Everything below about using graphs in windows also applies to using them in user objects.

Placing a graph in a window

❖ **To place a graph in a window:**

1. Open the Window painter and select the window that will contain the graph.

   The Window painter workspace displays showing the window.



2. Click the Graph button in the PainterBar.
3. Click where you want the graph.

   PowerBuilder displays a model of the graph in the window.

4. Specify attributes for the graph, such as type, title, text attributes, and axis attributes.

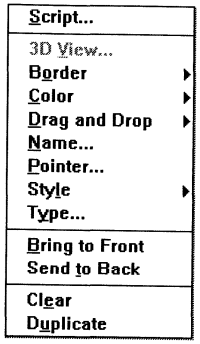
   See "Defining a graph's attributes" on page 655.

5. Write one or more scripts to populate the graph with data.

   See "Populating a graph with data" on page 674.

Using the popup menu

A graph's popup menu in the Window and User Object painters is similar to the popup menu in the DataWindow painter except there are items specific to use of a graph in a window or user object.



Menu item	Action
Script	Opens the PowerScript painter so you can write scripts for the graph's events
3D View	Defines the perspective used in 3D graphs (disabled if not working with 3D graph)
Border	Specifies the type of border around the graph
Color	Specifies the color for the graph's background and text
Drag and Drop	Specifies properties used for drag and drop
Name	Specifies the graph's name, title, type, and other properties
Pointer	Specifies the pointer to use when the mouse is positioned over the graph
Style	Specifies general properties for the graph
Type	Specifies the type of graph
Bring to Front	Places the graph in front of other controls
Send to Back	Places the graph behind other controls
Clear	Removes the graph
Duplicate	Copies the graph control

Basic control attributes

Graph controls in a window can be enabled or disabled, visible or invisible, and can be used in drag and drop.

They have the following events:

Clicked
Constructor
Destructor
DoubleClicked
DragDrop
DragEnter
DragLeave
DragWithin
GetFocus
LoseFocus
Other
RButtonDown

So, for example, you can write a script that is invoked when a user clicks a graph or drags an object on a graph (as long as the graph is enabled).

Populating a graph with data

You use the following PowerScript graph functions to manipulate data in a graph that is in a window.

> **For use in windows only**
> These functions are available only for graphs in windows. In DataWindow objects, graph data values are always tied directly to the data in the DataWindow itself and cannot be modified through a function.

Function	Action
AddCategory	Adds a category
AddData	Adds a data point
AddSeries	Adds a series
DeleteCategory	Deletes a category
DeleteData	Deletes a data point
DeleteSeries	Deletes a series
ImportClipboard	Copies data from the clipboard to a graph
ImportFile	Copies the data in a text file to a graph
ImportString	Copies the contents of a string to a graph
InsertCategory	Inserts a category before another category
InsertData	Inserts a data point before another data point in a series
InsertSeries	Inserts a series before another series
ModifyData	Changes the value of a data point
Reset	Reset the graph's data

Using AddSeries

You use AddSeries to create a series. AddSeries has this syntax:

graphName.**AddSeries** ("*seriesName*")

AddSeries returns an integer that identifies the series that was created. The first series is numbered 1, the second is 2, and so on. Typically you use this number as the first argument in other graph functions that manipulate the series.

So to create a series named Stellar, code:

```
int SNum
SNum = gr_1.AddSeries("Stellar")
```

Using AddData

You use AddData to add data points to a specified series. AddData has this syntax:

graphName.**AddData** (*seriesNumber,value,* "*categoryLabel* ")

The first argument to AddData is the number assigned by PowerBuilder to the series.

So to add two data points to the Stellar series, whose number is stored by the variable SNum (as shown above), code:

```
gr_1.AddData(SNum, 12, "Q1") // Category is Q1
gr_1.AddData(SNum, 14, "Q2") // Category is Q2
```

Getting a series number
You can use the FindSeries function to determine the number PowerBuilder has assigned to a series. FindSeries returns the series number. This is useful when you write general-purpose functions to manipulate graphs.

For more information

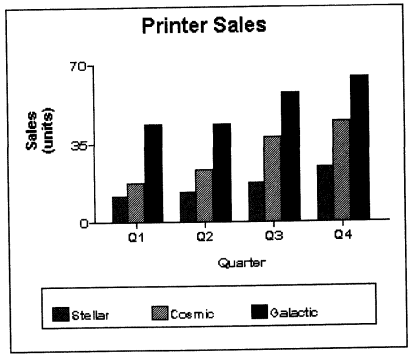
The *Function Reference* contains complete information about each graph function.

An example

Say you want to graph quarterly printer sales. Here is a script that populates the graph with data:

```
gr_1.Reset( All! )              // Resets the graph.

// Create first series and populate with data.

int SNum
SNum = gr_1.AddSeries("Stellar")
gr_1.AddData(SNum, 12, "Q1") // Category is Q1.
gr_1.AddData(SNum, 14, "Q2") // Category is Q2.
gr_1.Adddata(SNum, 18, "Q3") // Category is Q3.
gr_1.AddData(SNum, 25, "Q4") // Category is Q4.

// Create second series and populate with data.

SNum = gr_1.AddSeries("Cosmic")
gr_1.AddData(SNum, 18, "Q1") // Same categories
                             // as above
gr_1.AddData(SNum, 24, "Q2") // so the data
                             // appears next to
gr_1.Adddata(SNum, 38, "Q3") // the data in
gr_1.AddData(SNum, 45, "Q4") // the first series.

// Create third series and populate with data.

SNum = gr_1.AddSeries("Galactic")
gr_1.AddData(SNum, 44, "Q1")
gr_1.AddData(SNum, 44, "Q2")
gr_1.Adddata(SNum, 58, "Q3")
gr_1.AddData(SNum, 65, "Q4")
```

Here is the resulting graph:



You can add, modify, and delete data in a graph in a window through graph functions anytime during execution.

Accessing graphs at execution time

There are two kinds of graph attributes:

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

This section describes how you can access (and optionally modify) a graph by addressing these attributes in scripts at execution time.

Modifying graph attributes

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

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

With graphs in windows

For graphs in windows, you can modify attributes directly. For example, to change the type of the graph gr_emp to Column, you could code:

```
gr_emp.GraphType = ColGraph!
```

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

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

With graphs in DataWindow controls

For graphs in DataWindow controls, you have access to the same attributes and can use Modify to assign new values to them. For example, to change the title of graph gr_emp in DataWindow control dw_empinfo, you could code:

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

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

For more about Modify, see the *Function Reference*.

How parts of a graph are represented

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

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

Similarly, graphs have axes, each of which also has a set of attributes. These properties are stored as attributes in a subobject (structure) of Graph called **grAxis**. For example, graphs have an attribute Values of type grAxis, which contains attributes that specify the Value axis's properties, such as whether to use auto scaling of values, the number of major and minor divisions, the axis label, and so on.

Here is a representation of the attributes of a graph:

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

Referencing parts of a graph

You use dot notation to reference these display attributes.

For example, one of the attributes of a graph's title is whether the text is italicized or not. That information is stored in the boolean Italic attribute in the TitleDispAttr attribute of the graph.

With graphs in windows

In windows, you reference the attributes directly, so to italicize graph gr_emp's title, code:

```
gr_emp.TitleDispAttr.Italic = TRUE
```

Similarly, to turn on autoscaling of a graph's Values axis, code:

```
gr_emp.Values.Autoscale = TRUE
```

To change the label text for the Value axis, code:

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

To change the alignment of text in the Value axis's label text, code:

```
gr_emp.Values.LabelDispAttr.Alignment = Left!
```

With graphs in DataWindow controls

In DataWindow controls, you use Modify to change the attributes of a graph.

For example, to change the label text for the Value axis of graph gr_emp in the DataWindow control dw_empinfo, code:

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

For more information

For a complete list of graph attributes, see *Objects and Controls* or use the Object browser: select System for the Object Type and Attributes for the Paste Category.

For more about the Object browser, see Chapter 3, "Writing Scripts."

Accessing data attributes

To access attributes related to a graph's data during execution, you use the graph PowerScript functions.

The graph functions related to data fall into several categories:

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

How to use the functions

The syntax for these functions depends on whether you are referring to a graph in a DataWindow control or in a graph control in a window.

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

DataWindowName.FunctionName ("*graphName*", *otherArguments*)

For example, there is a function CategoryCount, which returns the number of categories in a graph. So to get the category count in the graph gr_printer, which is in the DataWindow control dw_sales, you would code:

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

In a graph control To call the functions for a graph in a graph control, use the following syntax:

graphControlName.FunctionName (Arguments)

For example, to get a count of the categories in the window graph gr_printer, code:

```
gr_printer.CategoryCount()
```

Getting information about the data

The following PowerScript functions allow you to get information about data in a graph in a DataWindow control or window at execution time:

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

Saving graph data

The following PowerScript functions allow you to save data from the graph:

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

Modifying colors, fill patterns, and other data attributes

The following PowerScript functions allow you to modify the appearance of data in a graph:

Function	Action
ResetDataColors	Resets the color for a specific data point
SetDataPieExplode	Explodes a slice in a pie graph
SetDataStyle	Sets the color, fill pattern, or other visual property for a specific data point
SetSeriesStyle	Sets the color, fill pattern, or other visual property for a series

Using these functions in a DataWindow control

You call the data-access functions after a graph has been created and populated with data. In graph controls in windows, you have full control over this, because these graphs are always populated through scripts.

But in DataWindow controls, some graphs (such as graphs that display data for a page or group of data) are destroyed and recreated internally as the user pages through the data. Any changes you made to the display of a graph (such as changing the color of a series) are lost when the graph is recreated.

To be assured that data-access functions are called whenever a graph has been created and populated with data, you can call the functions in a script for a user event that is triggered when a graph is created.

PowerBuilder provides an event ID, pbm_dwngraphcreate, that you can assign to a user event for a DataWindow control. The event is triggered by the DataWindow control after it has created a graph and populated it with data, but before it has displayed the graph. By accessing the data in the graph in this user event, you are assured that you are accessing the current data and that the data displays the way you want it.

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

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

2 Create a user event for the DataWindow control that is triggered whenever a graph in the control is created or changed:

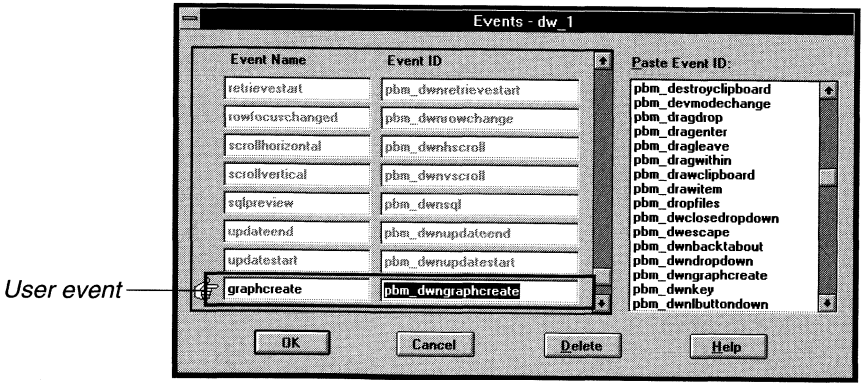
1 Select the DataWindow control.

2 Select Declare➤User Events from the menu bar.

The Events dialog box displays all the events defined for the DataWindow control.

3 Name the user event you are creating. You might want to call it GraphCreate.

4 Assign it the event ID pbm_dwngraphcreate by double-clicking pbm_dwngraphcreate in the Paste Event ID box.



5 Click OK to save the new user event.

3 Write a script for the new GraphCreate event that accesses the data in the graph. Use the data-access functions listed above.

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

Examples

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

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

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

```
int SeriesNum

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

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

For more information

For complete information about the data-access graph functions, see the *Function Reference.*

For more about user events, see Chapter 11, "Working with User Events."

Using point and click

Users can click graphs during execution. PowerScript provides a function that stores information about what was clicked in a graph. You can use this function in a number of ways in Clicked scripts. For example, you can provide your user with the ability to point and click on a data value in a graph and see information about the value in a message box. This section shows you how.

Clicked events and graphs

To cause actions when a user clicks a graph, you do the following:

- If the graph is in a DataWindow control, write a Clicked script for the DataWindow control
- If the graph is in a graph control, write a Clicked script for the graph control (make sure the graph is enabled)

Using ObjectAtPointer

The function that provides point-and-click functionality is ObjectAtPointer, which has the following syntax.

For graphs in a DataWindow control:

DataWindowName.**ObjectAtPointer** ("*graphName*", *seriesNumber, dataNumber*)

For graphs in a graph control:

graphName.**ObjectAtPointer** (*seriesNumber, dataNumber*)

You should call ObjectAtPointer in the first statement of a Clicked script.

When called, ObjectAtPointer does three things:

- It returns the kind of object clicked on as a grObjectType enumerated value. For example, if the user clicks on a data point, ObjectAtPointer returns TypeData!. If the user clicks on the graph's title, ObjectAtPointer returns TypeTitle!.

  For a complete list of the enumerated values of grObjectType, use the Object browser: select Enumerated as the object type and select grObjectType in the Objects window.

- It stores the number of the series the pointer was over in the variable *seriesNumber*, which is an argument passed by reference.
- It stores the number of the data point in the variable *dataNumber*, also an argument passed by reference.

After you have the series and data point numbers in a script, you can use other PowerScript functions to get or provide information. For example, you might want to report to the user the value of the clicked data point.

Example in a window

Assume there is a graph gr_sale in a window. The following script for gr_sale's Clicked event displays a message box:

- If the user clicks on a series (that is, if ObjectAtPointer returns TypeSeries!), the message box shows the name of the series clicked on. The script uses the function SeriesName to get the series name, given the series number stored by ObjectAtPointer.
- If the user clicks on a data point (that is, if ObjectAtPointer returns TypeData!), the message box lists the name of the series and the value clicked on. The script uses GetData to get the data's value, given the data's series and data point number:

```
int             SeriesNum, DataNum
double          Value
grObjectType    ObjectType
string          SeriesName, ValueAsString

// The following function stores the series number
// clicked on in SeriesNum and stores the number
// of the data point clicked on in DataNum.
ObjectType =  &
   gr_sale.ObjectAtPointer (SeriesNum, DataNum)

if ObjectType = TypeSeries! then
   SeriesName = gr_sale.SeriesName(SeriesNum)
   MessageBox("Graph",  &
      "You clicked on the series " + SeriesName)

elseif ObjectType = TypeData! then
   Value = gr_sale.GetData(SeriesNum, DataNum)
   ValueAsString = String(Value)
   MessageBox("Graph",  &
      gr_sale.SeriesName(SeriesNum) +  &
      " value is " + ValueAsString)
end if
```

Example in a DataWindow

Assume there is a graph gr_sale in the DataWindow control dw_printer. The following script for dw_printer's Clicked event provides the same functionality as the script above (the only difference between the two scripts is the syntax used for the graph functions):

```
int             SeriesNum, DataNum
double          Value
grObjectType    ObjectType
string          SeriesName, ValueAsString
string          GraphName

GraphName = "gr_sale"

// The following function stores the series number
// clicked on in SeriesNum and stores the number
// of the data point clicked on as DataNum.
ObjectType =  &
   dw_printer.ObjectAtPointer (GraphName,  &
   SeriesNum, DataNum)

if ObjectType = TypeSeries! then
   SeriesName =  &
      dw_printer.SeriesName(GraphName, SeriesNum)
   MessageBox("Graph",  &
      "You clicked on the series " + SeriesName)

elseif ObjectType = TypeData! then
   Value = dw_printer.GetData(GraphName, &
      SeriesNum, DataNum)
   ValueAsString = String(Value)
   MessageBox("Graph",  &
      dw_printer.SeriesName(GraphName,  &
      SeriesNum) + " value is " + ValueAsString)
end if
```

CHAPTER 19

Working with Crosstabs

About this chapter

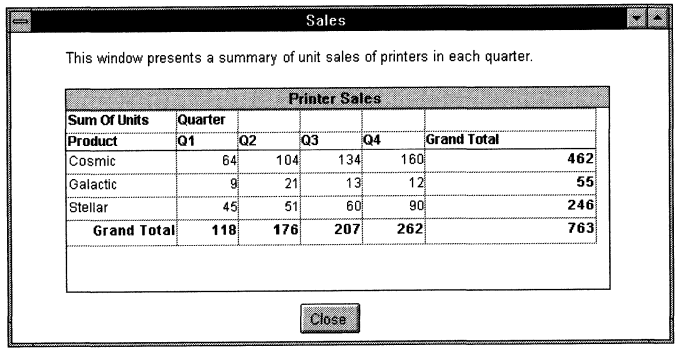
This chapter describes how to build crosstabs.

Contents

Overview

Users often want to analyze data. One technique is to graph the data, as described in the preceding chapter. Another useful technique is crosstabulation, which analyzes data in a spreadsheet-like grid. Instead of seeing a long series of rows and columns, users can look at a crosstab that summarizes the data.

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 DataWindow presentation style. When data is retrieved into the DataWindow, the crosstab processes all the data and presents the summary information that you have defined for it.

An example

Crosstabs are easiest to understand through an example. Consider the Printer table in the Powersoft Demo Database. It records quarterly unit sales of printers made by sales representatives in one year (this is the same data that was used to illustrate graphs in the preceding chapter):

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
.	.	.	.
.	.	.	.

Rep	Quarter	Product	Units
.			
.			
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:

First-quarter sales of Cosmic printers

DataWindow - d_crosstab

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, all you have to do is 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, it 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, it 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 can 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 during execution (if you let them).

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 execute the crosstab. The number of columns and rows in the crosstab match the data that exists when the crosstab is *executed.*

Using the preceding crosstab as an example, if a new printer is added to the database after the crosstab was saved, there would be an additional row in the crosstab when it is later executed. Similarly, if one of the quarter's results had been deleted from the database after the crosstab was saved, there would be one fewer column in the executed crosstab.

By default, crosstabs you build are dynamic.

Static crosstabs

Static crosstabs are quite different.

With static crosstabs, PowerBuilder establishes the columns and rows 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 in the DataWindow painter workspace). No matter what values are in the database when you later execute the crosstab, the crosstab will always have the same columns and rows as when you defined it.

Using the preceding crosstab as an example, if there were four quarters and three printers in the database when you defined and saved the crosstab, there would always be four columns (Q1, Q2, Q3, and Q4) and three rows (Cosmic, Galactic, and Stellar) in the executed crosstab, even if the number of printers or columns changed in the database.

Advantages of dynamic crosstabs

In most cases, you will want to use dynamic crosstabs, for the following reasons:

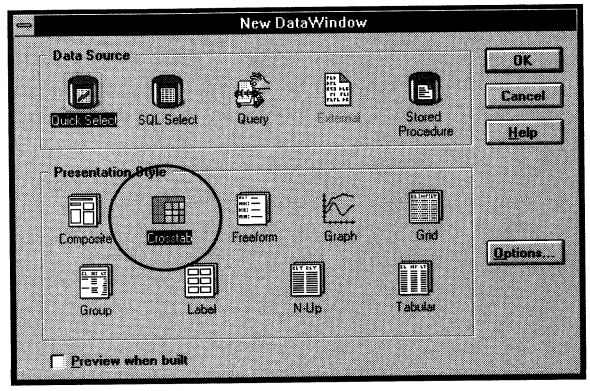
- You can define dynamic crosstabs very quickly because no database access is required at definition time.
- Dynamic crosstabs always use the current data to build the columns and rows in the crosstab. Static crosstabs show a snapshot of columns and rows as they were when the crosstab was defined.
- Dynamic crosstabs are easy to modify: all attributes for the dynamically built columns are replicated during execution automatically. With static crosstabs, you must work with one column at a time.

Creating crosstabs

❖ To create a crosstab:

1 Open the DataWindow painter and select New in the Select DataWindow dialog box.

The New DataWindow dialog box displays.



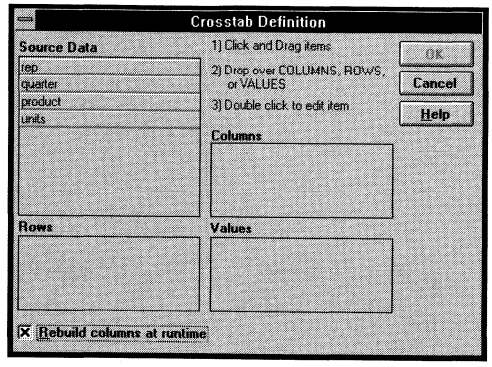
2 Select a data source and the Crosstab presentation style, then click OK.

You are prompted to specify the data.

3 Specify the data you want retrieved into the DataWindow object.

For more information, see Chapter 13, "Defining DataWindow Objects."

You are prompted to specify the data for the columns, rows, and cells in the crosstab.



4 Enter the definitions for the columns, rows, and cell values in the crosstab.

See "Associating data with a crosstab" on page 696.

5 Click OK.

PowerBuilder places the crosstab in the workspace.

6 (Optional) Preview the crosstab to see how it looks.

See "Previewing crosstabs" on page 704.

7 (Optional) Specify other attributes of the crosstab.

See "Enhancing crosstabs" on page 705.

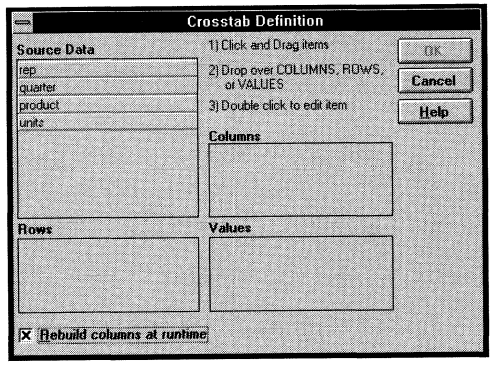
8 Save the DataWindow object in a library.

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 Crosstab Definition 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 714.

4 Click OK.

The process is illustrated using the following dynamic crosstab. The columns in the database are Rep, Quarter, Product, and Units, as shown on page 690. The crosstab shows sales of printers (the Product) by quarter.

DataWindow - d_crosstab

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 information

To define the crosstab, you simply drag the column names from the Source Data box in the Crosstab Definition dialog box into the Columns, Rows, or Values box, as appropriate.

If you change your mind, drag the column name out of the Columns, Row, or Values box and drop it. Then specify a different column.

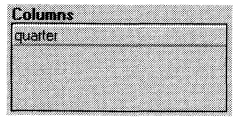
Specifying the columns

You use the Columns box to specify one or more of the retrieved columns to provide the columns in the crosstab. During execution, there will be 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 preceding 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 will be four columns in the crosstab.



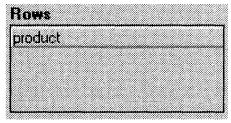
Specifying the rows

You use the Rows box to specify one or more of the retrieved columns to provide the rows in the crosstab. During execution, there will be 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 preceding 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), during execution there will be three rows in the crosstab.



Display values are used
If you specify columns in the database that use code tables—data is stored with a data value but displayed to the user with more meaningful display values—the crosstab uses the column's display values, not the data values.

For more about code tables, see "Defining a code table" in Chapter 15, "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. During execution, each cell has a calculated value based on the function you provide here and the column and row values for the particular cell.

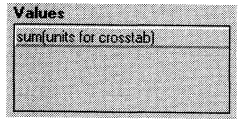
❖ To specify the crosstab's values:

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 preceding 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 PowerScript expression to define the columns, rows, and values used in the crosstab. You can use any non–object-level PowerScript 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:

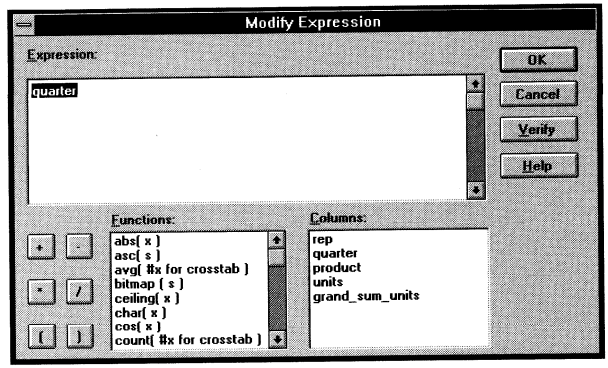
```
Month(SaleDate)
```

The Month PowerScript 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, double-click the item in the Columns, Rows, or Values box.

The Modify Expression dialog box displays.

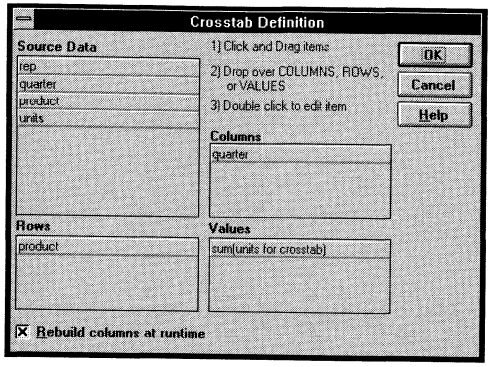


2 Specify the expression and click OK.

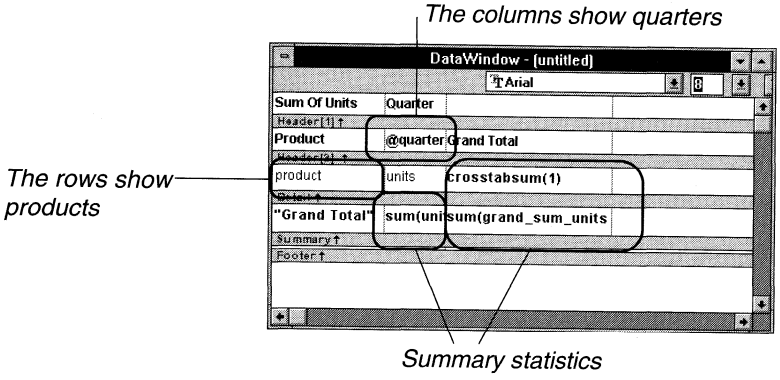
What happens

After you have specified the data for the crosstab's columns, rows, and columns, PowerBuilder displays the crosstab definition in the DataWindow painter workspace. The crosstab is implemented as a grid DataWindow.

To create the dynamic crosstab shown earlier, these values:



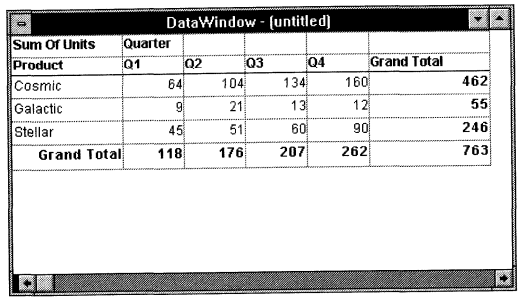
result in this workspace:



Notice that in the workspace 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) during execution.

The crosstab is generated with summary statistics: the rows and columns are totaled for you.

At this point, you can execute the crosstab by clicking the Preview button. Here is the crosstab during execution:



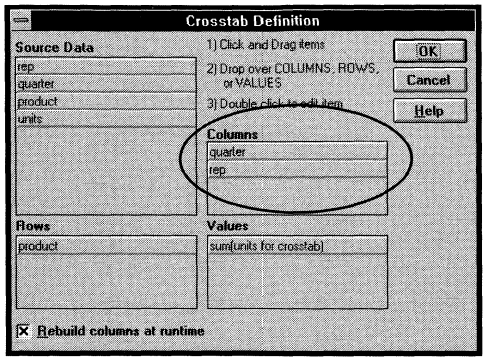
Sum Of Units	Quarter				
Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	60	90	246
Grand Total	118	176	207	262	763

- 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

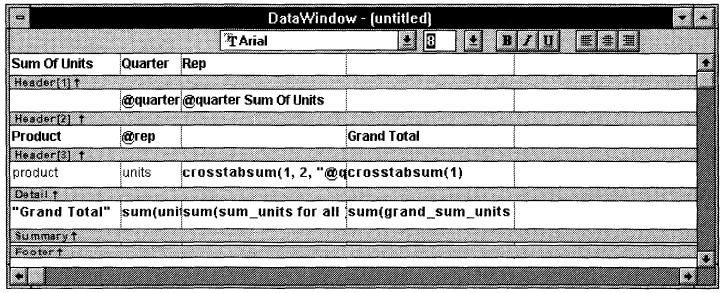
Typically you will specify one database column as the Columns definition and one database column for the Rows definition, as in the preceding crosstab. But you can specify as many columns, or expressions, as you want.

For example, consider the following, which specifies two database columns as the Columns definition:

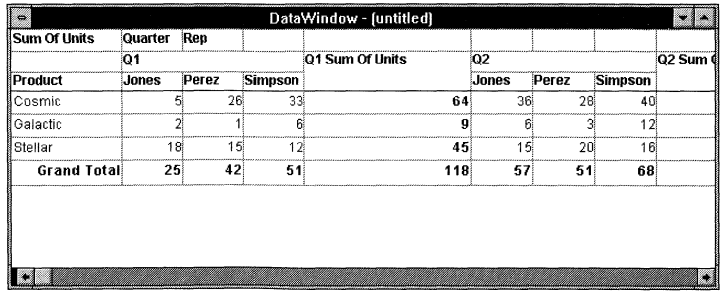


Here we want columns in the crosstab for quarters and for sales reps.

PowerBuilder displays this in the workspace:



This is what you get during execution:



For each quarter, the crosstab shows sales of each printer by each sales representative.

Previewing crosstabs

Once you have defined the data for the crosstab, you can execute it to see the data. You execute it by previewing the DataWindow object.

❖ To preview the crosstab:

1 Click the Preview button.
or
Select Design➤Preview from the menu bar.
or
Press CTRL+W.

You are now in preview. The bars that indicate the DataWindow bands disappear. PowerBuilder retrieves the rows and performs the crosstabulation on the data.

2 Continue previewing your DataWindow. You can do the same things you do when previewing standard grid DataWindow objects. For example, you can resize columns, filter rows, sort rows, save data in an external file, and print the results.

For more on what you can do in preview, see Chapter 14, "Enhancing DataWindow Objects."

3 When you have finished previewing the crosstab, click the Design button.

You return to the workspace. Changes you made (such as resizing columns) are retained.

Enhancing crosstabs

Once you have provided the data definitions, the crosstab is functional. But you will probably want to 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 might want to:

- Sort or filters rows
- Change the column headers
- Specify colors, fonts, 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 14, "Enhancing DataWindow Objects."

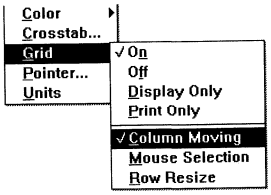
The rest of this section covers topics either unique to crosstabs or especially important when working in crosstabs:

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 during execution

❖ To specify the crosstab's basic properties:



1 Position the mouse below the footer band in the workspace and display the popup menu.

2 Select Grid from the popup menu.

3 Select the option you want.

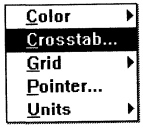
Option	Result
On	Grid lines always display
Off	Grid lines never display (users cannot resize columns during execution)
Display Only	Grid lines display only when the crosstab displays online
Print Only	Grid lines display only when the contents of the crosstab are printed
Column Moving	Users can reorder columns during execution
Mouse Selection	Users can select data during execution (and, for example, copy it to the clipboard)
Row Resize	Users can resize individual rows during execution

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.

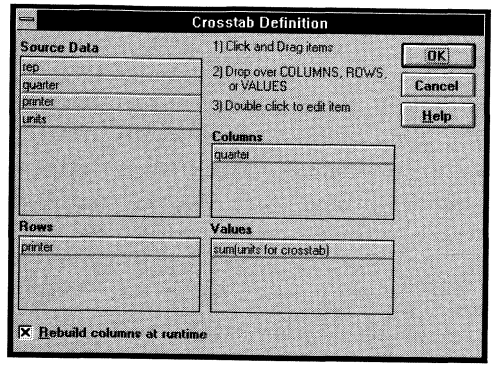
❖ To modify the data associated with a crosstab:



1 Position the mouse below the footer band in the workspace and display the popup menu.

2 Select Crosstab from the popup menu.

The Crosstab Definition dialog box displays.



3 Fill in the boxes for Columns, Rows, and Values as described in "Associating data with a crosstab" on page 696.

4 Click OK.

PowerBuilder changes the definition of the crosstab.

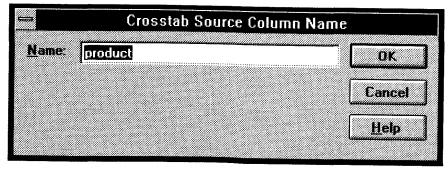
Changing the names used for the columns and rows

Sometimes names of columns in the database are not very user-friendly and might not be meaningful to your users. You can change the names that are used to label rows and columns in crosstabs so users better understand the data.

❖ To change the names used in crosstabs:

1 Double-click the name of the column in the Source Data box in the Crosstab definition dialog box.

The Crosstab Source Column Name dialog box displays.



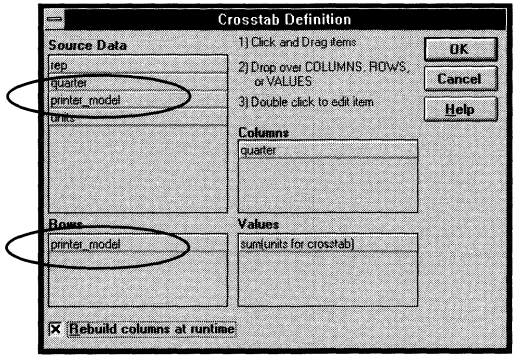
2 Specify the name you want to be used to label the corresponding column. You can have multiple-word labels by using underscores: underscores are replaced by spaces in the workspace and during execution.

3 Use the newly named column in the Columns, Rows, or Values box.

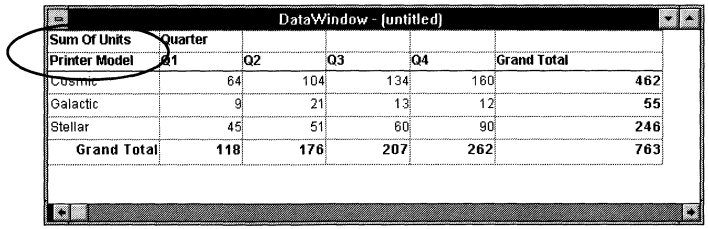
> **Be sure you use the new names everywhere**
> If you have renamed the columns in the Source Data box, be sure to also use the new names when you use the columns elsewhere in the Crosstab Definition dialog box. If you don't, you will get an error.

Example

For example, if you want the Product column to be labeled *Printer Model*, specify **Printer_Model** in the Crosstab Source Column Name dialog box.



During execution, you will 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.

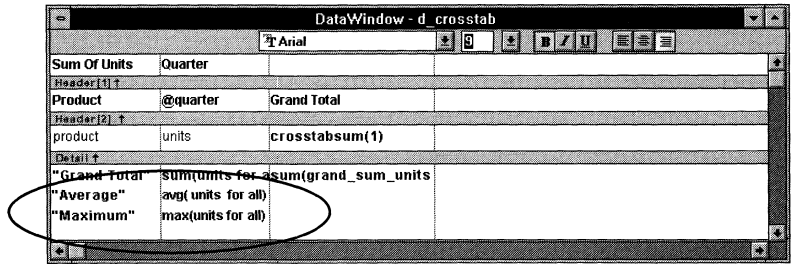
2 Click the Compute button in the PainterBar.
or
Select Objects➤Computed Field from the menu bar.

3 Click the cell in the summary band where you want the summary to display.

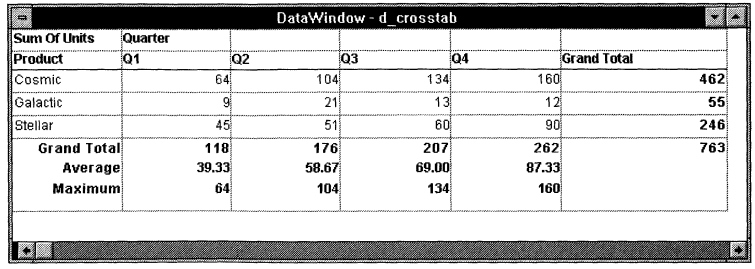
The Computed Field Definition dialog box displays.

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.

Here is a crosstab that has been enhanced to show averages and maximum values for each column. This is the workspace:



And this is the crosstab during execution:



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
Average	39.33	58.67	69.00	87.33	
Maximum	64	104	134	160	

❖ To define a row summary:

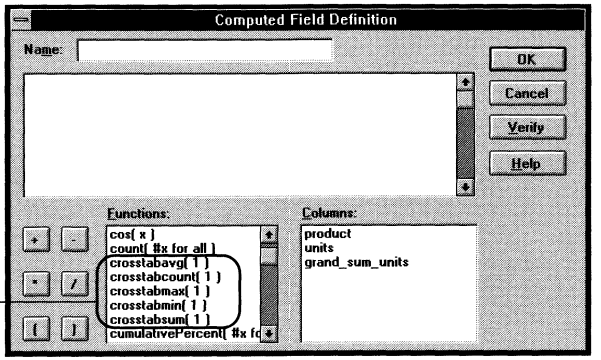
1 Click the Compute button in the PainterBar.
or
Select Objects➤Computed Field from the menu bar.

2 Click the empty cell to the right of the last column in the detail band.

3 Define the computed field. You should use one of the crosstab functions, described next.

Using crosstab functions

There are five special functions you can use only in crosstabs: CrosstabAvg, CrosstabCount, CrosstabMax, CrosstabMin, and CrosstabSum. These functions are listed in the Functions box when you define a computed field in a crosstab.



Crosstab functions

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 workspace.

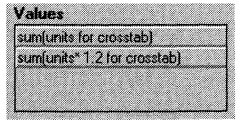
By default PowerBuilder places CrosstabSum in the detail band, which returns the total for the corresponding row.

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.

But assume you want to crosstabulate total unit sales and also a projection for future sales, assuming a 20 percent increase in sales (that is, sales that are 1.2 times the actual sales). You would define two expressions for Values:



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 14, "Enhancing DataWindow Objects."

For more about the crosstab functions, see the *Function Reference.*

Crosstabulating ranges of values

Often you want to build a crosstab where each row tabulates a *range* of values, instead of one discrete value. Similarly, you might want each column in the crosstab to correspond to a range of values.

For example, in crosstabulating 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 crosstabulate 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, 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 make between $30,000 and $40,000, how many make between $40,000 and $50,000, how many make 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
```

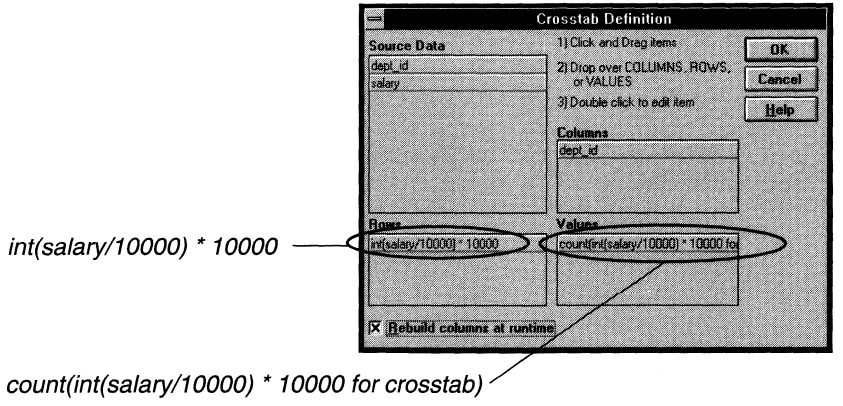
That expression divides the salary by 10,000 and takes the integer portion, then multiplies the result by 10,000. So for $34,000, the expression returns $30,000, as follows:

```
34000/10000 = 3.4
int(3.4)    = 3
3 * 10000   = 30000
```

With that information you can build the crosstab. The following uses the Employee table in the Powersoft Demo Database.

1 Build a crosstab and retrieve the Dept_ID and Salary columns.

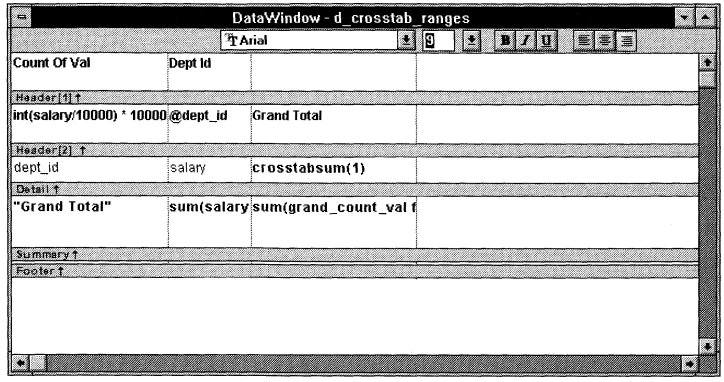
2 In the Crosstab Definition dialog box, drag the Salary column from the Source Data box to the Rows box and to the Values box, and edit the expressions as shown below.



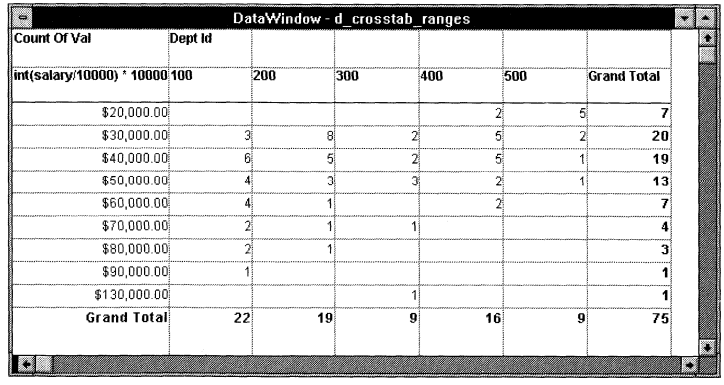
For more on providing expressions in a crosstab, see "Using expressions" on page 699.

3 Click OK.

This is the resulting workspace:



And here is the crosstab during execution:



Count Of Val	Dept Id					
int(salary/10000) * 10000	100	200	300	400	500	Grand Total
$20,000.00				2	5	7
$30,000.00	3	8	2	5	2	20
$40,000.00	6	5	2	5	1	19
$50,000.00	4	3	3	2	1	13
$60,000.00	4	1		2		7
$70,000.00	2	1	1			4
$80,000.00	2	1				3
$90,000.00	1					1
$130,000.00			1			1
Grand Total	22	19	9	16	9	75

You can see, for example, that two people in department 400 and five in department 500 make between $20,000 and $30,000.

Creating static crosstabs

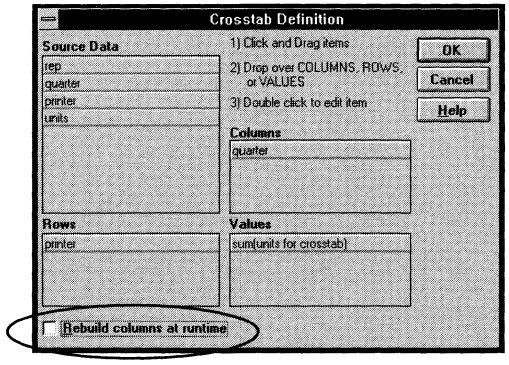
By default, crosstabs are dynamic: when they are executed, PowerBuilder retrieves the data and dynamically builds the columns and rows based on the retrieved data. This is usually what you want. For example, if you have defined a crosstab that computes sales of printers and a new printer type is entered in the database after you defined the crosstab, you want the new printer to be in crosstabs executed subsequently. That is, you want PowerBuilder to dynamically build the rows and columns 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 and rows to be established when you define the crosstab. You do not want additional columns or rows to display in the crosstab during execution; no matter what the data looks like when the crosstab is executed, you don't want the number of columns or rows to change. You only want the updated statistics for the predefined columns and rows. Here is how to do that.

❖ To create a static crosstab:

1 Display the Crosstab Definition dialog box.

2 Deselect the Rebuild Columns At Runtime checkbox.



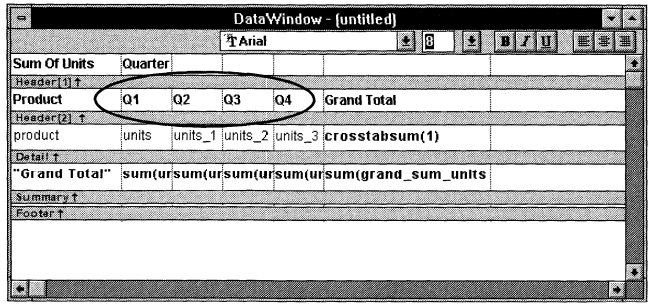
3 Define the data for the crosstab as usual.

4 Click OK.

What happens

With the checkbox deselected, 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 screen, the four values for Quarter (Q1, Q2, Q3, and Q4) are displayed in the workspace.



When the crosstab is executed, no matter what values are in the database for the column, the crosstab will show only the values that were specified when the crosstab was defined. In the preceding example, the crosstab will always have four columns and the same number of rows it had when it was first defined and previewed in the DataWindow painter.

Making changes

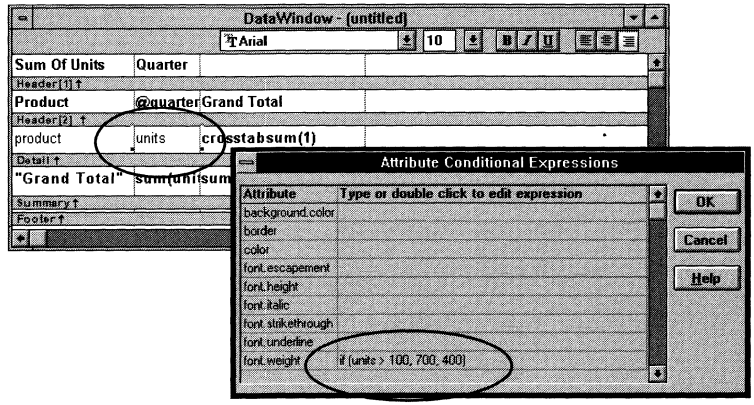
You can modify the attributes of any of the objects in a static crosstab. You can modify the attributes of each column individually, since each column is displayed in the workspace as part of the crosstab's definition. For example, in the preceding crosstab you can directly modify the way values are presented in each individual quarter, since each quarter is represented in the workspace (the values are shown as units, units_1, units_2, and units_3).

Using attribute conditional expressions

As with other DataWindow objects, you can specify attribute conditional expressions to modify attributes at execution time. 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 attributes when it builds the individual columns during execution. 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.

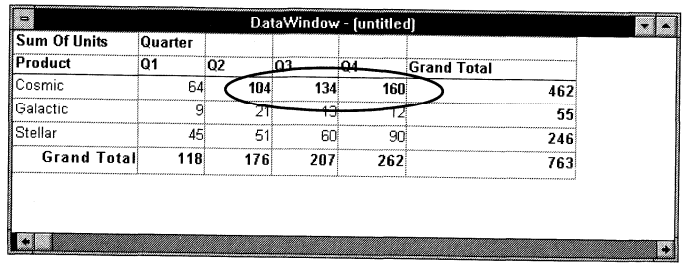


Here is the expression, which is for the Font.Weight attribute:

```
if (units > 100, 700, 400)
```

The expression specifies to use bold font (weight = 700) if Units is greater than 100. Otherwise, use normal font (weight = 400).

Here is the crosstab during execution:



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 large values are shown in bold.

For more about attribute conditional expressions, see Chapter 14, "Enhancing DataWindow Objects."

Using crosstabs in an application

When you have completed the crosstab's definition, you associate it with a DataWindow control in a window, resize the control as needed, then write scripts to manipulate the control.

For general information about using DataWindow objects (including crosstabs) in applications, see *Building Applications*.

Here are some tips specifically about using crosstabs in an application.

Viewing the underlying data

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

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

> **Don't share data between the two DataWindows or reports**
> They have the same SQL SELECT data definition, but they have different result sets.

For more about composite reports, see Chapter 17, "Using Nested Reports."

Letting the user redefine the crosstab

You can allow your user to redefine which columns in the retrieved data are associated with the crosstab's columns, rows, and values during execution by using the CrosstabDialog function.

If you call CrosstabDialog in a script, the Crosstab Definition dialog box is displayed to your user, who can then define the data used for the crosstab's columns, rows, and values using exactly the same techniques you use in PowerBuilder. When the user clicks OK in the dialog box, PowerBuilder rebuilds the crosstab with the new specifications.

Displaying information messages

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

❖ To display informational messages when a crosstab is rebuilt:

1 Define a user event for the DataWindow control containing the crosstab. Associate it with the event ID pbm_dwnmessagetext.

2 In the script for the user event, call the GetMessageText function, which returns the message that PowerBuilder would be displaying when building the crosstab in the DataWindow painter.

3 Use the return values from GetMessageText to display the messages in your window.

Examples

Example 1 The following line in a script for the DataWindow control's user event for pbm_dwnmessagetext displays informational messages in a static text control in the window containing the crosstab:

```
st_message.Text = This.GetMessageText( )
```

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

Example 2 The following line in the user event for pbm_dwnmessagetext displays informational messages as MicroHelp in an MDI application (w_crosstab is an MDI frame window):

```
w_crosstab.SetMicroHelp(This.GetMessageText( ))
```

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

For more information

For more about user events, see Chapter 11, "Working with User Events."

For more about CrosstabDialog and GetMessageText, see the *Function Reference*.

Modifying the crosstab's attributes during execution

As with other DataWindow objects, you can modify the attributes of a crosstab during execution using the Modify function. Some changes require PowerBuilder to dynamically rebuild the crosstab; others don't. (If the original crosstab was static, it becomes a dynamic crosstab when it is rebuilt.)

You can change the following attributes without forcing PowerBuilder to rebuild the crosstab:

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

If you change any other attributes, PowerBuilder will rebuild the structure of the crosstab when Modify is called. You should combine all needed expressions into one Modify call so PowerBuilder has to rebuild the crosstab only once.

For computations that are derived from existing columns, PowerBuilder will by default use the attributes from the existing columns. For completely new columns, attributes such as font, color, and so on will default to the first column of the preexisting crosstab. Attributes for text in headers will default to the attributes of the first text object in the preexisting crosstab's first header line.

CHAPTER 20

Using the Data Pipeline Painter

About this chapter

You use the Data Pipeline painter to pipe data within a database or from one database to another. This chapter describes the Data Pipeline painter and how to use it.

Contents

About data pipelines

The Data Pipeline painter gives you the ability to easily copy data from one database to another. With the Data Pipeline painter, you can perform some tasks that would otherwise be very time consuming. For example, you can:

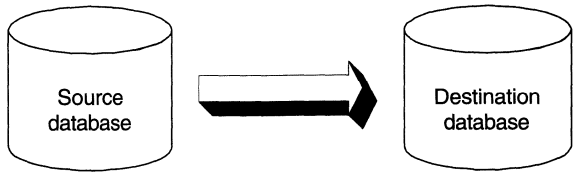
- Pipe data (and extended attributes) from one or more tables to a table in the same DBMS or a different DBMS
- Pipe an entire database, a table at a time, to another DBMS (and if needed, pipe the database's repository tables)
- Create a table with the same design as an existing table but with no data
- Pipe corporate data from a database server to a Watcom SQL 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 Alter Table dialog box

Piping data during execution
This chapter describes how to use the Data Pipeline painter to define pipeline objects and pipe data in the PowerBuilder development environment. You can also use pipeline objects to pipe data in applications you build.

For more about piping data in an application, see *Building Applications*.

Source and destination databases

You can use the Data Pipeline painter to pipe data in *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 a view.

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.

The source database and destination database are usually different databases, and they can even have different DBMSs. For example, you can pipe data in a SQL Server database to a Watcom SQL database on your computer.

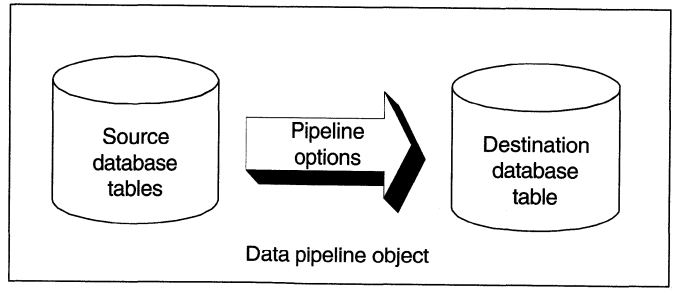
The source and destination can also be the same database.

Defining a data pipeline

When you use the Data Pipeline painter to create a pipeline, you define:

- Source database
- Destination database
- Source tables to access and the data to retrieve from them
- Pipeline operation
- Destination table

When you define these items, you create a definition for the piping of data. This definition is called a data pipeline or **pipeline**. After you create a pipeline object, you can execute it immediately and close the pipeline without saving it, or you can save a pipeline as a named object to reuse in the development environment and to use in an application you build.



Saving a pipeline enables you to pipe the data that may have changed since the last pipeline execution or to pipe the data to other databases later.

Data types

Each DBMS has certain data types it supports. When you pipe data from one DBMS to a different DBMS, PowerBuilder makes its best guess for the destination data types. You can correct PowerBuilder's best guess as needed.

Supported data types
The Data Pipeline painter does not support blob data types. All other data types are supported.

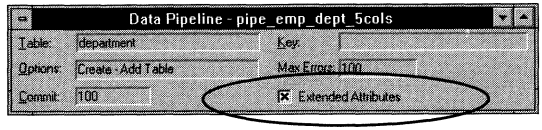
Piping extended attributes

The first time PowerBuilder connects to a database, it creates five Powersoft system tables, called the Powersoft repository. 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 about the repository, see Chapter 12, "Managing the Database," and Chapter 15, "Displaying and Validating Data."

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 checkbox in the Data Pipeline painter workspace.



When the Extended Attributes checkbox is selected, the extended attributes associated with the source database's selected columns automatically go into the repository of the destination database, with the following exception.

When extended attributes are not automatically piped

When you pipe data that has an edit style, display format, or validation rule associated with it, the style, rule, or format will *not* be piped if one with the same name exists in the repository of the destination database. In this situation, the data 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 Powersoft repository

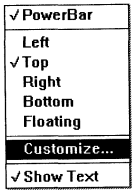
Selecting the Extended Attributes checkbox *never* results in the piping of named display formats, edit styles, and validation rules stored in the repository and independent of data. If you want extended attribute definitions from one database to exist in another database, you can pipe the appropriate Powersoft repository table or a selected row or rows from the table.

If you want to reproduce an entire database, you can pipe all database tables and repository tables, one table at a time.

Accessing the Data Pipeline painter

To access the Data Pipeline painter, you customize the PowerBar to add the Data Pipeline button, or you use the PowerPanel.

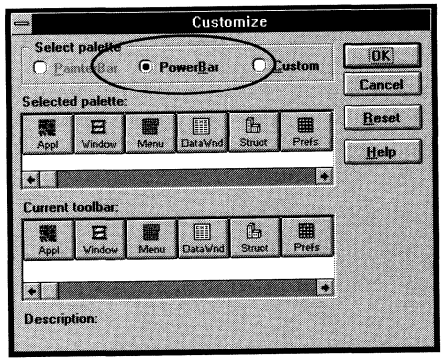
❖ To add the Data Pipeline painter to the PowerBar:



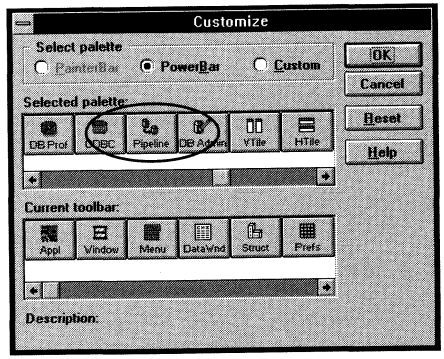
1 Move the pointer to the PowerBar and display the popup menu.

2 Select Customize from the popup menu.

The Customize dialog box displays with the PowerBar palette selected.

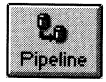


3 Scroll to display the Data Pipeline button in the Selected Palette box.



4 Move the pointer to the Data Pipeline button, drag the button to the position you want in the Current Toolbar box, and drop the button.

5 Click OK.

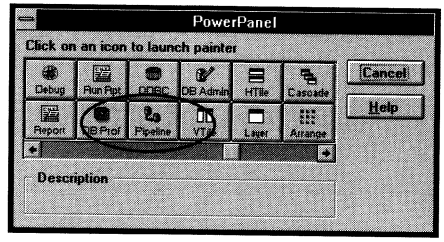


The Data Pipeline button displays in the PowerBar.

❖ To access the Data Pipeline painter in the PowerPanel:

1 Press CTRL+P to display the PowerPanel.

2 Scroll to the Data Pipeline button and click it.



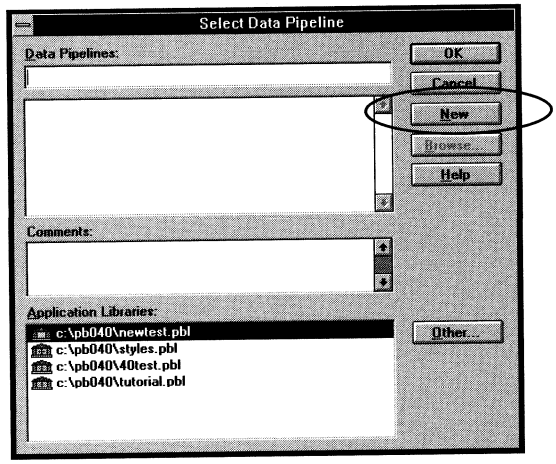
Creating a data pipeline

You can create a pipeline and execute it immediately. You can also save the pipeline to execute later, either in the development environment or in an application.

❖ To create a pipeline:

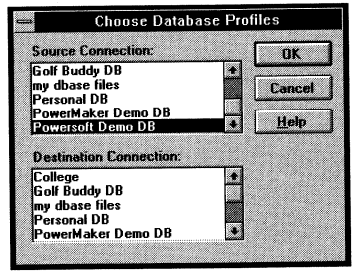
1 Click the Pipeline button in the PowerBar (if you have customized the PowerBar) or in the PowerPanel.

The Select Data Pipeline dialog box displays. Pipelines in the current library are listed.



2 Click New.

The Choose Database Profiles dialog box displays. The database you are currently connected to is selected in the Source Connection box.



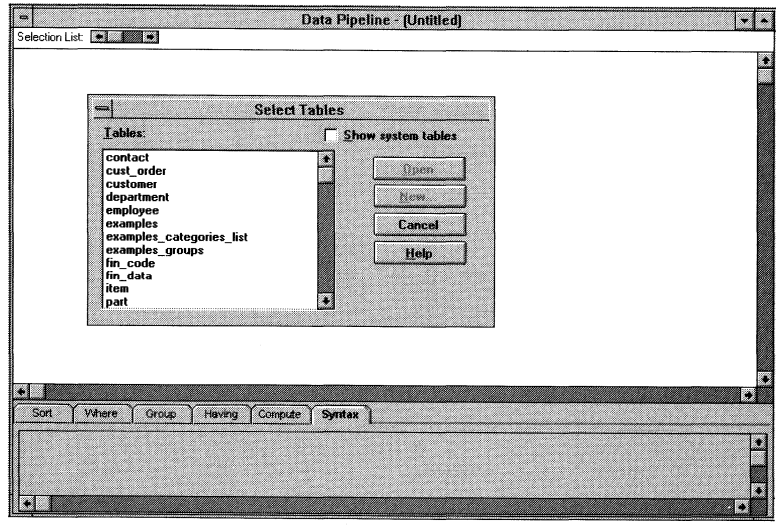
3 Select the source database profile and the destination database profile that you want.

> **Database profiles for the source and destination databases**
> The source and destination databases must each have a database profile defined.
>
> For information about defining a database profile, see *Connecting to Your Database.*

4 Click OK.

PowerBuilder connects to the source database and the destination database.

The Select Tables dialog box for the source database displays in the Select painter workspace.



5 Select one or more tables that contain the data you want to pipe and open the tables.

6 Define the data to pipe.

Where

To pipe a subset of data in the source tables, enter an appropriate expression in the Where tab to limit the data.

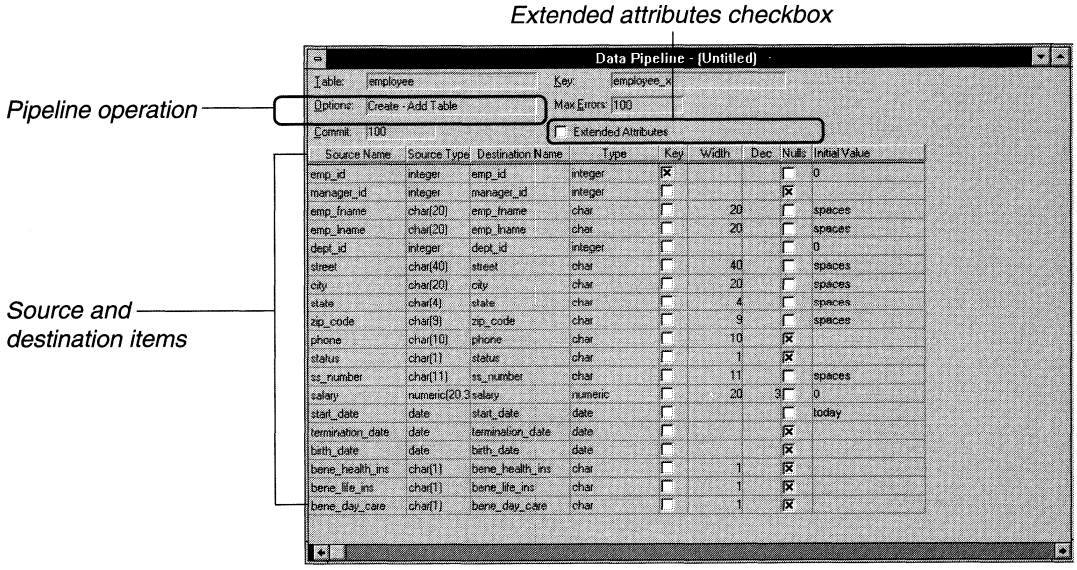
Using retrieval arguments

In the Select painter, you can specify retrieval arguments. When you specify retrieval arguments for a pipeline, you are prompted to supply a value before the Data Pipeline painter pipes data. The data piped depends on the value you supply. For example, if you specify a state as a retrieval argument and supply a state, such as NY or MA, then data for that state is piped to the destination database.

For information about using the Select painter to define data and retrieval arguments, see Chapter 13, "Defining DataWindow Objects."

7 Click the Design button.

The Data Pipeline painter workspace displays the pipeline definition, which includes a pipeline operation, a checkbox 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 in the Select painter.

Saving the pipeline

If you want to reuse the pipeline, you should save it now.

For information, see "Saving a pipeline" on page 748.

8 Select a pipeline operation (Create, Replace, Refresh, Append, or Update) from the Options dropdown listbox.

For information, see "Choosing a pipeline operation" on page 735.

9 Modify the pipeline definition as needed.

Modifications can include:

- Choosing to pipe extended attributes
- Specifying the Commit value to tell PowerBuilder the number of rows to pipe before committing the rows to the destination table
- Specifying the Max Errors value to tell PowerBuilder the error limit when execution stops
- Changing the destination database or reconnecting to the source database

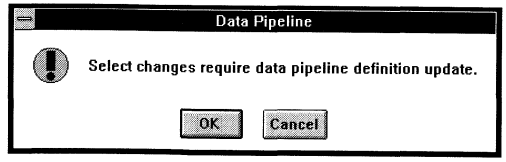
How you modify the definition depends on what you are trying to accomplish and the pipeline operation you choose.

For information about modifying the items in the workspace, see "Modifying a data pipeline" on page 739. For information about changing the destination and source databases, see "Changing the destination and source databases" on page 736.

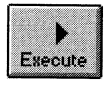
10 (Optional) Modify the source data as needed.

- Click the SQL Select button.
  or
  Select Options➤Edit Data Source from the menu bar.

  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.



11 Click the Execute button.
or
Select Options➤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 retrieval arguments.

During execution, 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 may stop execution when errors occur.

For information about execution and how rows are committed to the destination table, see "When execution stops" on page 737.

12 If you want to save the pipeline definition and have not already done so, save the pipeline definition.

For information, see "Saving a pipeline" on page 748.

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.

Choosing a pipeline operation

When PowerBuilder pipes data, what happens in the destination database depends on which pipeline operation you choose.

❖ To choose a pipeline operation:

- Select a pipeline operation from the Options dropdown listbox.

The five pipeline operations are described below.

When you choose this pipeline operation	This happens in the 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 don't match the key criteria values are inserted.

Piping to Watcom SQL

If you are building an application that pipes data to a Watcom SQL database and choose Create or Refresh as the pipeline operation, be sure that your end users have either the server edition of Watcom SQL or, if using a local version, that they have the full Watcom SQL development engine—not the runtime-only engine (the runtime engine doesn't support data definition statements such as CREATE TABLE).

If your users have the runtime-only engine, a workaround is to prime their database with the destination table and use Refresh as the pipeline operation.

Changing the destination and source databases

When you create a pipeline, you can change the destination database. And anytime you want to pipe the same data to more than one destination, you can execute a pipeline many times and change the destination database each time.

❖ To change the destination database:

- Click the Pipeline Profile button in the PainterBar.
  or
  Select File➤Destination Connect from the menu bar.

Normally you don't change the source database, because your pipeline definition is dependent on the source. If for some reason you are no longer connected to your source database, you can reconnect to your source database.

❖ To reconnect to your source database:

- Select File➤Source Connect from the menu bar.

When execution stops

Execution of a pipeline stops for one of two reasons:

- You stop the execution.
- PowerBuilder stops execution when the error limit is reached.

When a pipeline is executing, the Execute button in the PainterBar changes to a Cancel button. While the Cancel button is displayed, some rows may not pipe because of errors, but the Max Errors limit is not reached.

❖ To stop execution:

- Click the Cancel button.

If there are rows that can't be piped to the destination table for some reason and the error limit is reached, PowerBuilder stops execution and displays the error rows. You can correct errors there or return to the workspace to change the pipeline definition and then execute the pipeline again.

For information, see "Correcting pipeline errors" on page 746.

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:

- The Commit and Max Errors values
- When errors occur during execution
- Whether you click the Cancel button or PowerBuilder stops execution

If the Commit value is	Then when you click Cancel
All	Every row that was piped is rolled back.
A number *n*	Every row that was piped is committed.

For example, if you click the Cancel button or a time limit is reached when the 24th row is piped and if the Commit value is 20, then:

20 rows are piped and committed
3 rows are piped and committed
Piping stops

If the Commit value is	And the Max Errors value is	Then when PowerBuilder stops execution because the error limit is reached
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	*No Limit*	All rows that pipe without error are committed.
All	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 and no changes are made.

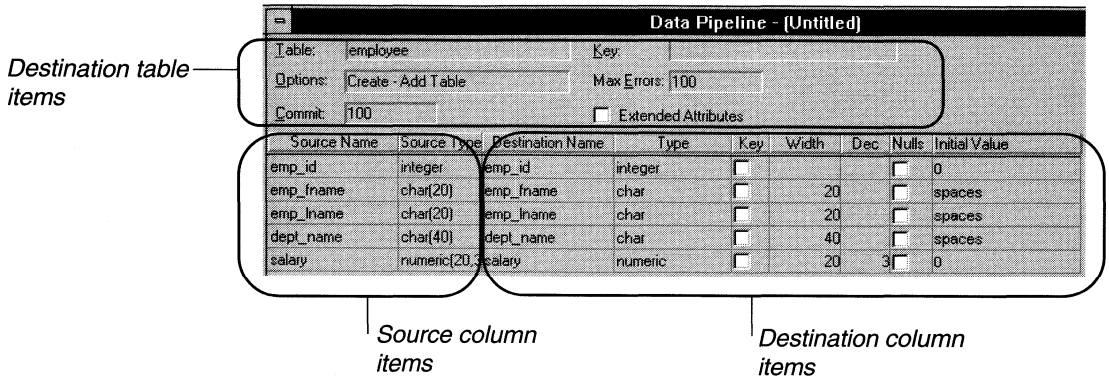
For example, if an error occurs when the 24th row is piped and the Commit value is 10 and the Max Errors value is 1, then:

10 rows are piped and committed
10 rows are piped and committed
3 rows are piped and committed
Piping stops

If the destination database does not support transactions or is not connected in transaction mode, each row that is inserted or updated is committed.

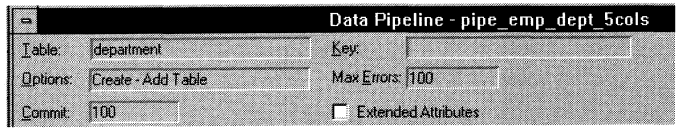
Modifying a data pipeline

After you define the source data in the Select painter, PowerBuilder generates a pipeline definition and displays it in the Data Pipeline painter workspace.



Whether or not you can edit the pipeline definition depends on what pipeline operation you selected and the destination DBMS.

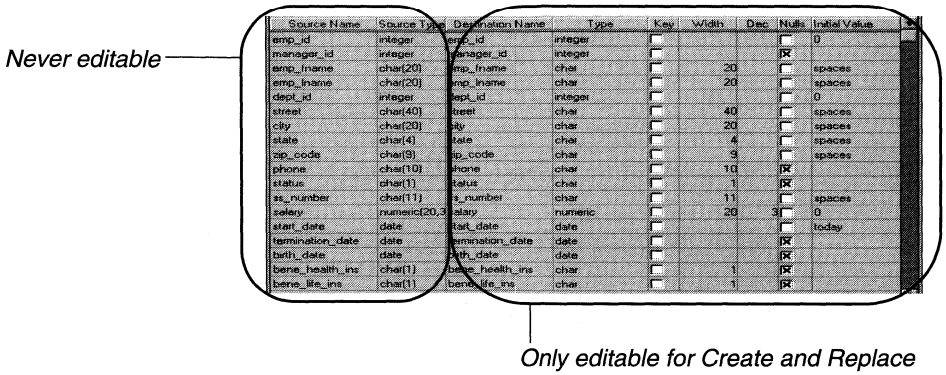
At the top of the workspace The items for the destination table display.



A description of each destination table item is shown below.

Item	Description	Default	How to edit
Table	Name of the destination table.	If source and destination are different, name of first table opened in the Select painter; if the same, *_copy* is appended	For Create or Replace, enter a name. For Refresh, Append, or Update, select a name from the dropdown listbox.
Options	Pipeline operation: Create, Replace, Refresh, Append, or Update.	Create - Add Table	Select an option from the dropdown listbox.
Commit	Number of rows piped to the destination database before PowerBuilder commits the rows to the database.	100 rows	Select a number or *all* from the dropdown listbox.
Key	Key name for the table in the destination database.	If the source is only one table, the table name is followed by *_x*	For Create or Replace, enter a name. Not editable for other pipeline operations.
Max Errors	Number of errors allowed before the pipeline stops.	100 errors	Select a number or *No Limit* from the dropdown listbox.
Extended Attributes	For Create and Replace, a checkbox that specifies whether or not the extended attributes of the selected source columns are piped to the repository of the destination database. Does not display for Refresh, Append, or Update.	Not checked	Click the checkbox.

At the bottom left of the workspace The source column names and data types display. These items are never editable, because you specified them in the Select painter.

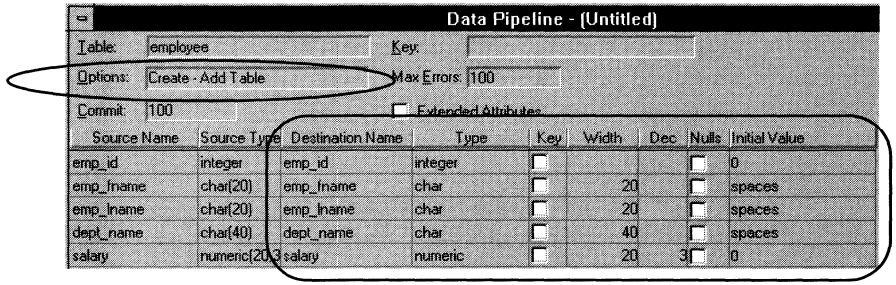


At the bottom right of the workspace Information about the destination table's columns and key displays. These items are editable only for the Create and Replace pipeline operations.

Item	Description	Default	How to edit for the Create and Replace operations
Destination Name	Column name	Source column name	Enter a name.
Type	Column data type	If the DBMS is unchanged, source column data type If the DBMS is different, a best-guess data type	Select a type from the dropdown listbox.
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 are selected)	Select or deselect checkboxes.
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 deselect checkboxes.

Item	Description	Default	How to edit for the Create and Replace operations
Initial Value	Column initial value	Source column initial value (if no initial value, character columns default to *spaces* and numeric columns default to 0)	Enter an initial value.

Using the Create - Add Table option (default)



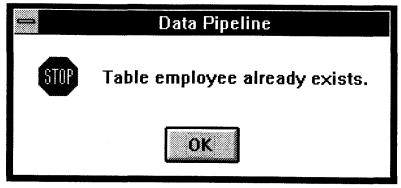
When you select the Create option, you can change these items.

You can	Comment
Change the destination table definition	Follow the rules of the destination DBMS.
Have both a key name and key columns or neither	Select key columns by selecting one or more checkboxes to define a unique identifier for rows.
Allow or disallow NULL for a column	If NULL is allowed (checkbox selected), 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	

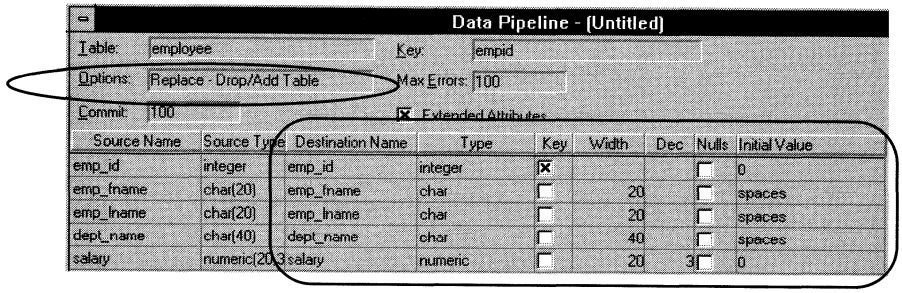
If you have specified key columns and a key name and if the destination DBMS supports primary keys, the Data Pipeline painter creates a primary key for the destination table. If the destination DBMS does not support primary keys, a unique index is created.

For Oracle databases
PowerBuilder generates a unique index instead of a primary key.

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.



Using the Replace - Drop/Add Table option



When you select the Replace option, you can change these items.

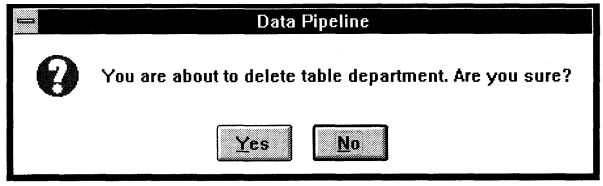
You can	Comment
Change the destination table definition	Follow the rules of the destination DBMS.
Have both a key name and key columns or neither	Select key columns by selecting one or more checkboxes to define a unique identifier for rows.
Allow or disallow NULL for a column	If NULL is allowed (checkbox selected), 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.

You can	Comment
Modify the Commit and Max Errors values	

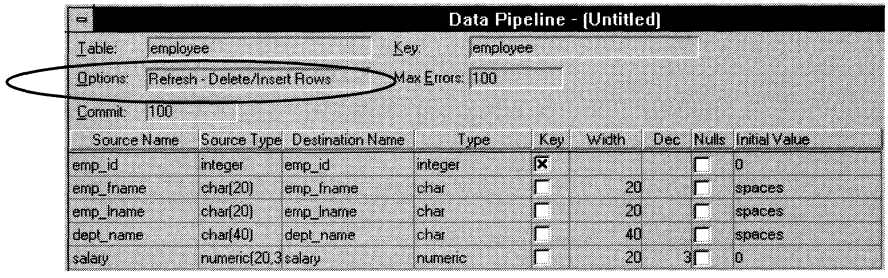
If you have specified key columns and a key name and if the destination DBMS supports primary keys, the Data Pipeline painter creates a primary key for the destination table. If the destination DBMS does not support primary keys, a unique index is created.

For Oracle databases
PowerBuilder generates a unique index instead of a primary key.

When you use the Replace option, PowerBuilder warns you that you are deleting a table, and you can choose another option if needed.



Using the Refresh - Delete/Insert Rows option

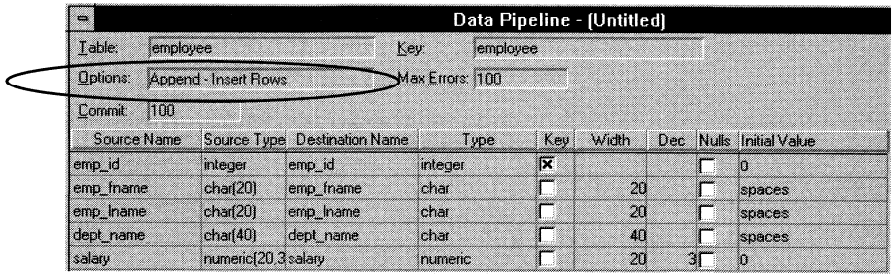


For the Refresh option, the destination table already exists. You can:

- Select an existing table from the dropdown listbox in the Table box
- Modify the Commit and Max Errors values

You cannot change the destination table definition.

Using the Append - Insert Rows option

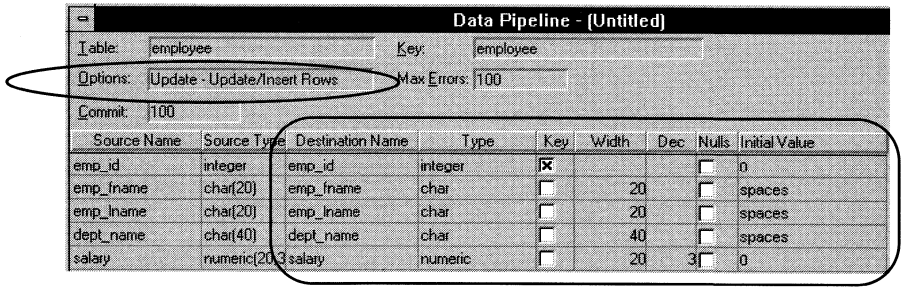


For the Append option, the destination table already exists. You can:

- Select an existing table from the dropdown listbox in the Table box
- Modify the Commit and Max Errors values

You cannot change the destination table definition.

Using the Update - Update/Insert Rows option



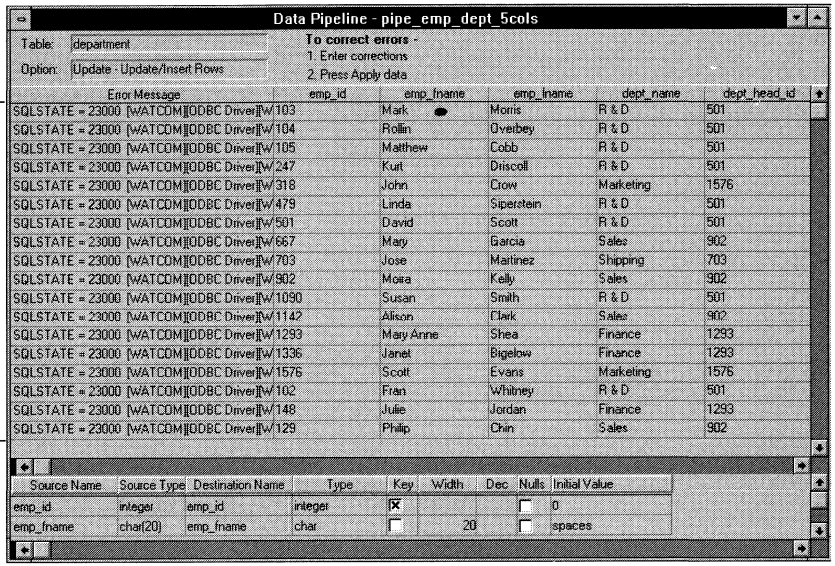
For the Update option, the destination table already exists. You can:

- Select an existing table from the dropdown listbox in the Table box
- Modify the Commit and Max Errors values
- Change the Key columns in the destination table's key (primary key or unique index, depending on what the DBMS supports), but key columns must be selected. The key determines the UPDATE statement's WHERE clause.

Bind variables and the Update option
If the destination database supports bind variables, the Update option takes advantage of them to optimize pipeline execution.

Correcting pipeline errors

If the pipeline can't pipe certain rows to the destination table for some reason, PowerBuilder displays the error rows.



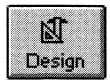
PowerBuilder shows:

- Name of the table in the destination database
- Pipeline operation you chose in the Options box
- Error messages to identify the problem with each row
- Data values in the error rows
- Source and destination column information

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 may 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 workspace without correcting errors:



- Click the Design button.

  The Data Pipeline painter workspace displays.

❖ To correct pipeline errors:

1 Change data values for the appropriate columns for the error rows.

Sometimes you cannot see an entire error message because the column isn't wide enough.

Error Message	emp_id	emp_fname	emp_lname	dept_name	salary
SQLSTATE = 23000 [WATCOM][ODBC Driver][WA102		Fran	Whitney	R & D	45700.000
SQLSTATE = 23000 [WATCOM][ODBC Driver][WA103		Mark	Morris	R & D	20000.000
SQLSTATE = 23000 [WATCOM][ODBC Driver][WA104		Rollin	Overbey	R & D	20000.000

- Move the pointer to the error message and press the RIGHT ARROW key to scroll through a message.
  or
  Drag the Error Message column border to the width needed to read all messages.

Error Message	emp_id
SQLSTATE = 23000 [WATCOM][ODBC Driver][WATCOM SQL integrity contraint violation: primary key for table 'employee' is not unique	102
SQLSTATE = 23000 [WATCOM][ODBC Driver][WATCOM SQL integrity contraint violation: primary key for table 'employee' is not unique	103
SQLSTATE = 23000 [WATCOM][ODBC Driver][WATCOM SQL integrity contraint violation: primary key for table 'employee' is not unique	104

Making the error messages shorter
For ODBC data sources, you can set the DBParm MsgTerse parameter in the destination database profile to make the error messages shorter. If you type:

```
MsgTerse = 'Yes'
```

then the SQLSTATE error number won't appear.

For more information, see *Connecting to Your Database*.

2 Click the Update Database button.
or
Select Options➤Apply Corrections 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.

Saving a pipeline

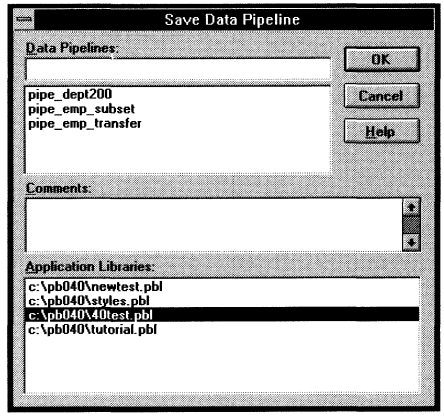
When you have generated a pipeline definition in the Data Pipeline painter workspace, you should save the pipeline. You can then reuse the pipeline at a later time, either in the development environment or in an application you build.

The first time you save a pipeline, you name the pipeline.

❖ To save a pipeline:

1 Select File➤Save from the menu bar.

The Save Data Pipeline dialog box displays.



2 Type a name for the pipeline.

The name can be any valid identifier up to 40 characters.

For information about PowerBuilder identifiers, see *PowerScript Language*.

3 Enter comments for the object.

4 Specify a library to save the object in.

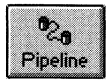
5 Click OK.

The pipeline is saved in the specified library.

Using an existing pipeline

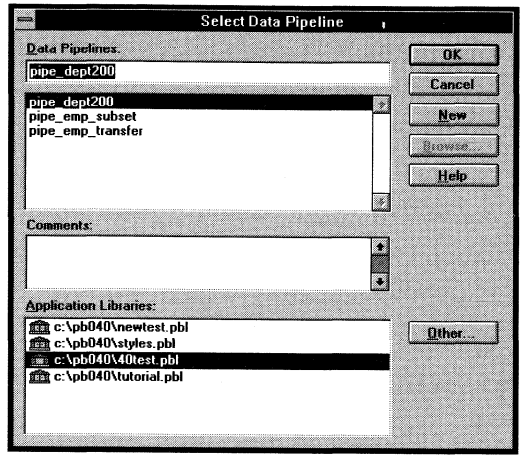
If you save a pipeline, you can modify and execute it at any time. You can also pipe data that may have changed since the last pipeline execution of a pipeline or pipe data to other databases.

❖ To use an existing pipeline:



1 Click the Pipeline button in the PowerBar (if you have customized the PowerBar) or in the PowerPanel.

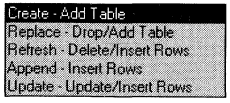
The Select Data Pipeline dialog box displays. Pipelines in the current library are listed.



2 Select the pipeline you want to execute.

3 Click OK.

The Data Pipeline painter workspace displays.



4 If you want to change the pipeline operation, select a new option.

5 Modify the pipeline definition as needed.

6 Click the Execute button.

Examples

Updating data in a destination table

You may want to pipe data and then update the data often.

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.

1 Click the Pipeline button, click New, select 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 In the Where tab, type an expression that will never evaluate to true, such as 1 = 2.

4 Select the Extended Attributes checkbox.

5 Click the Design button.

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.

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 change your destination database each time:

1 Click the Pipeline Profile button.
or
Select File➤Destination Connect from the menu bar.

2 Select the new destination database.

Using pipelines in an application

As you have seen, you can create and execute pipelines in the PowerBuilder development environment. You can also build applications that pipe data, using pipelines you created in the Data Pipeline painter.

For information about piping data in an application, see *Building Applications*.

PART FIVE

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 the user stops it or until an error occurs. The second chapter describes how to package your application for distribution to your users.

CHAPTER 21

Debugging and Running Applications

About this chapter

This chapter describes how to debug and run an application in PowerBuilder. It also lists the errors that can occur at execution time.

Contents

Overview

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.

About the two modes

In **debug mode**, you can insert stops (breakpoints) in scripts and functions, single-step through code, and display the contents of variables to locate logic errors and mistakes that will result in errors during execution.

In **regular mode**, the application responds to user interaction and runs until the user stops it or until an execution-time error occurs. This is the mode you and your users will use to run the completed application.

This chapter describes:

- Running applications in debug mode
- Running applications in regular mode

Debugging an application

Sometimes your application doesn't behave the way you think it will. Perhaps a variable is not being assigned the value you expect, or a script doesn't do what you want it to. In these situations, you can closely examine your application by running it in debug mode.

You use debug mode to set **stops** (breakpoints). When you run your application in debug mode, PowerBuilder stops execution just before it hits a line containing a stop. You can then look at (and change) the values of variables.

❖ To debug an application:

1 Open the Debug window.

2 Set stops at places in your application where you have a problem.

3 Run the application in debug mode.

4 When execution is suspended at a stop, look at the values of variables or change values.

5 Step through the code line by line if you want.

6 As needed, add or modify stops in the middle of running your application.

7 When you uncover the problem, fix your code.

These steps are all described below.

Opening the Debug window

❖ To open the Debug window:

1 Click the Debug button in the PowerBar or PowerPanel.
or
Select File➤Debug from the menu bar.

> **Changing applications**
> To run an application other than the current application, change the application before you open the Debug window.

If you are in a painter with unsaved work, you are asked whether you want to save the object.

2 If necessary, save your work.

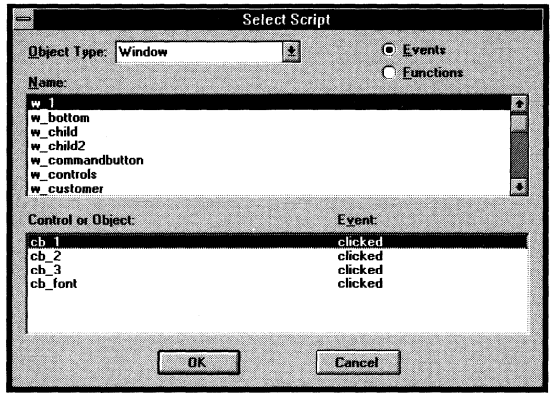
You go to the PowerBuilder Debug window.

If no stops are currently defined, the Select Script dialog box displays (see the next section).

If one or more stops are defined, the Edit Stops dialog box displays (see "Editing stops" on page 760).

Adding stops

To add a stop, you first use the Select Script dialog box to specify the script containing the line you want to stop execution at.



Initially, the Select Script dialog box lists the windows in the current application and the controls in the selected window that have scripts (only events having scripts are listed).

❖ To add a stop:

1 Select the object type in the Select Script dialog box. To add a stop in a window, leave the value in the Object Type box as is. You can also choose to add a stop to a menu, user object, global function, or to the application object by changing the selection in the Object Type box.

2 Specify whether you want to add a stop in a script for an event or in an object-level function by selecting the Events or Functions radio button.

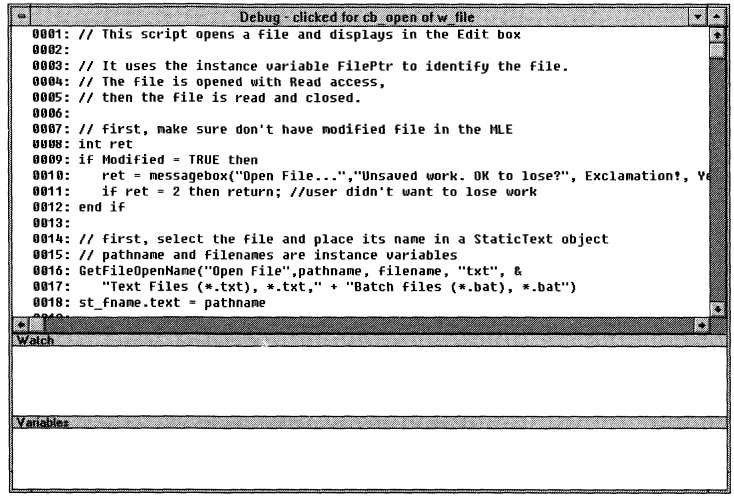
3 Select the object containing the script or function you want to add the stop for in the Name box.

4 If you are adding a stop for a script, select the control and event in the Control Or Object box.

If you are adding a stop for an object-level function, choose the function in the Function box.

5 Click OK.

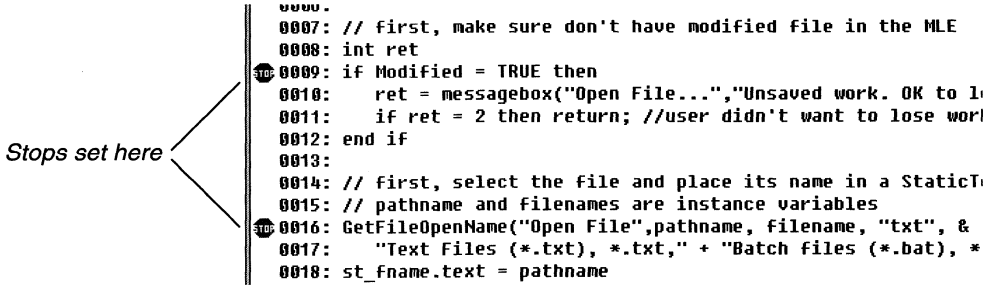
The Debug window displays with the script or function listed.



6 Double-click each line in which you want to insert a stop.

A stop sign displays at the start of each line that has a stop.

Select executable lines
Insert stops in lines that contain an executable statement; do not insert stops in variable-declaration lines, comment lines, or blank lines.



7 To remove an existing stop, double-click on the line containing the stop.

When PowerBuilder runs your application in debug mode, it stops just before executing a line containing a stop.

Tips

Do not set a stop in the Activate or GetFocus event (going to and returning from the Debug window can cause recursive triggering of the events).

To be able to see values for variables assigned in the last line of a script, add a dummy executable line at the end of the script and set a stop there.

Stops are remembered

When you close Debug, PowerBuilder saves the information about the stops as Stop variables in the Debug section of PB.INI. The next time you debug your application, whether in the current PowerBuilder session or another one, the stops will be used.

Editing stops

You can add and modify stops when you first go to the Debug window or anytime in a debugging session. In either case, you use the Edit Stops dialog box.

❖ To open the Edit Stops dialog box:

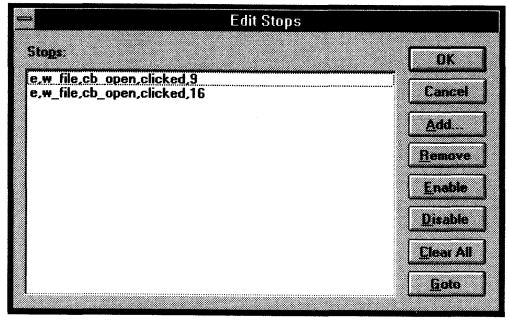
1 Do one of the following:

- Open the Debug window, as described above.

- If you are already in the Debug window and want to set a new stop or modify an existing one, click the Edit Stops button in the PainterBar.

A shortcut
To add a new stop from within the Debug window workspace, you can also click the Select Script button in the PainterBar, which displays the Select Script dialog box.

The Edit Stops dialog box displays, listing the defined stops.



2 Do one of the following:

To	Do this
Add a new stop	Click the Add button and follow the procedure described in "Adding stops" on page 758
Modify an existing stop	Select the stop, then click the Goto button (see "Modifying stops" on page 762)
Enable or disable a stop	Select the stop, then click the Enable or Disable button (see "Enabling or disabling stops" on page 762)
Delete a stop	Select the stop, then click the Remove button
Delete all stops	Click the Clear All button

3 When you have completed specifying stops, click OK.

The Edit Stops dialog box closes and you return to the Debug window.

The information displayed in the Edit Stops dialog box

The Edit Stops dialog box lists all stops currently defined using the following format:

state,object,control,event,lineno

where:

- *state* is the state of the stop (enabled or disabled)
- *object* is the name of object that contains the stop (a window, menu, user object, global function, or the application object)
- *control* is the name of the control, MenuItem, or user object whose script contains the stop
- *event* is the name of the event with the stop
- *lineno* is the number of the line in the script at which the stop is set

(*Control* and *event* are not used when showing stops for functions.)

Modifying stops

When you click Goto in the Edit Stops dialog box, the script or function displays. You add or remove stops by double-clicking on the appropriate lines.

Enabling or disabling stops

You can disable stops without deleting them. PowerBuilder will not stop execution at disabled stops. Later you can enable the stop again.

❖ To disable a stop:

- Select the stop in the Edit Stops dialog box and click Disable.

  PowerBuilder grays the Stop Sign button indicating disabled stops when displaying scripts.

❖ To enable a disabled stop:

- Select the stop in the Edit Stops dialog box and click Enable.

  The stop is active again. PowerBuilder will stop execution just before it hits the stop.

Deleting stops

You can remove stops you have set.

❖ **To remove one stop:**

- Do one of the following:
  - In the Edit Stops dialog box, select the stop and click the Remove button.
  - When looking at the script or function itself in the Debug window's workspace, double-click on the line containing the stop.

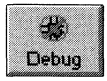
❖ **To remove all stops:**

- Click the Clear All button in the Edit Stops dialog box.

Running in debug mode

Once you have set your stops, you can run the application in debug mode. The application will proceed as normal until it hits a line containing a stop.

❖ **To run in debug mode:**



1 If necessary, open the Debug window by clicking the Debug button.

2 Click the Start button in the PainterBar.
or
Select Run➤Start from the menu bar.

The application starts and proceeds until it hits a stop. The application is then suspended and you return to the Debug window, with the line containing the stop displayed.

3 You can examine the state of the application at this point (see "Viewing information while stopped" next).

4 To execute the next statement and stop, click the Step button.

Debug executes the current line, then stops and returns you to the Debug window.

5 To continue execution, click the Continue button.

The application continues until the next stop is hit.

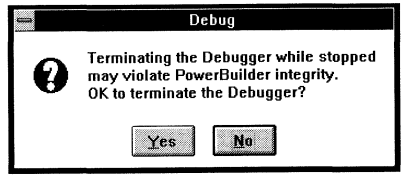
6 Proceed through your application. Terminate it normally once you have learned what you needed to know.

You return to the Debug window.

7 To close the Debug window, select File➤Close from the menu bar.

Leaving while stopped

You are allowed to close the Debug window while stopped at a breakpoint, but doing so may cause unexpected problems later. PowerBuilder displays the following dialog box when you attempt to close the Debug window while stopped at a breakpoint:

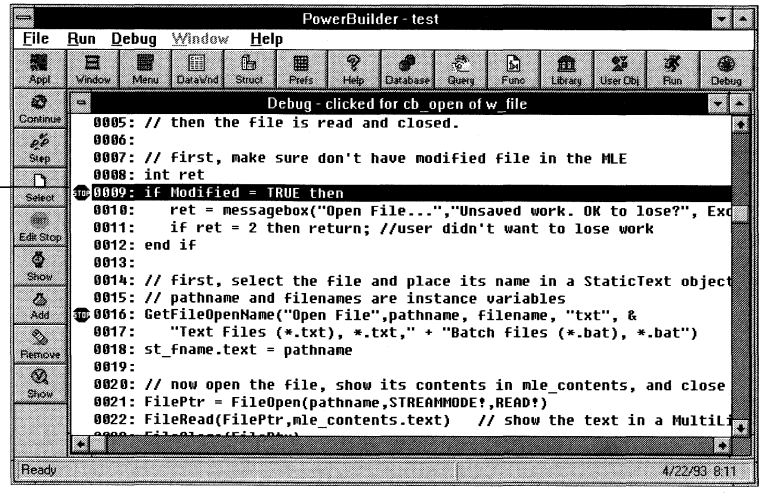


The Debug window closes (but information about all the stops and watch variables is retained).

Viewing information while stopped

When Debug encounters a stop, it suspends the application before executing the statement with the stop, then displays the Debug window. The statement that will be executed next is highlighted.

Execution stopped here



At this point, you can look at the state of your application. You can:

- Display objects in the current application and the current values of the instance variables and attributes of the objects
- Display the current values of the global, shared, and local variables
- Change the values of variables
- Select the variables you want to watch
- Edit the stops
- Print the variables and watch variables lists

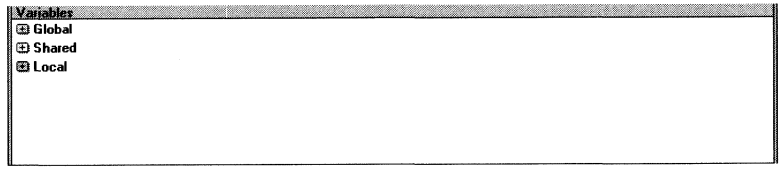
You can also continue executing the application (either one step at a time or at normal speed) or select another script to debug.

Displaying variables

❖ To display the current values of variables:

1 Click the Show Variables button in the PainterBar.
or
Select Debug➤Show Variables from the menu bar.

The Variables window displays.



Setting is saved
When you close Debug, PowerBuilder saves the setting as the VariablesWindow Debug variable in PB.INI. This variable determines whether the Variables window displays when the Debug window opens.

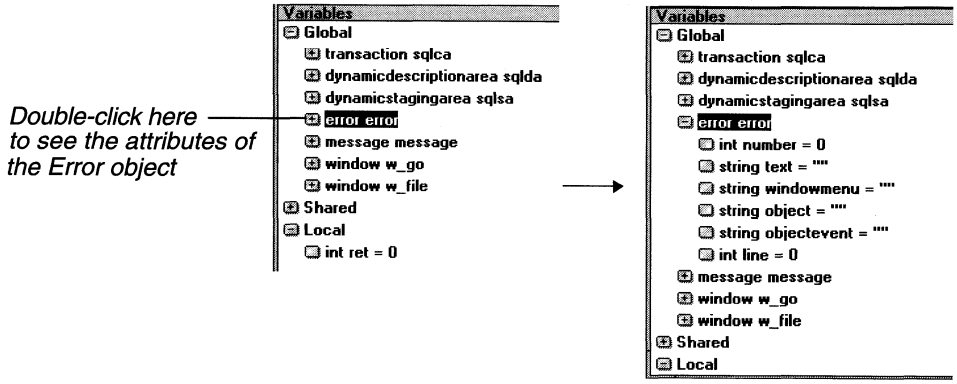
2 Display the variables whose values you want to look at, as follows:

To look at	Do this
Global variables	Double-click the Global button. PowerBuilder displays all global variables defined for the application and lists all objects (such as windows) that are open.
Shared variables	Double-click the Shared button. PowerBuilder displays all objects that have been opened so far. Click an object to see its shared variables.
Variables that are local to the current script or function	Double-click the Local button.
Instance variables of an object	Locate the variable in the object itself. See "Looking at instance variables and attributes" on page 767.

Looking at long strings
The Debug window displays the first 128 characters of a variable's value. If you want to see a value that exceeds 128 characters, double-click the variable as if you were going to modify the value. You can scroll through the contents in the resulting window.

Using the Variables window

The Variables window displays information in an expandable outline. If there is information below a displayed entry, the entry displays with a + next to it. Double-click the entry to see the information.



Looking at instance variables and attributes

Instance variables are part of an object's definition, so PowerBuilder lists instance variables as attributes of the object itself.

❖ To look at instance variables and attributes for an object:

1 Double-click the Global button.

All global variables and open objects display.

2 Double-click the object containing the instance variable.

All attributes for the selected object display. Instance variables are displayed at the end of the list.

3 Scroll to the end of the list to examine the instance variables.

Debugging windows opened as local variables

One way to open a window is through declaring a local variable of type window and opening it through a string, such as:

```
window mywin
string named_window
named_window = sle_1.Text
Open(mywin, named_window)
```

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. If you want to debug windows opened this way, you can declare a global variable of type window and assign it the local variable. For example, assuming GlobalWindow is a global window of type window, you could add the following line at 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.

Using a watch list

You can select variables you want to watch as the application proceeds. PowerBuilder displays these variables in a Watch window and updates their values as the application runs.

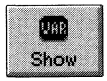
Watch variables are remembered

PowerBuilder records which variables you selected as watch variables as Watch variables in the Debug section of PB.INI. The next time you debug your application, whether in the same PowerBuilder session or another one, the watch variables will be used.

Creating a watch list

There are two ways to create a watch list. Use the first method to add a few watch variables one at a time. Use the second method to add groups of variables at once, then remove the ones you don't want.

❖ To select the variables you want to watch one at a time:



1 From the Debug window, open the Variables window by clicking the Show Variables button in the PainterBar or by selecting Debug➤Show Variables from the menu bar.

2 Click the Show Watch button in the PainterBar.
or
Select Debug➤Show Watch from the menu bar.

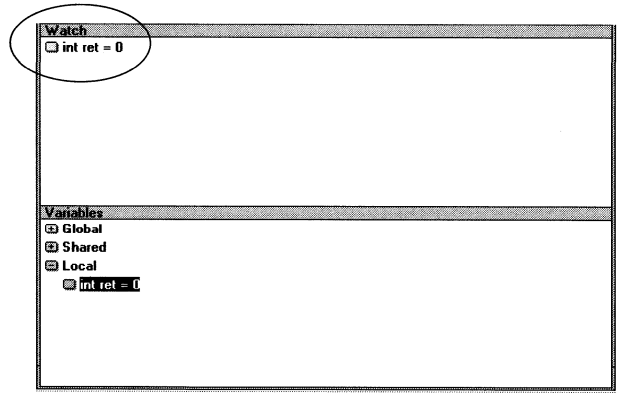
The Watch window displays.

> **Setting is saved**
> When you close Debug, PowerBuilder saves the variable WatchWindow in the Debug section of PB.INI. This variable determines whether the Watch window displays the next time the Debug window displays.

3 Select a variable that you want to watch by clicking the variable name in the Variables window.

4 Click the Add button in the PainterBar.

PowerBuilder adds the variable to the watch list and displays it in the Watch window.



> **Shortcut**
> You can SHIFT-click a variable in the Variables window to add it to the Watch window in one step.

❖ To select the variables you want to watch by group:

1 From the Debug window, open the Variables window by clicking the Show Variables button in the PainterBar or by selecting Debug➤Show Variables from the menu bar.

2 Display the Watch window by clicking the Show Watch button in the PainterBar or by selecting Debug➤Show Watch from the menu bar.

3 In the Variables window, click the name of the variable type you want to watch (Global, Shared, or Local).

4 Select Debug➤Add Watch from the menu bar.

PowerBuilder displays the name of the selected variable type in the Watch window.

5 Expand the variable list in the Watch window by clicking the button next to the name of the variable type.

6 Remove the variables you do not want to watch from the list by selecting the variable and clicking the Remove button or selecting Debug➤Remove Watch from the menu bar.

PowerBuilder removes the selected variables from the watch list.

Sizing windows
To see more lines in the script, you can close the Variables window after selecting the watch variables. To size the Watch window, drag the bar at the top of the Watch window.

Changing variable values

While running an application in debug mode, you can change the values of variables, then continue the application. You can use this technique to easily examine different flows throughout the application.

Limitation
You cannot change the values of enumerated variables.

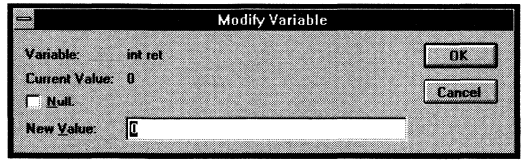
❖ To change a variable's value:

1 Run the application in debug mode.

2 When at a stop, display the variable whose value you want to change in either the Watch window or the Variables window.

3 Double-click the variable.

The Modify Variable dialog box displays.



4 Enter the new value for the variable.

5 Click OK.

PowerBuilder closes the Modify Variable dialog box and changes the value of the selected variable.

When you continue the application, the new value will be used.

Printing variable values

You can print the lists of variables and watch variables on your default printer. If you want to use a different printer, select Printer Setup on the File menu before printing.

❖ To print variable values while at a stop:

1 To print the watch variables and their values, select Debug➤Print Watch from the menu bar.

2 To print the variables and values displayed in the Variables window, select Debug➤Print Variables from the menu bar.

Fixing your code

If you find an error in a script or function while using the debugger, you can fix the problem, as follows.

❖ To fix a problem:

1 Terminate your application.

You return to the Debug window.

2 Close the Debug window by selecting File➤Close from the menu bar.

> **Caution**
> *Be sure to close the Debug window before making any changes to your application.*

3 Open the appropriate painter.

4 Fix the script or function.

5 To check the fix, you can run the application in debug mode again by clicking the Debug button in the PowerBar and clicking the Start button.

All stops and watch variables you previously set are still defined.

Running an application

When the application seems fine, 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 an execution-time error occurs. You can rely on the default execution-time error reporting by PowerBuilder or write a script that specifies your own error processing.

Running the application

❖ **To run the current application:**

- Click the Run button in the PowerBar or PowerPanel.
  or
  Press CTRL+R.
  or
  Select File➢Run from the menu bar.

  PowerBuilder becomes minimized (its icon displays at the bottom of the screen along with the icons of other minimized applications), and your application executes.

❖ **To stop a running application:**

- Either end the application normally, or double-click the PowerBuilder icon to stop it immediately.

Handling errors during execution

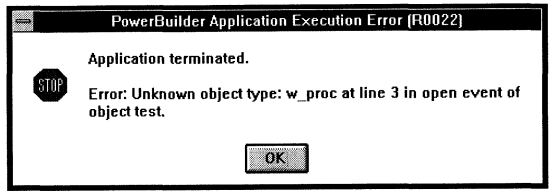
A serious error during execution (such as attempting to access a window that has not been opened) will trigger the SystemError event in the application object.

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. It is a good idea to build an application-level script for the SystemError event to trap and process any execution-time errors, as described next.

Using the Error object

In the script for the SystemError event, you can access the built-in Error object to learn which error occurred and where it occurred. The Error object contains the following attributes:

Attribute	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, Object will be the same as WindowMenu.

Attribute	Data Type	Description
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 attributes and define object-level functions for your Error object to allow for additional processing. In the Application painter, you specify that you want to use your user object inherited from Error as the global Error object in your application.

For more information, see Chapter 10, "Working with User Objects."

Execution-time error numbers

The following table lists the execution-time error numbers returned in the Number attribute of the Error object and the meaning of each number:

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, must be converted to current version
12	DataWindow column type does not match GetItem type
13	Unresolved attribute reference

Number	Meaning
14	Error opening DLL library for external function
15	Error calling external function
16	Maximum string size exceeded
17	DataWindow referenced in DataWindow object does not exist
18	Function doesn't return value
19	Cannot convert *name* in Any variable to *name*
20	Database command has not been 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 doesn't match its definition
25	Double or Real expression has overflowed
26	Field *name* assignment not supported
27	Cannot take a negative to a non-integer power
28	VBX Error: *name*
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 attribute *name*
37	Type mismatch accessing external object attribute *name*
38	Incorrect number of subscripts accessing external object attribute *name*
39	Error accessing external object attribute *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*

Number	Meaning
50	Application reference could not be resolved
51	Failure loading dynamic library
52	Missing ancestor object *name*
53	Missing ancestor object *name* in *name*
54	Conflicting ancestor object *name* in *name* and *name*
55	Window close occurred processing yield function
56	Database Interface does not support Remote Procedure Calls
57	Database Interface does not support Array variables (function *name*)
58	Blob variable for *name* cannot be empty
59	Maximum size exceeded

Errors 50 and above 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 attributes of the Error object by using an assignment statement and PowerBuilder dot notation (for example, **Error.Line=45**).

2 Call the SignalError function to trigger the SystemError event.

The script for the SystemError event executes.

For information about the SignalError function, see the *Function Reference.*

CHAPTER 22

Creating an Executable

About this chapter

This chapter describes how to create an executable version of your application.

Contents

Overview

You want your users to be able to execute your application the same way they execute other applications on their computer. To do that, you create an executable version of your application.

You have two basic ways to package your application:

- As one standalone executable file that contains all the objects in the application
- As an executable file and one or more PowerBuilder dynamic libraries that contain objects that are linked at execution time

You might also need to provide some additional resources that your application uses, such as bitmaps and icons. There are two ways to provide resources: you can include them in your executable and/or dynamic libraries, or you can distribute them separately.

This chapter describes how to use PowerBuilder to create an executable version of your application.

For more information

The *Building Application* manual describes executable files, PowerBuilder dynamic libraries, and resources. It also presents the various strategies for packaging your application. Read it to gain an understanding of the best way for you to package your application. Then follow the procedures in this chapter to implement your strategy.

Two ways to build an executable

There are two ways to produce executables in PowerBuilder:

- You can use the Project painter to create a project object that specifies the executable file name, PowerBuilder dynamic libraries (PBD files), and PowerBuilder resource files (PBR files). This method allows you to create a new executable file and dynamic libraries without having to redefine the components of the application each time you build it.
- Use the Application painter and each time specify which libraries are dynamic. Then go to the Library painter and build each of the dynamic libraries you need.

Which to use

If you are working on a substantial project that includes dynamic libraries that you expect to rebuild a number of times, using the Project painter to create a project object will save you a great deal of time. You don't have to keep specifying which libraries are dynamic or manually rebuild dynamic libraries specifying the appropriate resource files.

If you have a simple application that you will only need to build once and doesn't use dynamic libraries, you might want to create the executable in the Application painter—you probably don't need to create a project object.

Defining a project

This section describes how to create an executable application using the Project painter.

About the Project painter

The Project painter allows you to streamline the generation of executable files and PowerBuilder dynamic libraries. You use the Project painter to create and maintain PowerBuilder project objects. 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 PowerBuilder dynamic libraries (PBD files)
- Which PowerBuilder resource files (if any) should be used to build the executable file and the dynamic libraries

Once you have defined the project, you can rebuild your application from the Project painter by simply clicking the Build button.

Building a project object for your application can greatly reduce the amount of time spent creating an executable, PBDs, and PBRs for every build you want to generate.

Using a version control system
If you are using a version control system to manage your PowerBuilder objects, you can also use the Project painter to restore previous versions of your libraries.

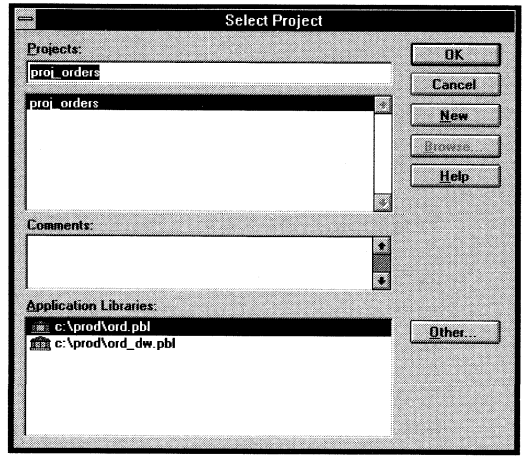
For more information, see *Version Control Interfaces*.

Defining a project object

❖ To define a project object:

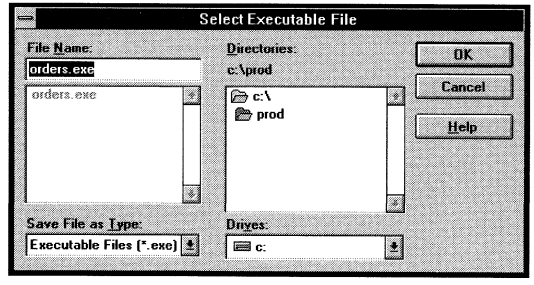
1 Open the Project painter by clicking the Project button in the PowerBar or PowerPanel.

The Select Project dialog box displays.



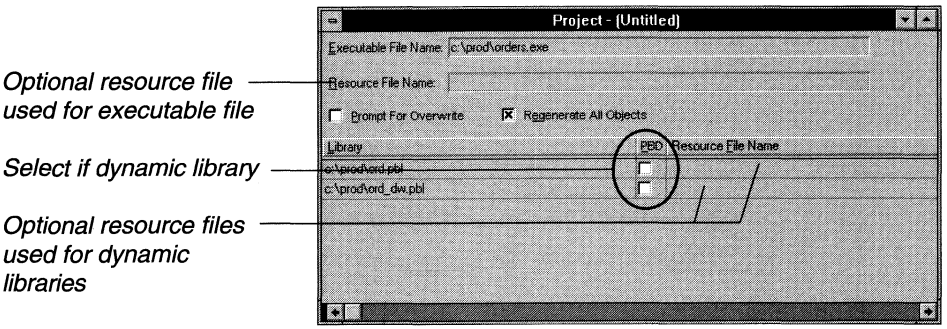
2 Click New to create a new project object.

The Select Executable File dialog box displays.



3 Enter a filename for the executable file you want to create and click OK.

The Project painter workspace displays. It displays the executable name you specified and lists all libraries defined for the current application.



4 Specify a PowerBuilder resource file (PBR file) beneath the name of the executable if necessary. You need to use a resource file 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 Resource File Name box or select Edit➤Paste Executable from the menu bar to browse your directories for the resource file you want to include.

For more about PowerBuilder resource files, see "Distributing resources" on page 795.

5 Select Prompt for Overwrite if you want PowerBuilder to prompt you before overwriting any files it creates when building your application.

6 Select Regenerate All Objects if you want PowerBuilder to always regenerate all objects in the application libraries before it creates the executable and dynamic libraries. If you leave this checkbox unselected, PowerBuilder does not regenerate the objects. In most cases, you will want to regenerate your objects before rebuilding your project. This ensures that the objects are fine.

7 Choose which libraries to include as PowerBuilder dynamic libraries (PBDs). To define a library as a PowerBuilder dynamic library to be distributed with your application, select the PBD checkbox.

For more about PowerBuilder dynamic libraries, see "Using dynamic libraries" on page 786.

8 For each of the dynamic libraries you want to create, you can specify a PowerBuilder resource file. You need to define a resource file for a dynamic library if it uses resources and you want the resources included in the dynamic library instead of having to distribute the resources separately.

For more about PowerBuilder resource files, see "Distributing resources" on page 795.

9 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 Library painter.

10 Close the Project painter by selecting Close from the File menu or from the Control menu.

Using dynamic libraries

You can store the objects used in your distributed PowerBuilder application in more than one library and at execution time 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 PowerBuilder dynamic libraries (PBD files).

When PowerBuilder builds a dynamic library, it copies the compiled versions of *all objects* from the source library (PBL file) into the dynamic library (PBD file). The easiest way to specify source libraries is to simply use your standard PowerBuilder libraries as source libraries.

However, then your dynamic libraries are larger than they need to be because they include all objects from the source library, not just the ones used in your application. You might want to create a PowerBuilder library that contains only the objects that you want to be in a dynamic library.

❖ **To create the source 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 Library➤Create from the menu bar, then move the objects into the new library by selecting them and then selecting Entry➤Move from the menu bar.

2 Make sure the application's library search path includes the new library.

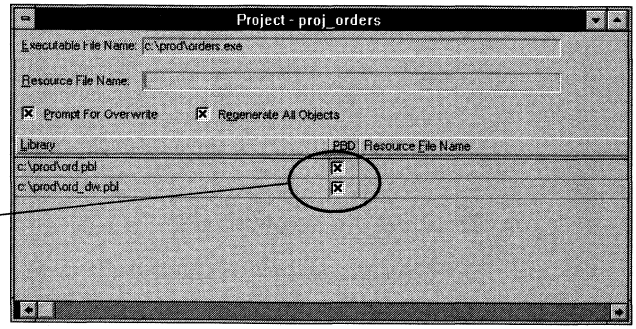
Multiple dynamic libraries
You can use as many dynamic libraries as you want in an application. Just create a source library (PBL file) for each of them.

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 (PBD files).

❖ **To specify the dynamic libraries:**

- In the Project painter, select each of the libraries that will be dynamic.

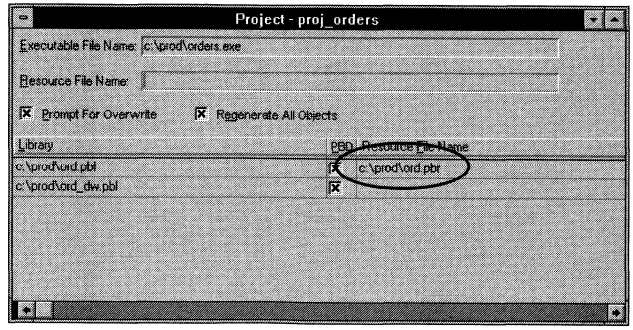


Including additional resources for a dynamic library

When building a dynamic 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 don't 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 dynamic library when it builds it.

To reference these resources, list the resources in a PBR file as described in "Using PowerBuilder resource files" on page 795.

Then use the Resource File Name box to reference the PBR file in the dynamic library.



Building a project

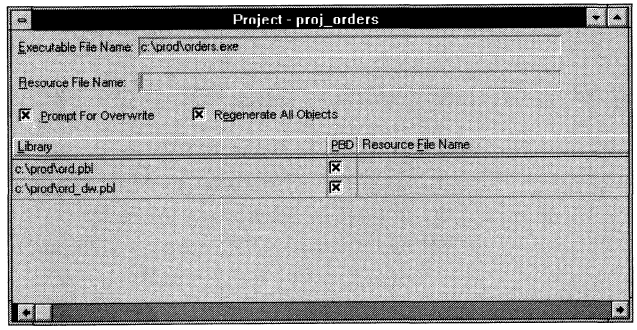
Once you have defined your project, you build it—that is, you tell PowerBuilder to create the executable files and all specified dynamic libraries. You can build your project any time you have made changes to the objects and want to test or deploy another version of your application.

You can build a project only for your current application.

❖ To build the application:

1 Open the project you built in the Project painter.

The Project painter workspace displays.



2 Click the Build button in Project PainterBar.

> **If the application's library list has changed**
> When you click Build, PowerBuilder checks your application'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.

This process is described in detail in the next section.

How PowerBuilder builds the project

Here is what happens when PowerBuilder builds your project:

- If you selected Regenerate All Objects, PowerBuilder first regenerates all the objects in the libraries.
- If you selected Prompt for Overwrite, PowerBuilder displays a message box asking for confirmation before overwriting the executable file and each dynamic library.

To create the executable file you specified, PowerBuilder searches through your application and copies the compiled versions of *referenced* objects from the libraries in the application's library search path into the executable file. It does not copy objects that are *not referenced* in the application to the executable file.

Also, it does not copy objects to the executable file from libraries you declared to be PowerBuilder dynamic libraries. These objects are linked to the application at execution time; they are not stored in the executable file. Instead, it creates a dynamic library (PBD file) for each of the specified libraries and maintains a list of all the PBD files that were specified for the application (PowerBuilder maintains the unqualified filenames of the PBD files; it does not save the path name).

The generated PBD files have the same names as the corresponding PBL files. For example, PowerBuilder creates TEST.PBD from TEST.PBL.

What happens during execution
When an object such as a window is referenced in the application, PowerBuilder first looks in the executable file for the object. If it doesn't find it there, it looks in the PBDs that are defined for the application. For example, if you specified that TEST.PBL is a dynamic library, PowerBuilder will look for TEST.PBD at execution time. The PBDs must be on the search path. If PowerBuilder can't find the object in any of the PBD files, it reports an execution-time error.

How PowerBuilder searches for objects

When searching through the application, PowerBuilder doesn't necessarily find all the objects that are used in your application and copy them to the executable file. This section describes which objects it finds and copies and which ones it doesn't.

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

For example:

- If a window script contains the following statement:

  ```
  Open(w_continue)
  ```

  Then w_continue is copied to the executable file.

- If a menu item script refers to the global function f_calc:

  ```
  f_calc(EnteredValue)
  ```

  Then f_calc is copied to the executable file.

- If a window uses a popup menu through the following statements:

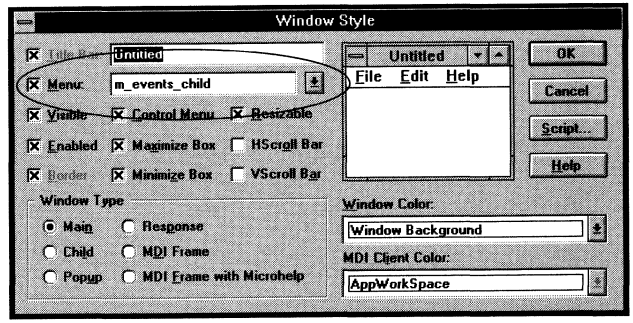
  ```
  m_new    mymenu
  mymenu = create m_new
  mymenu.m_file.PopMenu( PointerX(), PointerY() )
  ```

  Then m_new is copied to the executable file.

Objects that are referenced in painters

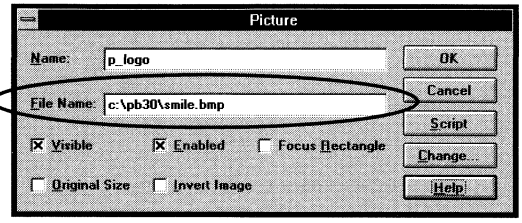
For example:

- If a menu is associated with a window in the Window painter, the menu is copied to the executable file as a result of the window being copied.



- 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, each user object is copied.
- Resources assigned in a painter. For example, when you place a Picture control in a window in the Window painter, you can associate a bitmap file with it.



Also, you can associate an icon with the application in the Application painter.

Resources assigned this way are copied to the executable file.

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 (the compiler saves this information).

But it does not identify objects that are referenced dynamically through string variables (it would have to read through all the scripts and process all assignment statements to uncover all the referenced objects).

For example:

- If the DataWindow object d_emp is associated with a DataWindow control dynamically through the following statement:

```
dw_info.DataObject = "d_emp"
```

Then d_emp is not copied to the executable file.

- If a resource is assigned dynamically in scripts, such as:

```
if Balance < 0 then
   p_logo.PictureName = "frown.bmp"
else
   p_logo.PictureName = "smile.bmp"
end if
```

  Then FROWN.BMP and SMILE.BMP are not copied to the executable file.

- If a window script has the following statements:

```
window   mywin
string   winname = "w_go"
Open(mywin,winname)
```

  Then the reference to window w_go is *not* found by PowerBuilder when building the executable file, so w_go is not copied to the executable file.

Which objects are not copied to the dynamic libraries

When building a dynamic library, PowerBuilder does not inspect the objects; it simply removes the source form of the objects. Therefore, the 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.

How to include the objects that were not found

If you didn't use any of the types of references described in the preceding sections, you don't need to do anything else to ensure that all objects get distributed: they were all built into the executable file. Otherwise, you have the following choices for how to include the objects that were not found.

Distributing resources

For resources such as icons and bitmaps, you have two choices:

- Distribute them separately.
- Include them in a PowerBuilder resource file, 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 PowerBuilder resource file, then build an executable file or dynamic PowerBuilder library using the resource file.
- Include them directly in a dynamic PowerBuilder library.

Distributing other objects

All other objects (such as windows referenced only in string variables) must be included directly in a dynamic PowerBuilder library.

Summary of distribution possibilities

Object	Separately	Through resource file	Directly in dynamic library
Resource	✓	✓	
DataWindow object		✓	✓
Other used object			✓

For information about distributing resources, see the next section.

For information about dynamic PowerBuilder libraries, see "Using dynamic libraries" on page 786.

Listing the objects in a project

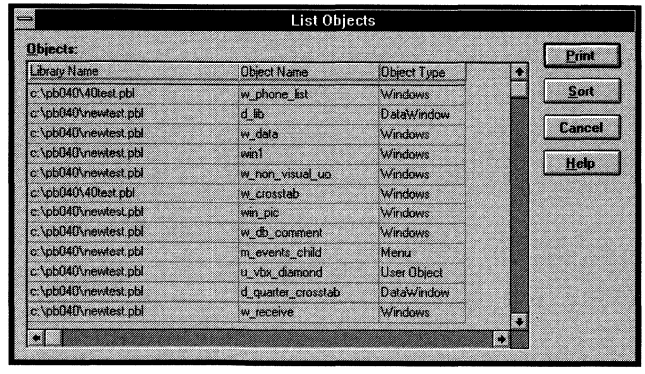
After you have built your project, you can display a list of objects in the project.

❖ To list the objects in a project:

1 Build your project.

2 Select Options➤List Objects from the menu bar.

The List Objects dialog box displays.



The dialog box lists the objects that PowerBuilder placed in the executable file and dynamic libraries it created when it built the project.

What's in the report

The report is a grid DataWindow with the following columns:

Column	Meaning
Library Name	Source library that contains the object
Object Name	Name of the object
Object Type	Type of object

Using version control

If you are using a version control system to manage your PowerBuilder objects, there are additional columns in the report, which map the objects in the project to your archive files.

For more information, see *Version Control Interfaces*.

What you can do

Because the report is a grid DataWindow, you can resize and reorder columns just as you can in other grid DataWindows.

You can also sort the rows and print the report using the Sort and Print buttons.

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

If a resource has not been included in the executable file or in a PowerBuilder dynamic library, when a resource is referenced at execution time PowerBuilder looks in the search path for the resource (such as the file FROWN.BMP). So you need to distribute the resources with your application and make sure they get installed in the user's search path.

The Windows search path

The Windows search path is the current directory, the Windows directory, the Windows System subdirectory, and all directories on the DOS path.

Example

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 could distribute the files FROWN.BMP and SMILE.BMP with your application. As long as the files are on the search path at execution time, the application can load them when they are needed.

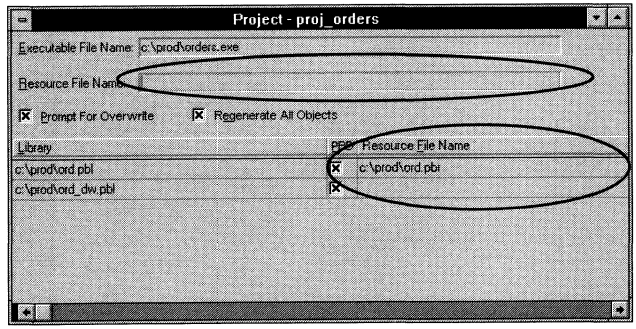
Using PowerBuilder resource files

Instead of distributing resources separately, you can create a PowerBuilder resource file (a PBR file) that lists all dynamically assigned resources. You can also include dynamically assigned DataWindow objects in PBR files. PowerBuilder compiles the listed resources into the executable file or a PBD file, so the resources are available directly at execution time.

Using dynamically assigned DataWindow objects
If you are assigning DataWindow objects to DataWindow controls dynamically in scripts, you *must* create a PowerBuilder resource file and include the objects in the executable or PBD file, or include the DataWindow objects themselves in a PBD file. You cannot distribute them separately (as you can, for example, bitmaps and cursors).

❖ To create and use a PowerBuilder resource file:

1. Using a text editor, create a text file that lists all resource files referenced dynamically in your application (see below for information about creating the file).When creating a resource file for a PowerBuilder dynamic library, list *all* resources used by the dynamic library, not just those assigned dynamically in a script.
2. Specify the resource files in the Project painter. The executable file can have a resource file attached to it as can each of the dynamic libraries.



When PowerBuilder builds the project, it includes all resources specified in the PBR file in the executable file or dynamic library. You no longer have to distribute your dynamically assigned resources separately; they are in the application.

Creating the PowerBuilder resource file

A PBR file is an ASCII text file in which you list resource names (such as BMP, CUR, ICO, RLE, and WMF files) and DataWindow objects. To create a PBR file, use a text editor. List the names of each resource, one resource on each line, then save the list as a file with the extension PBR.

Naming resources

If the resource file is in the current directory, you can simply list the file, such as:

```
FROWN.BMP
```

If the resource file is in a different directory, include the path to the file, such as:

```
C:\BITMAPS\FROWN.BMP
```

Important information about naming resources

The filename specified in the PBR file must exactly match the way the resource is referenced in scripts. If the reference in the script uses a path, you must specify the same path in the PBR file. If the resource file is not qualified with a path in the script, it must not be qualified in the PBR file.

For example, if the script reads:

```
p_logo.PictureName = "FROWN.BMP"
```

then the PBR file must read:

```
FROWN.BMP
```

If the PBR file says something like:

```
C:\MYAPP\FROWN.BMP
```

and the script doesn't specify the path, PowerBuilder will not find the resource at execution time. That is because PowerBuilder does a simple string comparison at execution time. In the preceding example, when it executes the script it will look for the object identified by the string "FROWN.BMP" in the executable file. It won't find it, because the resource is identified in the executable file as "C:\MYAPP\FROWN.BMP".

In this case, the picture will not display at execution time; the control will be empty in the window.

Including DataWindows objects in a PBR file

To include a DataWindow object in the list, enter the name of the library (with extension PBL) followed by the DataWindow object name enclosed in parentheses. For example:

```
sales.pbl(d_emplist)
```

If the DataWindow library is not in the directory that is current when the executable is built, fully qualify the reference in the PBR file. For example:

```
c:\myapp\sales.pbl(d_emplist)
```

What happens at execution time

When a resource such as a bitmap is referenced at execution time, 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.

Using the Application painter

Instead of using the Project painter to define and build your application, you can also use the Application painter to create an executable version of your application. In most cases, you will use the Project painter as described earlier in this chapter, but if you have a simple application that you don't expect to have to rebuild much, you can use the Application painter. You have to do more of the work manually, but that is not a problem for simple applications.

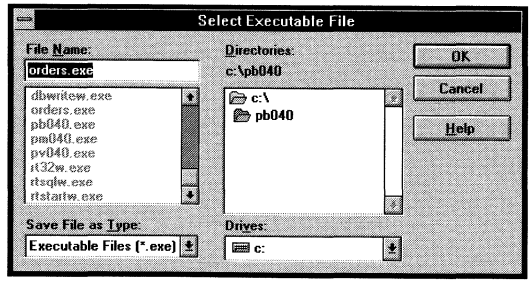
Creating the executable

❖ To create an executable version of your application using the Application painter:

1 Open the Application painter and select the application you want to make executable.

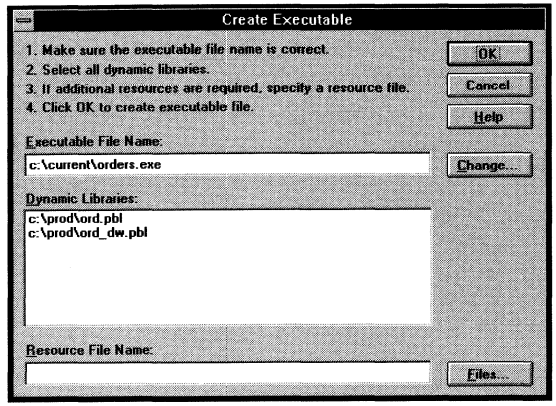
2 Click the Create Exe button in the PainterBar.
or
Select File➤Create Executable from the menu bar.

The Select Executable File dialog box displays.



3 Name the executable in the File Name box and click OK. The name you provide here is the name of the executable file that will be created.

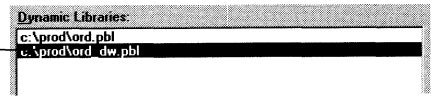
The Create Executable dialog box displays.



All the libraries in the application's library search path are listed in the Dynamic Libraries box. If you don't highlight a library in this list, PowerBuilder will copy referenced objects from the library into the executable file.

4 Select any libraries that you want linked dynamically *during execution* by highlighting them in the Dynamic Libraries box. You must later create a dynamic library from each of the specified libraries.

Objects in this library will not be copied into the executable file; they will come from a PowerBuilder dynamic library



5 If necessary, in the Resource File Name box specify a PowerBuilder resource file that lists the dynamically assigned resources that you want to include in the executable file.

For more about PowerBuilder resource files, see "Distributing resources" on page 795.

6 Click OK.

What happens

PowerBuilder creates an executable file in the specified directory. It searches through your application and copies the compiled versions of *referenced* objects from the libraries in the application's library search path into the executable file. It does not copy objects that are *not referenced* in the application to the executable file.

Also, it does not copy objects to the executable file from libraries you declared to be PowerBuilder dynamic libraries. These objects are linked to the application at execution time; they are not stored in the executable file.

For more about using dynamic libraries, see "Using dynamic libraries" on page 786.

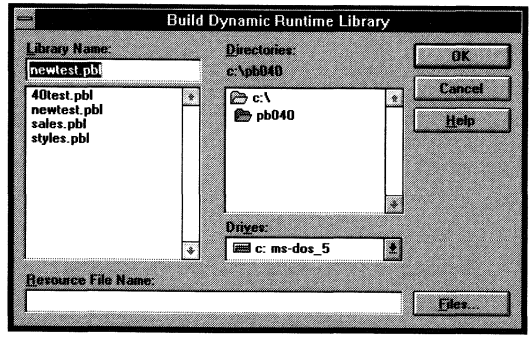
Creating dynamic libraries

When you create an executable using the Application painter, PowerBuilder does not automatically create the specified dynamic libraries. You do that yourself in the Library painter.

❖ To create a dynamic library:

1 In the Library painter, select Utilities➤Build Dynamic Library from the menu bar.

The Build Dynamic Runtime Library dialog box displays listing all libraries in the current directory.



2 Select the source library for the dynamic library from the list or enter the name of the library in the Library Name box.

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" below).

4 Click OK.

PowerBuilder closes the dialog box and creates a dynamic library that has the same name as the selected library with the file extension PBD.

PowerBuilder copies the compiled versions of *all* objects from the PowerBuilder source library (PBL file) to the PBD file (PowerBuilder can't know which objects are referenced in a particular application using this dynamic library, so it has to copy all objects).

Including additional resources

When building a dynamic 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 don't 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 dynamic library when it builds it.

For more about PowerBuilder resource files, see "Distributing resources" on page 795.

After you have defined the resource file, specify it in the Resource File Name box in the Build Dynamic Runtime Library dialog box to include the named resources in the dynamic library.

An example

Here's a simple example of using a dynamic library.

Let's say you want window w_emp to be in a dynamic library, so it can easily be shared among applications. Here's what you could do:

1 Create a library in the Library painter. You can name it anything. Let's say you name it EMP.PBL.

2 Move w_emp to the EMP.PBL library in the Library painter.

3 Update the application's library search path to include EMP.PBL.

4 When creating the executable in the Application painter, select EMP.PBL as a dynamic library by highlighting it in the Dynamic Libraries box.

   When PowerBuilder creates the executable, it does not copy the w_emp object into the executable. Instead, it adds EMP.PBD to its list of dynamic libraries—the libraries to search at execution time for objects.

5 Go to the Library painter and create a PBD file from EMP.PBL: select Build Dynamic Library from the Utilities menu, then select EMP.PBL as the library to be processed into a PBD file.

When you distribute the application, you must distribute the executable file and EMP.PBD. At execution time, when the user opens the window w_emp, the window's definition is linked from EMP.PBD.

PART SIX

Managing Your Environment

This part describes how to manage your PowerBuilder libraries including how to use check-out and check-in to coordinate your work with other developers. It also describes how to customize PowerBuilder to meet your needs.

CHAPTER 23

Managing Libraries

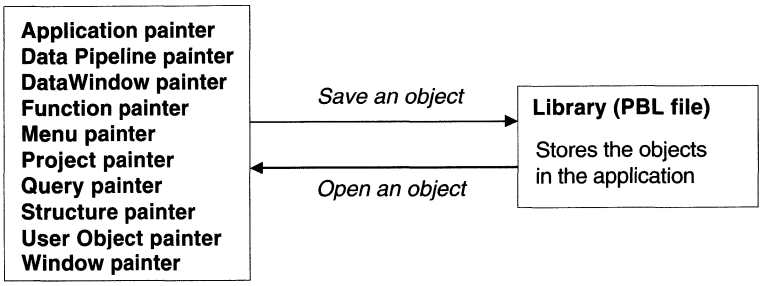
About this chapter

PowerBuilder stores all the objects you create in libraries. When you work with an application, you specify the libraries that will be used for it. This chapter describes how to work with your libraries.

Contents

Overview of 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.



Applications can use as many libraries as you want. Libraries can be on your own PC or on a server. When you create an application, you specify which libraries it uses. You can also change the library search path for an application anytime during development.

For more on assigning libraries to an application, see Chapter 2, "Working with Applications."

How the information is saved

Every object is saved in two parts in a library:

- Source form, which is a syntactic representation of the object, including the script code.
- Object form, which is a binary representation of the object, similar to an OBJ file in the C language. 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 an application's libraries will probably evolve over the development cycle. That is fine. PowerBuilder lets you reorganize your libraries easily anytime.

For small applications, you might use only one library. But for larger applications, you will want to split the application into different libraries.

About library size

There are no limits to how large libraries can be, but for performance and convenience, you should follow the guidelines below.

Try to keep your libraries smaller than about 800K. If your libraries are larger, performance can suffer because PowerBuilder has to search more in order to save or open an object.

Also, 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 doesn't affect performance. But if you have many objects in a library, you will find that listboxes that list library objects become unmanageable and that the Library painter becomes more difficult to use.

On the other hand, you don't want to have to manage a large number of libraries with only a few objects. That makes the library search path too long and can slow performance by forcing PowerBuilder to look through many libraries to find an object. So you should 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 want to put all objects of one type in their own library. Or you might want to divide your application into subsystems and place each subsystem in its own library.

A recommended organization

If you are working with other developers on a large application, here is a setup that works well.

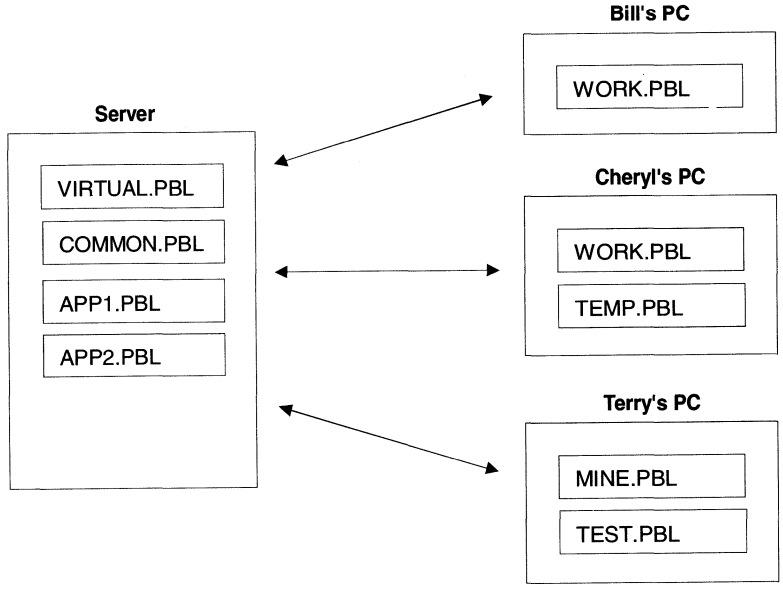
Put libraries containing objects that are shared by developers on a server machine on the network. That way, all developers have direct access to them. Put objects that only you are working on in a library on your PC.

Here is an example. The following libraries would be publicly available on a server:

- VIRTUAL.PBL—This library contains all ancestor objects used in the application. For example, if all the windows in your application inherit from w_master, place w_master in this library.
- COMMON.PBL—This library contains all objects that are used across applications, such as user objects and functions.
- APP1.PBL—This library contains objects specific to application 1.

- APP2.PBL—This library contains objects specific to application 2.

Also, each developer would have one or more private libraries where they would keep the objects they are working on.



Sharing objects with others

PowerBuilder provides check-out/check-in facilities that let you check an object out of one library, such as APP1.PBL, and store a working copy in another library, such as your private library. While you have the object checked out, no one else can modify it. When you finish updating an object, you can check the object back in to the public library.

Ordering the application's library search path

If you use the scenario described above with public and private libraries, you should place the private libraries first in the library search path. Then, when you check an object out of a public library and place it in your private library, PowerBuilder will find the private one first when executing the application. When you check the object back in to the public library, it is removed from the private library and PowerBuilder finds the updated public version when executing the application.

For more about check-out/check-in, see "Using check-out and check-in" on page 825.

Specifying the library search path

You specify an application's library search path in the Application painter.

For more information, see Chapter 2, "Working with Applications."

Working with libraries

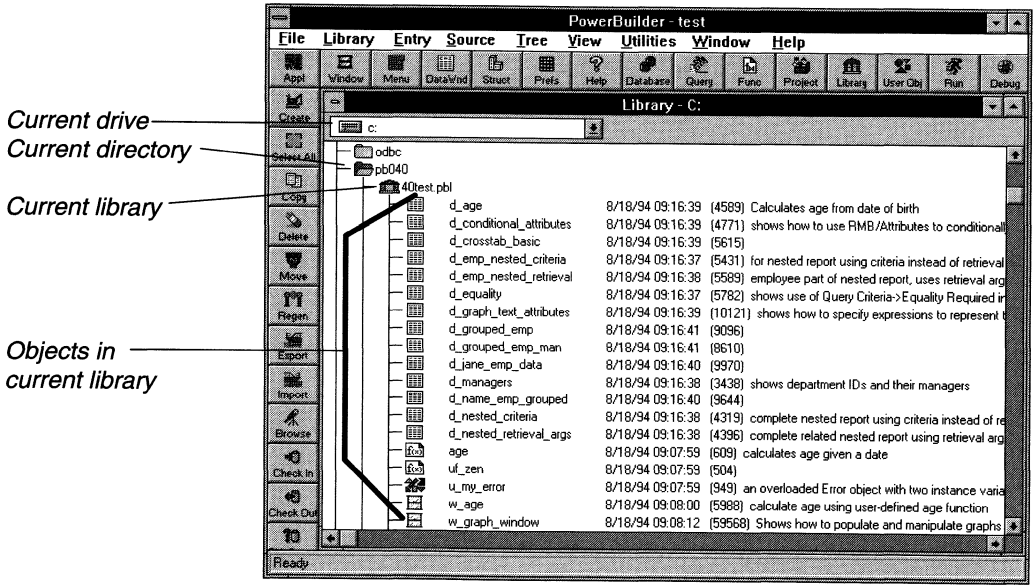
You work with libraries in the Library painter.

❖ To open the Library painter:

- Click the Library painter button in the PowerBar or PowerPanel.

  The Library painter displays.

When you open the Library painter, it lists all directories on the current drive. It expands the current directory and expands the current library (the most recently used library) to show its entries.



Viewing the tree

You can expand (display the contents of) libraries and directories. You can also collapse the display of libraries and directories.

❖ To expand a library or directory:

- If the library or directory is currently collapsed, double-click the library or directory.

  PowerBuilder displays all objects stored in the selected library or the files and subdirectories in the selected directory.

Each entry in a library has an icon that identifies the painter in which the entry was created:

 Application

 Data Pipeline

 DataWindow

 Function

 Menu

 Project

 Query

 Structure

 User Object

 Window

❖ To collapse a library or directory:

- If the library or directory is currently expanded, double-click the library or directory.

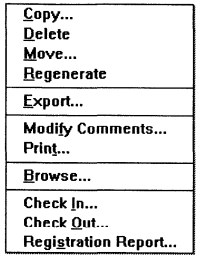
  PowerBuilder hides all objects stored in the selected library or the files and subdirectories in the selected directory.

Using the popup menu

Like the other painters, the Library painter has a popup menu that provides items that apply to the selected object in the workspace.

❖ To use the popup menu:

1 Position the mouse pointer on an object listed in the workspace.



2 Click the right mouse button.

The popup menu displays. Which items display on the menu depend on the type of object you have selected.

3 Select the item you want from the menu.

Limiting the display of library entries

You can change what is shown in expanded libraries. You can specify:

- Which objects are displayed
- What information is shown for the displayed objects

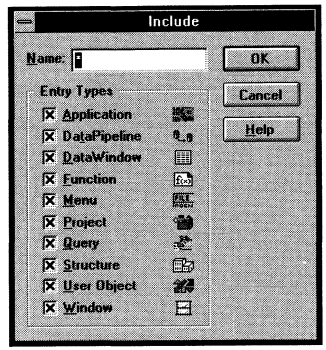
Specifying which objects are shown

Initially, the Library painter displays all objects in expanded libraries. You can have the 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 View➤Include from the menu bar.

The Include dialog box displays.



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: ? represents one character, * represents any string of characters. The default is all entries of the selected types.
- To limit the display to specific entry types, deselect the entry types that you do not want to display. The default is all entries.

3 Click OK.

The Include dialog box closes.

4 Expand libraries to display the entries that meet the criteria.

Specifying which information is shown for the displayed objects

Initially, PowerBuilder displays the name, modification date, size (of the compiled object), check-out status, and comments for displayed entries.

❖ To specify which information is displayed:

- Select Comments, Modification Date, Check Out Status, or Size from the View menu to toggle the display of comments, dates, check-out status, and sizes.

Settings are remembered

PowerBuilder records the following in the Library section of the PB.INI file:

- Which object types are displayed
- Which information is displayed

So the next time you open the Library painter, you get the same type of display.

You can modify the settings in the Library painter, in the Preferences painter, or by editing the PB.INI file.

Selecting library entries

You can select one or more library entries to act on. You can select entries in different libraries at the same time.

❖ **To select multiple entries:**

- To select noncontiguous entries, press CTRL and click each entry you want to select.
- To select contiguous entries, click the first entry you want to select, press SHIFT, then click the last entry you want to select.

  PowerBuilder highlights selected entries.

Tips

To select all entries in a library, select the library, then click the Select All button.

To select most but not all of a block of entries, select them all, then press CTRL and click the entries you do not want to select.

Using 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 descendant objects, or identify a PowerBuilder library.

You can associate comments with an object when you first save it in a painter. You can use the Library painter to add or modify comments for a saved object.

Modifying comments of saved objects
The Library painter is the only place where you can modify comments for a saved object.

Updating library comments

❖ To update comments for an existing PowerBuilder library:

1 Select the library in the workspace.

2 Select Modify Comments from the library's popup menu.
or
Select Library➤Modify Comments from the menu bar.

The Modify Library Comments dialog box displays the name of the library and any comments associated with it.

3 Add or modify the comments.

4 Click OK.

You return to the Library painter workspace.

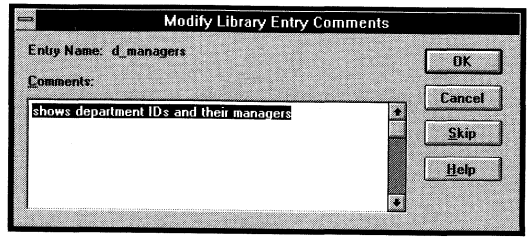
Updating comments for an entry

❖ To update comments for a library entry:

1 Select the entries whose comments you want to update.

2 Select Entry➢Modify Comments from the menu bar.

The Modify Library Entry Comments dialog box displays the comments for the first entry you selected.



3 Add or modify the comments. To display the comments for the next selected entry without changing the comments for the current entry, click Skip.

4 Click OK to save the comments.

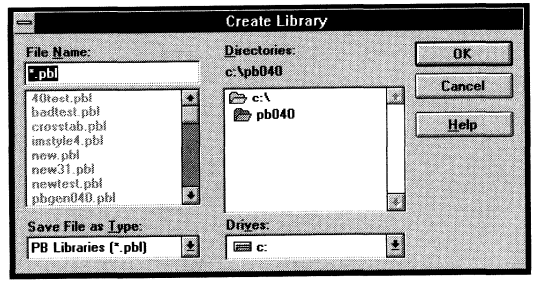
If you have selected more than one entry, comments for the next entry are displayed. Continue updating comments until you have gone through all selected entries.

Creating and deleting libraries

❖ To create a library:

1 Click the Create button.
or
Select Library➤Create from the menu bar.

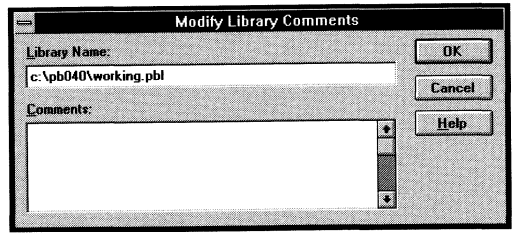
The Create Library dialog box displays the name of the current directory and lists the libraries and subdirectories in the current directory.



2 Enter the name of the library you are creating and specify the directory in which you want to store it. You are naming a file so must follow the operating system rules on your platform. Make sure the file extension is PBL.

3 Click OK.

The Modify Library Comments dialog box displays.



4 Enter any comments you want to associate with the library. It is good practice to add comments to describe the purpose of a library.

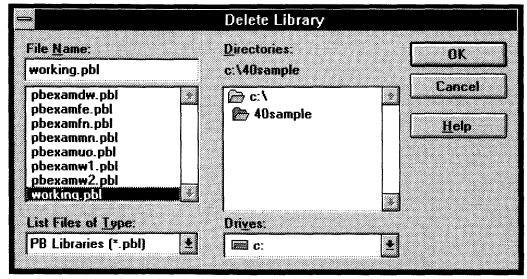
5 Click OK.

PowerBuilder creates the library.

❖ To delete a library:

1 Select Library➤Delete from the menu bar.

The Delete Library dialog box displays listing the current library and shows the current directory.



2 Specify the library you want to delete.

> **Restriction**
> You cannot delete a library that is in the current application's library search path.

3 Click OK.

You are asked to confirm the deletion.

> **Being asked for confirmation**
> By default, PowerBuilder asks you to confirm each deletion. If you don't want to have to confirm deletions, deselect Confirm on Delete from the View menu to remove its checkmark. From now on, you won't be asked to confirm deletions.
>
> PowerBuilder records this preference as the DeletePrompt variable in the Library section of the PB.INI file.

4 Click Yes to delete the library.

The library and all its entries are deleted. You cannot get them back.

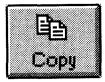
> **Creating and deleting libraries during execution**
> You can use the LibraryCreate and LibraryDelete functions in scripts to create and delete libraries.
>
> For information about these functions, see the *Function Reference*.

Copying, moving, and deleting entries

As your application needs change, you will want to rearrange libraries. Perhaps you are dividing your application into different libraries. To do that, you will want to copy and move entries between libraries or delete entries that you no longer need.

❖ To copy entries to a different library:

1 Select the entries you want to copy to another library.



2 Click the Copy button.
or
Select Entry➤Copy from the menu bar.

The Copy Library Entries dialog box displays.

3 Select the library to which you want to copy the entries.

4 Click OK.

PowerBuilder copies the entries. If a library entry with the same name already exists, PowerBuilder silently replaces it with the copied entry.

❖ To move entries to a different library:

1 Select the entries you want to move to another library.

2 Click the Move button.
or
Select Entry➤Move from the menu bar.

The Move Library Entries dialog box displays.

3 Select the library to which you want to move the entries.

4 Click OK.

PowerBuilder moves the entries. PowerBuilder deletes the entry from the source library. If a library entry with the same name already exists in the destination library, PowerBuilder silently replaces it with the moved entry.

❖ To delete entries:

1 Select the entries you don't need anymore.

2 Click the Delete button.
or
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 don't want to have to confirm deletions, deselect Confirm On Delete from the View menu to remove its checkmark. From now on you won't be asked to confirm deletions.
>
> PowerBuilder records this preference as the DeletePrompt variable in the Library section of the PB.INI file.

3 Click Yes to delete the entry. Click No to skip the current entry and go on to the next one selected.

Browsing library entries

You can browse library entries to locate where a specified text string is used in your application. For example, you can search for:

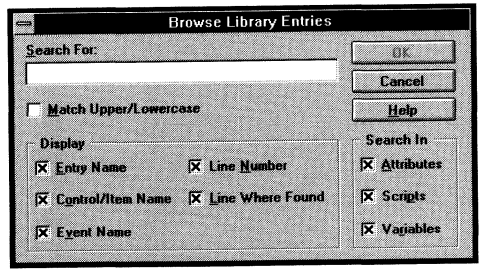
- All scripts that use the SetTransObject function
- All windows that contain the CommandButton cb_exit (all controls contained in a window are listed in the window definition's source form in the library so can be searched for as text)
- All DataWindow objects accessing the Employee table in the database

❖ To browse library entries for a string:

1 Select the entries you want to browse. You can select entries across libraries by expanding the libraries and using SHIFT-click and/or CTRL-click to select multiple entries.

2 Click the Browse button.
or
Select Entry➤Browse from the menu bar.
or
Press CTRL+B.

The Browse Library Entries dialog box displays.



3 Enter the string you want to locate (the search string). The string can be all or part of a word or phrase used in an attribute, a script, or a variable. You cannot use wildcards in the search string.

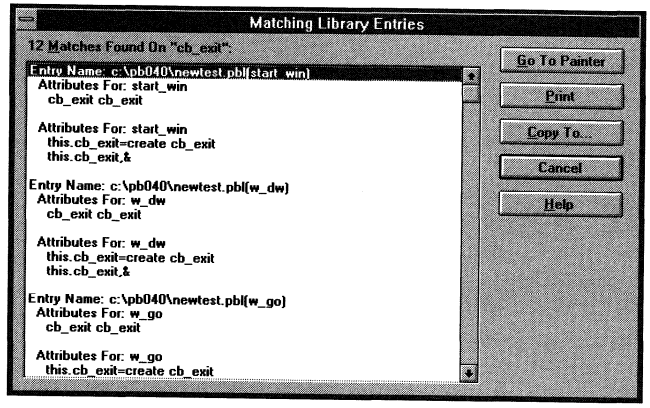
4 Select the information you want to display in the results of the browse.

5 Select the parts of the object that you want PowerBuilder to inspect: attributes, scripts, and/or variables.

6 Click OK.

PowerBuilder browses the libraries for matching entries. When the browse is complete, PowerBuilder displays the matching entries in the Matching Library Entries window.

The following dialog box displays the results of a search for the string cb_exit (in this case, all references to a CommandButton named cb_exit).



What you can do

From the Matching Library Entries dialog box, you can:

- Jump to the painter in which an entry was created; this is described next
- Print the contents of the window
- Copy the browse results to a text file

Jumping to a painter

You can jump from the Library painter directly to the painter where a specific entry was created.

❖ To jump to a painter:

- From the directory tree in the painter's workspace, double-click an entry.
- From the Match Library Entries dialog box that resulted from your browsing entries, double-click the entry.
  or
  Select the entry, then click the Go To Painter button.

  For more on browsing entries, see page 820.

  PowerBuilder opens the object in its painter.

You can view the object and make changes as needed in the painter. When you close the painter, PowerBuilder returns you to the point in the Library painter from which you initiated the jump.

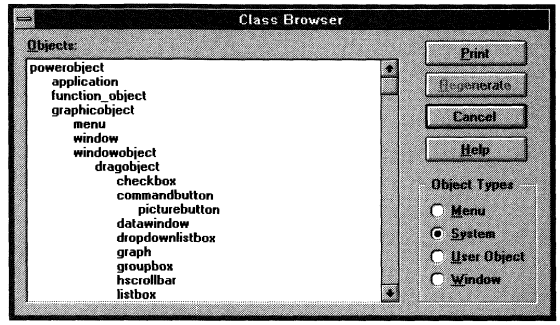
Browsing the class hierarchy

You have probably used inheritance to define hierarchies of windows, menus, and user objects. You can examine these hierarchies in the Library painter. In object-oriented terms, these are called class hierarchies: each PowerBuilder object defines a class.

❖ To browse the class hierarchies in your application:

1 Select Utilities➤Browse Class Hierarchy from the menu bar.

The Class browser displays. You can display several kinds of hierarchies. The default object type is System, which shows the hierarchy of the built-in objects.



2 Choose the type of hierarchy to display in the browser by clicking the object type in the Object Types box.

PowerBuilder displays the hierarchy for the selected object type in the current application. Descendant objects are shown indented under their ancestors.

Another way to browse hierarchies

You can also access the Class browser when you are defining an inherited window, menu, or user object, as follows:

1 Click the Inherit button in the Select Window/Menu/ User Object window.

The Inherit From dialog box displays.

2 Click the Browse button.

Regenerating objects
This dialog box provides a convenient way to regenerate objects and their descendants.

For more information, see "Regenerating library entries" on page 833.

Using check-out and check-in

If you are working with other developers on a large application, you will want to prevent multiple developers from modifying a library entry at the same time. To control access to library entries, PowerBuilder provides check-out and check-in facilities. Using them, you can check an object out of a public library and store a working copy in a private library. While the object is checked out, no one can modify it in the public library. When you finish updating an object, you can check the object back in to the public library.

For more about setting up libraries to support this type of development environment, see "Overview of libraries" on page 806 and the *Building Applications* manual.

About PowerBuilder and version control systems

PowerBuilder provides interfaces to external version control systems so that you can manage your PowerBuilder objects through an external system while developing PowerBuilder applications. Most of the items on the Source menu in the Library painter apply only if you are using a version control system.

For more about using PowerBuilder with external version control systems, see *Version Control Interfaces*.

This section describes check-out and check-in, facilities that are available even if you are not using a version control system to manage PowerBuilder objects.

Overview of the process

This section describes how the check-out and check-in facilities work.

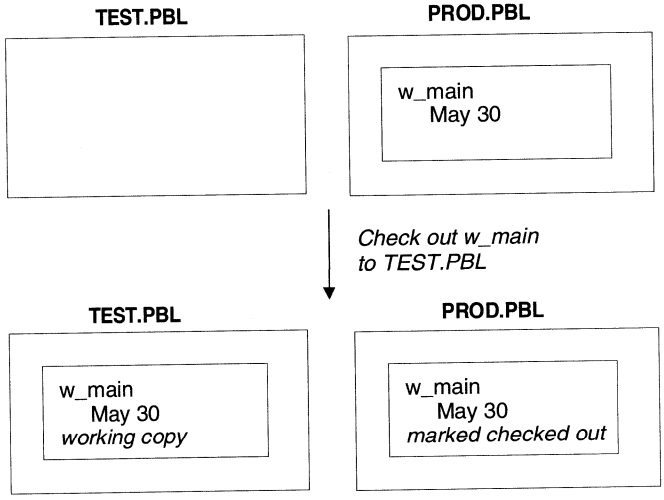
How check-out works

When you want to work on an object and prevent others from making changes to the object, you check it out.

When you check out a library entry, PowerBuilder:

- Makes a working copy of the entry in a specified library (for example, a private test library)
- Sets the status of the original entry to checked out

The following figure shows what happens if w_main is checked out to a test library.



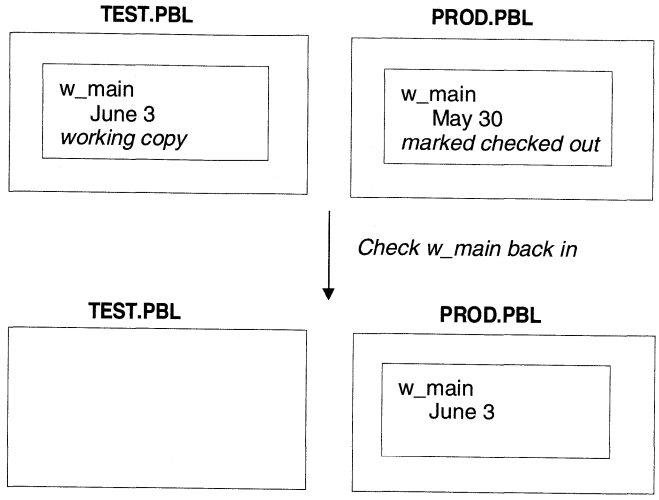
As long as the status of a library entry is checked out, changes can be made only to the working copy. If you or another user tries to open the original, PowerBuilder displays a message box warning that the entry is checked out and can be opened but cannot be saved.

How check-in works

When you finish working with an entry that you checked out, you check the entry back in. PowerBuilder:

- Replaces the entry in the library from which you checked it out with the working copy
- Deletes the working copy from the library to which you checked it out

The following figure shows what happens when w_main is checked back in.



If you don't 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 check-out status of the entry in the original library and deleting the working copy.

Checking entries out

❖ To check out entries:

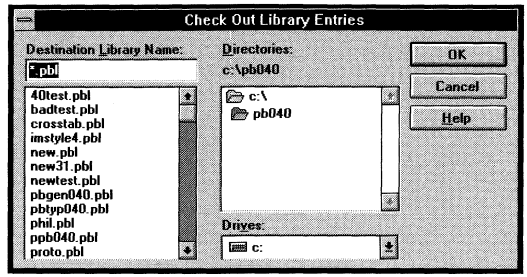
1 Select the entries you want to check out. You can check out more than one entry at a time.

2 Click the Check Out button.
or
Select Source➤Check Out from the menu bar.

3 If this is the first time you have checked out an entry, the User ID dialog box displays. Enter your user ID and click OK.

PowerBuilder saves your ID as the variable UserID in the Library section of PB.INI and will not prompt you for an ID again. You can change the ID in Preferences or in the PB.INI file. PowerBuilder uses the ID to identify who checked out objects.

The Check Out Library Entries dialog box displays the current directory and lists the libraries in that directory.



4 Enter the name of the library in which you want to save the working copy in the Destination Library Name box or select the library from the list of libraries. You must specify a library in the current application's library search path, or you won't be able to save the working copy later.

If you selected an entry that another user has checked out, PowerBuilder displays a message box and asks if you want to continue. Click Yes to process any remaining entries.

5 Click OK.

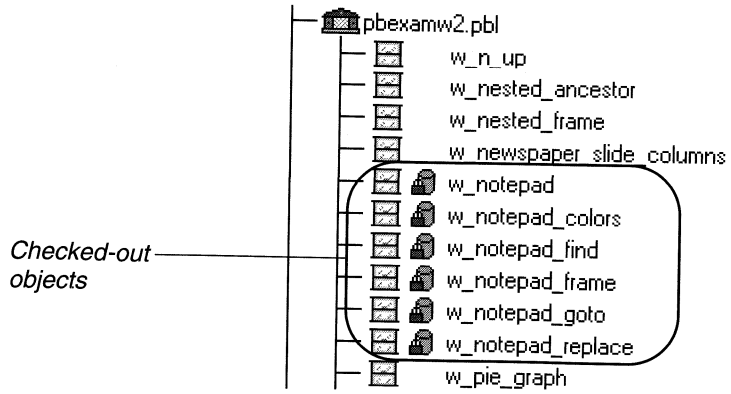
PowerBuilder makes a working copy of each selected entry and stores it in the destination library you specified. The Check Out Library Entries dialog box closes and the Library painter workspace displays.

Checked-out items shown in workspace

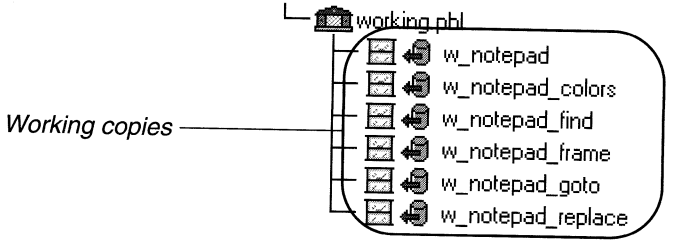
The Library painter workspace shows with icons which objects have been checked out and which objects are working copies.

Icon in workspace	Meaning
	Object is checked out and therefore locked. No one can modify this object.
	Object is a working copy of a checked-out object. This object can be modified.

For example, the following shows that six windows have been checked out of PBEXAMW2.PBL:



The working copies are in WORKING.PBL:



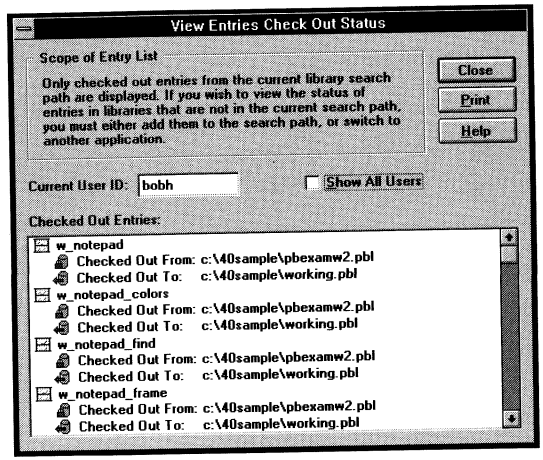
Viewing the checked-out entries

You can display a list of the entries in the current application that are checked out.

❖ To display the checked-out entries:

1 Click the Check Out Status button.
or
Select Source➤View Check Out Status from the menu bar.

The View Entries Check Out Status dialog box displays the name of each entry in the current application that you have checked out and the name of the library it is checked out to.



2 Select the Show All Users checkbox to display all the entries that are checked out from the current application.

3 Click Print to print the list of checked-out entries.

4 Click Close.

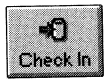
The View Entries Check Out Status dialog box closes and the Library painter workspace displays.

Checking entries back in

When you finish working with an entry that you checked out, you check the entry back in.

❖ To check in entries:

1 Select the entries (the working copies) you want to check in.



2 Click the Check In button.
or
Select Source➤Check In from the menu bar.

PowerBuilder replaces the entries in the original library with the working copies, deletes the working copies, then displays the Library Painter workspace.

If you selected an entry that another user has checked out, PowerBuilder displays a message box and asks if you want to continue. Click Yes to process any remaining entries.

Clearing the check-out status of entries

Sometimes you will want to remove (clear) the check-out status of an entry without checking the entry back in. Perhaps you have decided not to update the object.

❖ To clear the check-out status of entries:

1 Select the entries whose check-out status you want to remove.

2 Select Source➤Clear Check Out Status from the menu bar.

A message box asks whether you want to clear the check-out status of the entry.

3 Click Yes to clear the status.

You are asked whether you want to delete the working copy.

4 Click Yes to delete the working copy. Click No to retain it.

Optimizing libraries

As you add, modify, and delete objects, libraries can get fragmented: they can contain gaps of unused space, and they can end up storing objects in noncontiguous areas of the hard disk. This can slow performance.

So you should optimize your libraries regularly. Optimizing removes gaps in libraries and unfragments the storage of objects. Optimizing only affects layout on disk; it doesn't 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 Select Library➤Optimize from the menu bar.

The Optimize Library dialog box displays the name of the current directory and a list of libraries and subdirectories in that directory. The current directory is the directory containing the last library you used in PowerBuilder.

2 Select the library you want to optimize.

3 If you do not want to save a backup copy of the library, deselect the Save Original Library as .BAK File checkbox. If you do deselect this option, the new setting will remain in effect until you change it.

4 Click OK.

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. If the Save Original Library as .BAK File checkbox was selected, PowerBuilder saves the unoptimized version as a BAK file in the same directory.

Regenerating library entries

Occasionally you might need to regenerate library entries. For example:

- When you modify an ancestor object, you can regenerate descendants so they pick up the revisions to their ancestor.
- When you upgrade to a new version of PowerBuilder, you need to migrate (regenerate) your applications.

When you regenerate an entry, PowerBuilder recompiles the source form stored in the library and replaces the existing compiled form with the recompiled form.

❖ To regenerate library entries:

1 Select the entries you want to regenerate.

2 Click the Regen button.
or
Select Entry➤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 modification date and size are updated.

Regenerating descendants

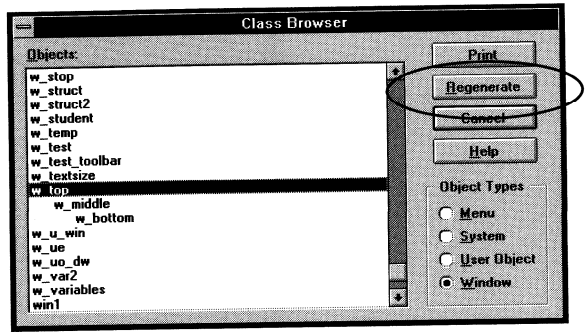
You can use the Class browser to easily regenerate all descendants of a changed ancestor object.

❖ To regenerate descendants:

1 Open the Class browser by selecting Utilities➤Browse Class Hierarchy from the menu bar.

2 Select the type of object you want to work with in the Object Types box. For example, if you want to regenerate all descendants of window w_top, click the Window button.

3 Select the ancestor object.



4 Click the Regenerate button.

PowerBuilder regenerates all descendants of the selected ancestor.

For more about the Class browser, see "Browsing the class hierarchy" on page 823.

Exporting and importing entries

You can export object definitions to ASCII 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 might 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.

> ***Caution***
> *The primary use of the Export feature is to export source code, not to modify the source. Modifying source in an ASCII text file is not recommended.*

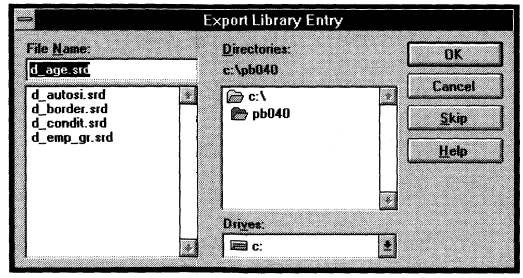
Later you can import the files back into PowerBuilder for storage in a library.

❖ To export entries to text files:

1 Select the Library entries you want to export.

2 Click the Export button.
or
Select Entry➤Export from the menu bar.

The Export Library Entry dialog box displays 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 directory containing the last library you used.



If necessary, PowerBuilder truncates the object name to eight characters (because you are creating a file). PowerBuilder appends the file extension SR*x*, where *x* represents the object type.

3 Specify the filename and directory for the export file. Do not change the file extension from the one that PowerBuilder appended.

4 Click OK.

PowerBuilder converts the entry to ASCII file format, 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.

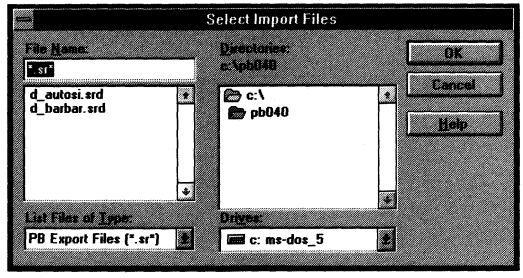
5 Repeat steps 3 and 4 until you have processed all the selected entries.

You can't see export files in the Library painter
Since export files are ASCII text files, the Library painter does not show them; it shows only libraries and directories.

❖ To import text files to library entries:

1 Click the Import button.
or
Select Entry➤Import from the menu bar.

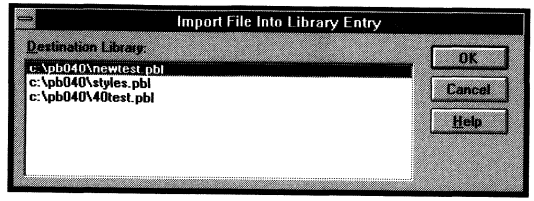
The Select Import Files dialog box displays with the name of the current directory and a list of files with the extension SR* in that directory. The current directory is the directory containing the PowerBuilder library you used last.



2 Select the files you want to import. Use SHIFT+click to select multiple files.

3 Click OK.

The Import File Into Library Entry dialog box displays listing the libraries in the application's library search path.



4 Select the library you want to import the text files to.

5 Click Import.

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 silently replaces it with the imported entry.

Creating dynamic libraries

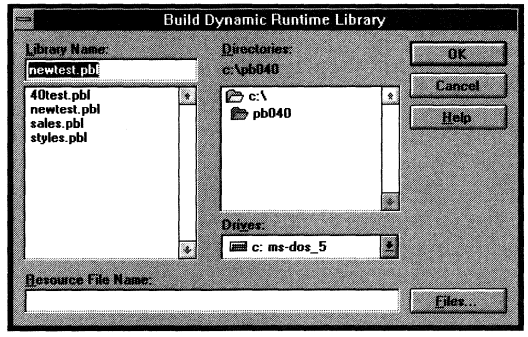
If you want your distributed application to use dynamic libraries, you can create them in the Library painter.

For information about using dynamic libraries, see Chapter 22, "Creating an Executable." That chapter also describes the Project painter, which you can use to automatically create dynamic libraries.

❖ To create a dynamic library:

1 Select Utilities➤Build Dynamic Library from the menu bar.

The Build Dynamic Runtime Library dialog box displays listing all libraries in the current directory.



2 Select the source library for the dynamic library from the list or enter the name of the library in the Library Name box.

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" below).

4 Click OK.

PowerBuilder closes the dialog box and creates a dynamic library that has the same name as the selected library with the file extension PBD.

Including additional resources

When building a dynamic 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 don't 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 dynamic library when it builds it.

For more on resource files, see Chapter 22, "Creating an Executable."

After you have defined the resource file, specify it in the Resource File Name box to include the named resources in the dynamic library.

Creating reports on library contents

You can generate three types of reports from the Library painter:

- The browse results report
- Library entry reports
- The library directory report

The browse results report contains the matching-entries information that PowerBuilder displays after it completes a browse; these reports are described in "Browsing library entries" on page 820. The other two types of reports are described below.

Creating library entry reports

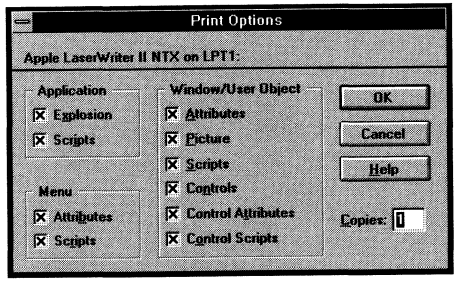
These reports provide information about selected entries in the current application. You can use these reports to get printed documentation about the objects you have created in your application.

❖ To create library entry reports:

1 Select the library entries you want information about.

2 Select Entry➤Print from the menu bar.

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 attributes for selected windows to appear in the report, make sure the Attributes box is checked in the Window/User Object group in the Print Options dialog box.

> **The settings are saved**
> PowerBuilder records these settings in the Library section of the PB.INI 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. It lists 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 Library➤Print Directory from the menu bar.

The Print Library Directory dialog box displays.

2 Select the library you want the report for.

3 Click OK.

PowerBuilder sends the library directory report to the printer specified in Printer Setup in the File menu.

CHAPTER 24

Customizing PowerBuilder

About this chapter

This chapter describes how to customize PowerBuilder to suit your needs.

Contents

Overview

You can use the Preferences painter to customize PowerBuilder to meet your needs by changing the values of variables that PowerBuilder uses to determine aspects of its behavior. The information is stored in the PB.INI file in your PowerBuilder directory. (You can also manually edit the PB.INI file, but this is seldom necessary.)

This chapter describes how to use the Preferences painter to specify the values of variables.

How the variables are organized

The variables are divided into sections. There is a section that applies to PowerBuilder in general and sections that apply to each of the major painters, as follows:

- PowerBuilder
- Application
- Database
- DataWindow
- Debug
- Library
- Menu
- Window

Specifying your preferences

Many of the variables that appear in the Preferences painter are normally set directly in the painter that they apply to. For example, one of the variables in the Application section of the Preferences painter names the current application. You use the Application painter to change the current application; you don't use the Preferences painter. The choice you make in the Application painter is recorded in the PB.INI file and displays when you later open the Preferences painter.

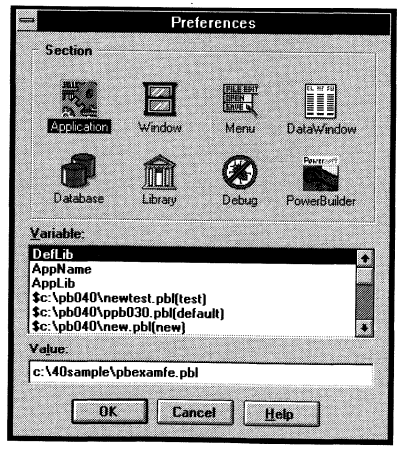
Other variables, such as the default prefixes for controls you place in windows and user objects, are specified only in the Preferences painter.

This section describes how to use the Preferences painter—but keep in mind that PowerBuilder automatically sets many of the variables as you work in the other painters.

❖ To use the Preferences painter:

1 Open the Preferences painter by clicking the Preferences button in the PowerBar or PowerPanel.

The Preferences painter displays. Each section of variables is represented by an icon. Variables for the selected section display in the Variable box. The current value of the selected variable displays in the Value box.



2 Click the icon that represents the section of variables you want to set. For example, to change settings related to the Window painter, click the Window icon.

The variables for the selected section display in the Variable box.

3 Select the variable whose value you want to change.

The current value displays in the Value box.

4 Type a new value for the variable.

5 Repeat steps 2 through 4 to set any other variables you want.

6 Click OK.

The Preferences painter closes.

Using the PB.INI file

PowerBuilder stores your preferences in the PB.INI file in the PowerBuilder directory. The file is divided into the same sections as shown in the Preferences painter (there are additional sections of variables too, which contain variables that are maintained automatically by PowerBuilder). Each time you start PowerBuilder, PowerBuilder reads the values from PB.INI to set your environment.

Editing the PB.INI file

Normally, you don't need to edit the PB.INI file; you can specify all your preferences from one of the painters. But it might happen that a variable doesn't appear by default in the Preferences painter. So if you don't see a variable whose value you want to change in the Preferences painter, use a text editor to add the variable to the appropriate section of PB.INI. The variable will then appear in the Preferences painter.

Using a master PB.INI file

If you want to set up several users with the same environment, you can copy a master PB.INI to each of their PowerBuilder directories, then make whatever individual changes you want.

Listing the preferences

This manual describes many of the preference variables in the context in which they are used. For example, the section on the PowerScript painter describes how to set preference variables to change the font used in the painter.

For a complete listing of the preference variables, see the online Help topic "Preferences."

Appendix

The appendix describes the Powersoft repository.

APPENDIX

The Powersoft Repository

About this chapter

This chapter describes the Powersoft repository.

Contents

About the repository

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 Powersoft **repository**. The repository tables contain all the information related to the extended attributes for the tables and columns in the database. These extended attributes are used in DataWindow objects.

There are five Powersoft 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

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 repository information used in your database by building a DataWindow whose data source is the repository tables.

> ***Caution***
> *You should not change the values in the repository tables. PowerBuilder maintains this information automatically whenever you change information for a table or column in the Database painter.*

This appendix describes each column in each of the repository tables.

The PBCatTbl table

Column	Column name	Description
1	pbt_tnam	Table name
2	pbt_tid	SQL Server Object ID of table (used for SQL Server 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)
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)

Column	Column name	Description
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=Don't Care, 16=Roman, 32=Swiss, 48=Modern, 64=Script, 80=Decorative)

The PBCatCol table

Column	Column name	Description
1	pbc_tnam	Table name
2	pbc_tid	SQL Server Object ID of table (used for SQL Server only)
3	pbc_ownr	Table owner
4	pbc_cnam	Column name
5	pbc_cid	SQL Server Column ID (used for SQL Server 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)
17	pbc_init	Initial value
18	pbc_cmnt	Column comments
19	pbc_edit	Edit style name
20	pbc_tag	(Reserved)

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

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

The PBCatEdt table

Column	Column name	Description
1	pbe_name	Edit style name
2	pbe_edit	Format string (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 (type dependent, see below)
7	pbe_work	Extra field (type dependent, see below)

Available edit style types

The following edit style types are available.

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)

Here is a sample row in the PBCatEdt table 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	

where

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)
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)

Here is a sample row in the PBCatEdt table 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	

where

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
	About the display and data values Display and data values are repeated in pairs for each radio button defined in the edit style
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)

Here is a sample row in the PBCatEdt table 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	

where

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

About the display and data values
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: Lowercasee (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)

Here is a sample row in the PBCatEdt table 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%*

where

Value	Meaning
DataWin	Name of DataWindow to use
DataCol	Data column from DataWindow

Value	Meaning
DisplayCol	Display column from DataWindow
Limit	Character representation (in decimal) of Limit value
Key	One-character accelerator key
Width%	Width of the drop down 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)

Here is a sample row in the PBCatEdt table 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	

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Display2*	89	1	5	0	
MyEdit	*Data2*	89	1	6	0	

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 15, "Displaying and Validating Data."

where

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.

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: 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)

Here is a sample row in the PBCatEdt table for an Edit Mask 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	

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 15, "Displaying and Validating Data."

where

Value	Meaning
Format	Display format mask
DtFcKy	Concatenated string with one-character data-type code, one-character focus-rectangle code (0 or 1), and one-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"
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

Value	Meaning
Data2	Data value for second entry in code table **About the display and data values** 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

Index

Symbols

2

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

X

Y

Z