



Open Tools from Sybase, Inc.

PowerBuilder

Getting Started

Windows

Version 6

Power Builder®

AB0884

October 1997

Copyright © 1991-1997 Sybase, Inc. and its subsidiaries.

All rights reserved.

Printed in the United States of America.

Information in this manual may change without notice and does not represent a commitment on the part of Sybase, Inc. and its subsidiaries.

The software described in this manual is provided by Powersoft Corporation under a Powersoft License agreement. The software may be used only in accordance with the terms of the agreement.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc. and its subsidiaries.

Sybase, Inc. and its subsidiaries claim copyright in this program and documentation as an unpublished work, revisions of which were first licensed on the date indicated in the foregoing notice. Claim of copyright does not imply waiver of other rights of Sybase, Inc. and its subsidiaries.

ClearConnect, Column Design, ComponentPack, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, S-Designor, SQL SMART, and Sybase are registered trademarks of Sybase, Inc. and its subsidiaries. Adaptive Component Architecture, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Warehouse, AppModeler, DataArchitect, DataExpress, Data Pipeline, DataWindow, dbQueue, ImpactNow, InstaHelp, Jaguar CTS, jConnect for JDBC, MetaWorks, NetImpact, Optima++, Power++, PowerAMC, PowerBuilder Foundation Class Library, Power J, PowerScript, PowerSite, Powersoft Portfolio, Powersoft Professional, PowerTips, ProcessAnalyst, Runtime Kit for Unicode, SQL Anywhere, The Model For Client/Server Solutions, The Future Is Wide Open, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, Viewer, WarehouseArchitect, Watcom, Watcom SQL Server, Web.PB, and Web.SQL are trademarks of Sybase, Inc. or its subsidiaries. Certified PowerBuilder Developer and CPD are service marks of Sybase, Inc. or its subsidiaries. DataWindow is a patented proprietary technology of Sybase, Inc. or its subsidiaries.

AccuFonts is a trademark of AccuWare Business Solutions Ltd.

All other trademarks are the property of their respective owners.

Contents

About This Book	vii
------------------------------	------------

PART 1 WELCOME TO POWERBUILDER

1 Introduction to PowerBuilder	3
About PowerBuilder for Windows	4
What PowerBuilder is	5
PowerBuilder objects	8
The PowerBuilder environment	16

PART 2 THE POWERBUILDER TUTORIAL

2 About the PowerBuilder Tutorial	23
What you will do	24
How you will proceed	26
How long it will take	26
What you will learn	26
Setting up	28
 3 Starting PowerBuilder	 29
Start PowerBuilder	30
Create and save the Application object	31
Control the toolbars	35
Use online Help	39
Display a Help topic by navigating from the TOC	40
Use the Index facility to search for a Help topic	41
Display context-sensitive Help for a dialog box	42
Use the Powersoft Online Books	44

4	Modifying the Application Object.....	45
	Specify an icon for the application	46
	Add a library to the search path	48
	Run the application	50
5	Building a Logon Window.....	53
	Create a new window	54
	Add controls to the window	57
	Add a Picture control	58
	Add StaticText controls	60
	Specify properties of the StaticText controls	61
	Add SingleLineEdit controls	63
	Specify properties of the SingleLineEdit controls	64
	Add CommandButton controls	65
	Specify properties of the CommandButton controls	66
	Change the tab order on the window	67
	Save the window	68
	Preview the window	69
	Write the script to open the window	70
	Modify a script in the PowerScript painter	71
	Compile the script.....	74
6	Connecting to the Database	75
	Look at the Powersoft Demo Database	76
	Look at the table definitions in the Powersoft Demo DB	77
	Look at the database profile for the Powersoft Demo DB	79
	Establish the execution time connection	82
	Modify the script for the application Open event	83
	Add a window function	86
	Write scripts for the buttons.....	89
	Run the application	91
7	Setting Up the Menus	93
	Open the menu for the MDI frame	94
	Add items to the File dropdown menu.....	95
	Preview the menu	97
	Create a new menu.....	98
	Add items to the new menu.....	99
	Add toolbar buttons for the new menu items.....	101
	Save the new menu	103
	Preview the new menu.....	104

8	Building an Ancestor Window	105
	Create a new window	106
	Add DataWindow controls	107
	Add a DataWindow control for the master DataWindow	108
	Add a DataWindow control for the detail DataWindow	110
	View the scripts inherited from the user object	111
	Add user events	113
	Add scripts for the user events	116
	Add a script to retrieve data for the master DataWindow	119
	Add a script to retrieve data for the detail DataWindow	120
	Attach a menu to the window	122
	Save the window	123
	Add menu scripts to trigger the user events	124
9	Building Two Descendent Windows	127
	Create the Customer window	128
	Create the Product window	130
	Add a menu script to open the Customer window	131
	Add a menu script to open the Product window	133
	Run the application	134
10	Building the First DataWindow Object	137
	Create the new DataWindow object	138
	Preview the DataWindow object	141
	Save the DataWindow object	142
	Make cosmetic changes	143
11	Adding the First DataWindow	145
	Attach the DataWindow object to the control	146
	Run the application	147
12	Building the Second DataWindow Object	149
	Create the new DataWindow object	150
	Select the data source and style	151
	Select the table and columns	152
	Define a retrieval argument	154
	Specify a WHERE clause	155
	Preview the DataWindow object	157
	Save the DataWindow object	159
	Enhance the DataWindow object	160
	Rearrange the columns	161
	Align the labels and columns	162

	Preview again.....	163
13	Adding the Second DataWindow	165
	Attach the DataWindow object to the control	166
	Run the application	168
14	Adding DataWindows to the Product Window.....	171
	Attach the DataWindow objects	172
	Run the application	174
15	Running the Debugger.....	177
	Add a breakpoint for the Application object.....	178
	Add breakpoints for the Welcome and Customer windows.....	180
	Run in debug mode.....	182
	Step through the code line by line.....	184
	Stop at breakpoints and examine variables	186
	Set a watch and a conditional breakpoint	188
16	Creating the Executable File for the Application.....	191
	Specify an application initialization file	192
	Create the application initialization file	193
	Modify the application's Open event script.....	194
	Create the executable file	195
	Create an icon.....	198
	On Windows 95.....	199
	On Windows NT	200
	Test the executable file	202
	What to do next	203
	Glossary	205

About This Book

Subject

This book provides information you need to start using PowerBuilder:

- ◆ **Part 1** is an *overview* of the PowerBuilder development environment
- ◆ **Part 2** is a *tutorial* in which you build your first PowerBuilder application

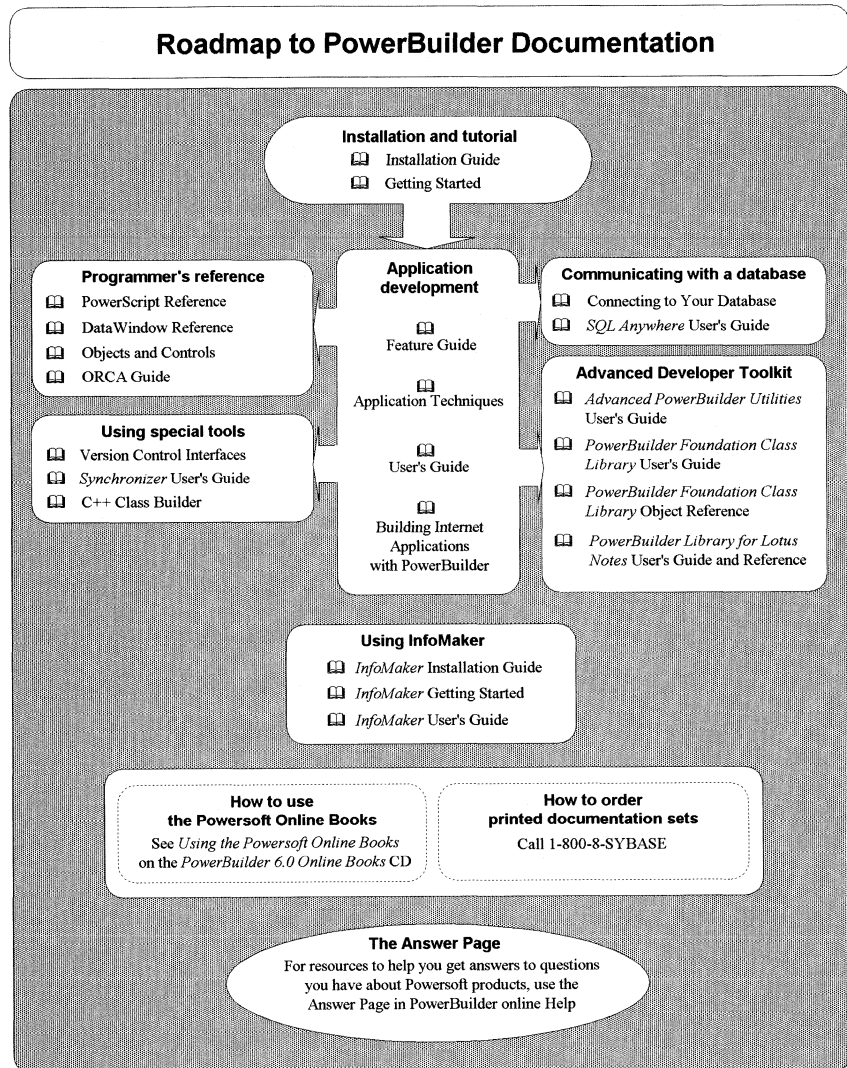
This book is for anyone building applications using PowerBuilder.

Online Help

When you have a question about using PowerBuilder, you can access its extensive online Help system. In online Help you can see:

- ◆ **Procedures** for accomplishing tasks in PowerBuilder
- ◆ **Reference information** about PowerBuilder topics or components
- ◆ **Context-sensitive information** about PowerBuilder functions or reserved words in scripts

FOR INFO For more information on getting online Help in PowerBuilder, see Part 2, "The PowerBuilder Tutorial".



About the PowerBuilder documentation

These are the books in the PowerBuilder documentation set, grouped by topic. Use this table to identify the books you need:

Topic	Book	Description
Installation and tutorial	Installation Guide	Provides instructions for installing PowerBuilder
	Getting Started	Introduces you to PowerBuilder and provides a tutorial you can step through to learn the basics
Application development	Feature Guide	Presents new features in PowerBuilder 6.0; provides a high-level view of the different kinds of applications you can build with PowerBuilder and the tools and techniques you use to build them
	User's Guide	Tells how to use the painters to build objects in PowerBuilder
	Application Techniques	Presents collections of techniques for implementing many common application features, along with deployment details and tips for cross-platform and international development and deployment
	Building Internet Applications with PowerBuilder	Tells how to use PowerBuilder components to build various types of Web applications
Programmer's Reference	PowerScript Reference	Describes syntax and usage information for the PowerScript language, including variables, expressions, statements, functions, and events
	DataWindow Reference	Provides reference information for the DataWindow object; includes syntax for accessing properties and data; lists the DataWindow functions and properties

Topic	Book	Description
	Objects and Controls	Lists properties, events, and related functions for PowerBuilder system objects and controls
	ORCA Guide	Provides reference information for functions you call in C programs to manipulate objects in PBLs (intended for developers of tools that work with PowerBuilder)
Communicating with a database	Connecting to Your Database	Tells how to connect to a database from PowerBuilder; describes how to set up, define, and manage database connections accessed through the Powersoft ODBC interface or one of the native Powersoft database interfaces
	<i>SQL Anywhere</i> User's Guide	Describes the SQL Anywhere DBMS and how to use it with PowerBuilder; includes a reference for all aspects of SQL Anywhere: administration tools, SQL statements, error messages, system tables, and so on
Advanced Developer Toolkit	<i>Advanced PowerBuilder Utilities</i> User's Guide	Describes the Advanced PowerBuilder Utilities, a set of tools that extend the PowerBuilder development environment
	<i>PowerBuilder Foundation Class Library</i> User's Guide	Tells how to use and extend the PowerBuilder Foundation Class Library, which includes objects, services, and utilities to accelerate the application development process
	<i>PowerBuilder Foundation Class Library</i> Object Reference	Describes the objects, instance variables, events, and functions provided with the PowerBuilder Foundation Class Library

Topic	Book	Description
	<i>PowerBuilder Library for Lotus Notes</i> User's Guide and Reference	Tells how to use the PowerBuilder Library for Lotus Notes to develop PowerBuilder applications that access Lotus Notes databases; describes the objects in the PowerBuilder Library for Lotus Notes
Using special tools	Version Control Interfaces	Tells how to use third-party version control systems to manage PowerBuilder objects
	<i>Synchronizer</i> User's Guide	Tells how to use the Synchronizer to build Sync data files that provide synchronization instructions, automatically update Windows applications with the latest files, and add synchronization ability to application windows and Web pages
	C++ Class Builder	Tells how to use the C++ Class Builder to define C++ classes, compile them, store them in DLLs, and then execute their methods from PowerBuilder applications

PowerBuilder documentation packaging

Book	Enterprise	Professional	Desktop
PowerBuilder			
Installation Guide	X P	X P	X P
Getting Started	X P	X P	X P
Feature Guide	X P	X P	X P
Connecting to Your Database	X P	X P	X P
User's Guide	X	X	X
Objects and Controls	X	X	X
Application Techniques	X	X	X
PowerScript Reference	X	X	X
DataWindow Reference	X	X	X
<i>SQL Anywhere</i> User's Guide	X	X	X
<i>Synchronizer</i> User's Guide	X	X	X
Building Internet Applications with PowerBuilder	X	X	X
Version Control Interfaces	X	X	—
<i>Advanced PowerBuilder Utilities</i> User's Guide	X	\$	—
<i>PowerBuilder Foundation Class Library</i> User's Guide	X	\$	—
<i>PowerBuilder Foundation Class Library</i> Object Reference	X	\$	—
<i>PowerBuilder Library for Lotus Notes</i> User's Guide and Reference	X	\$	—
ORCA Guide	X	\$	—
C++ Class Builder	X	—	—
InfoMaker			
Installation Guide	X P	—	—
Getting Started	X P	—	—
Connecting to Your Database	X P	—	—
User's Guide	X	—	—

Key to symbols

Symbol	Means
X	The book is applicable and provided on the PowerBuilder 6.0 Online Books CD
P	The book is provided as a printed book in the box
\$	The book is provided on the Powersoft Online Books CD, but the product must be purchased separately (in which case the printed book, if any, is provided)
—	The book is provided on the Powersoft Online Books CD, but the feature is not available (to use this feature, you can upgrade to a different edition of PowerBuilder)

Arabic and Hebrew

InfoMaker is not available in Arabic- and Hebrew-enabled versions of PowerBuilder Enterprise.

PART 1

Welcome to PowerBuilder

This part is an overview of the PowerBuilder development environment.

Introduction to PowerBuilder

About this chapter

This chapter introduces the PowerBuilder development environment. It defines a number of terms and concepts that are used throughout the PowerBuilder documentation set. It also describes each of the development tools you'll be using in the tutorial in Part Two.

Contents

Topic	Page
About PowerBuilder for Windows	4
What PowerBuilder is	5
PowerBuilder objects	8
The PowerBuilder environment	16

About PowerBuilder for Windows

PowerBuilder for Windows is one of several versions of PowerBuilder that provide application development capabilities on a variety of computing platforms: Windows, Macintosh, and UNIX.

As you'd expect, you can use each version of PowerBuilder to build applications for its own platform. For instance, with PowerBuilder for Windows you can develop Motif applications to be deployed on supported Windows systems. But no matter which platform you're developing on, the PowerBuilder applications you create can be edited or deployed on any of the other platforms.

Each platform version of PowerBuilder is the same basic PowerBuilder product—with the same language, painters, and core features. But they're all tailored to suit their respective platforms when it comes to look and feel, platform-specific features, and platform limitations.

What PowerBuilder is

PowerBuilder is an object-oriented application tool that allows you to build powerful, multitier applications to run on multiple platforms and to interact with various databases. PowerBuilder is one of a group of Sybase products that together provide the tools to develop client/server, distributed, and Internet applications.

What's in a PowerBuilder application?

A PowerBuilder application contains:

- ◆ **A user interface** Menus, windows, and window controls that users interact with to direct an application
- ◆ **Application processing logic** Event and function scripts in which you code business rules, validation rules, and other application processing. PowerBuilder allows you to code application processing logic as part of the user interface or in separate modules, called custom class user objects

PowerBuilder applications are event driven

In a PowerBuilder application, users control what happens by the actions they take. For example, when a user clicks a button, chooses an item from a menu, or enters data into a text box, one or more **events** are triggered. You write scripts that specify the processing that should happen when events are triggered.

Windows, controls, and other application components you create with PowerBuilder each have a set of predefined events. For example, each button has a Clicked event associated with it and each text box has a Modified event. Most of the time, the predefined events are all you need. However, in some situations, you may want to define your own events.

PowerScript language

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 a text box's Modified event might evaluate the data and perform processing based on the data.

The execution of an event script can also cause other events to be triggered. For example, the script for a Clicked event in a button might open another window, triggering the Open event in that window.

PowerScript functions	<p>PowerScript provides a rich assortment of built-in functions you can use to act on the various components of 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 update the database, and so on.</p> <p>You can also build your own functions to define processing unique to your application.</p>
Object-oriented programming with PowerBuilder	<p>Each menu or window you create with PowerBuilder is a self-contained module called an object. The basic building blocks of a PowerBuilder application are the objects you create. Each object contains the particular characteristics and behaviors (properties, events, and functions) that are appropriate to it. By taking advantage of object-oriented programming techniques such as <i>encapsulation</i>, <i>inheritance</i>, and <i>polymorphism</i>, you can get the most out of each object you create, making your work more reusable, extensible, and powerful.</p>
Internet applications	<p>You can develop PowerBuilder applications that run on the Web. PowerBuilder Web applications take advantage of these technologies:</p> <ul style="list-style-type: none">◆ Web.PB Includes object functions that can be invoked by Web browsers◆ DataWindow plug-in Allows a Web browser to display a PowerBuilder report (PSR)◆ PowerBuilder Window plug-in and PowerBuilder window ActiveX Allow a Web browser to display a PowerBuilder window
Distributed applications	<p>PowerBuilder lets you build applications that run in a distributed computing environment. A distributed application lets you:</p> <ul style="list-style-type: none">◆ Centralize business logic on servers◆ Partition application functions between the client and the server, thereby reducing the client workload◆ Build scalable applications that are easy to maintain
Cross-platform development	<p>PowerBuilder supports cross-platform development and deployment. For example, you can develop an application using PowerBuilder on Windows and deploy the very same application on UNIX—or vice versa. You can even have a cross-platform team of developers, some using Windows and some using UNIX, 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.</p>

FOR INFO For information about porting applications, see the chapter on building an application for multiple platforms in *Application Techniques*.

Database connectivity

PowerBuilder provides easy access to corporate information stored in a wide variety of databases using Powersoft database interfaces.

A **Powersoft database interface** is a native (direct) connection to a database. For example, if you have the appropriate SQL Server 4.x software installed, you can access a SQL Server 4.x database through the Powersoft SQL Server 4.x interface.

Each Powersoft database interface has its own interface DLL that communicates with the specified database. When you use a Powersoft database interface, the interface DLL connects to the database through the database vendor's application programming interface (API).

PowerBuilder objects

The basic building blocks of a PowerBuilder application are objects:

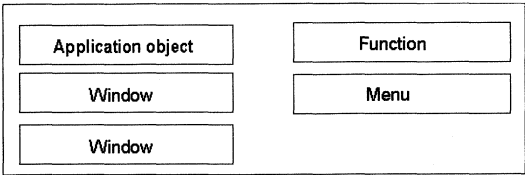
Object	Use
Application	Entry point into an application
Window	Primary interface between the user and a PowerBuilder application
DataWindow	Retrieves and manipulates data from a relational database or other data source
Menu	List of commands or options that a user can select in the currently active window
Global function	Performs general-purpose processing
Query	SQL statement used repeatedly as the data source for a DataWindow object
Structure	Collection of one or more related variables grouped under a single name
User object	Reusable processing module or set of controls
Library	Stores PowerBuilder objects, such as windows and menus
Project	Packages application for distribution to users

These objects are described in more detail in the following sections.

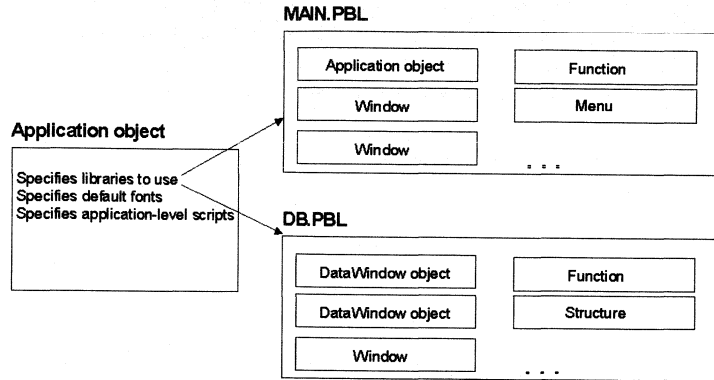
Application object

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

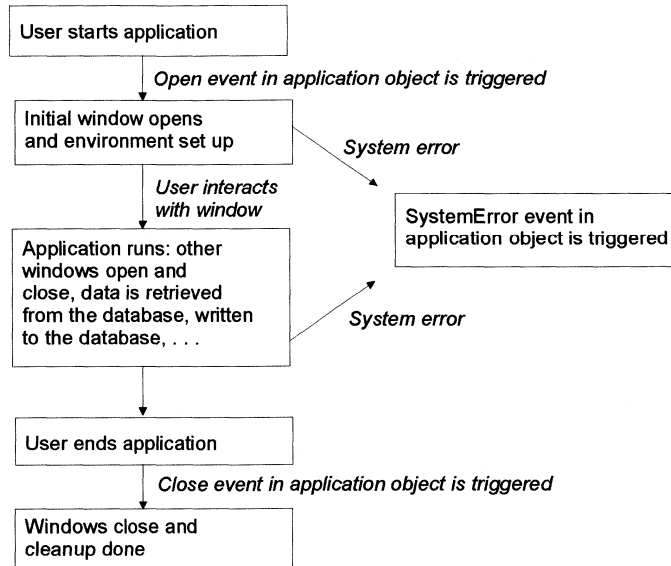
MAIN.PBL



What the Application object does 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.



What happens at application startup time 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 to a preferences file. If there are serious errors during execution, the Application object's SystemError event is triggered.



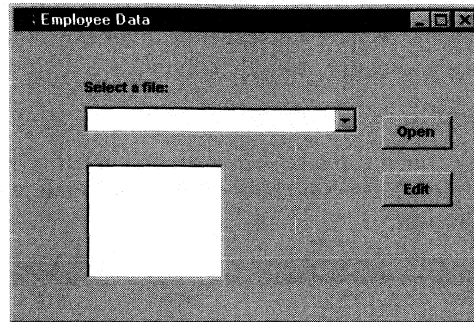
Windows

Windows are the primary interface between the user and a PowerBuilder application. Windows can display information, request information from a user, and respond to the user's mouse or keyboard actions.

A window consists of:

- ◆ **Properties**, which define the window's appearance and behavior (for example, a window might have a title bar or a Minimize box)
- ◆ **Events**, which are triggered by user actions
- ◆ **Controls**, which are placed in the window

Windows have various kinds of controls, as illustrated in the following picture:



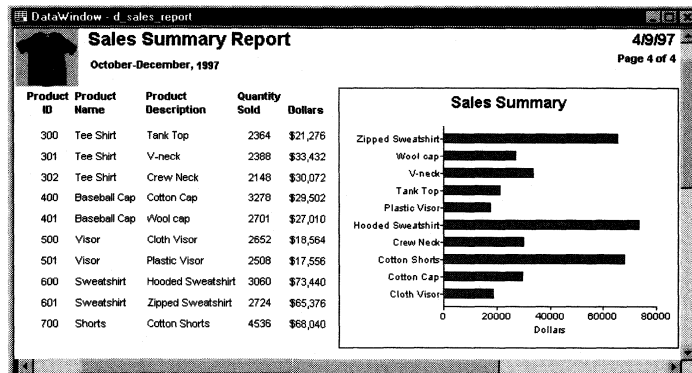
At the top of the window is the title bar with minimize, maximize, and close buttons. This window contains a static text control, a dropdown listbox, and a multiline edit box, as well as two command buttons.

DataWindow objects

A **DataWindow object** is an object that you use to retrieve and manipulate data from a relational database or other data source (such as an Excel worksheet or dBASE file).

Presentation styles DataWindow objects also handle the way data is presented to the user. You can choose from several presentation styles. For example, you can display the data in a Tabular or a Freeform style.

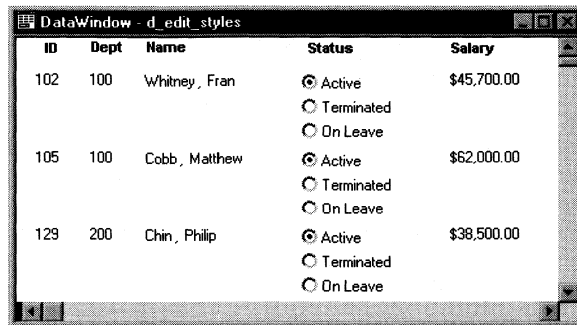
There are many ways to enhance the presentation and manipulation of data in a DataWindow object. For example, you can include computed fields, pictures, and graphs that are tied directly to the data retrieved by the DataWindow.



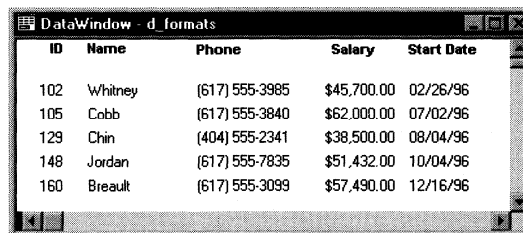
Display formats, edit styles, and validation You can specify how to display the values for each column, and you can validate data entered by users in a DataWindow object. You do this by defining display formats, edit styles, and validation rules for columns.

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.



- ◆ If the data includes phone numbers, salaries, and dates, you can format the display to suit the data.



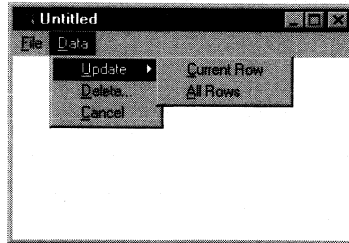
- ◆ If a column can take numbers only in a specific range, you can specify a simple validation rule for the data so that you don't need to write code to make sure users enter valid data.

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). Most windows in a PowerBuilder application have menus associated with them.

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, or use accelerator keys defined for the items.

A dropdown menu is a menu under an item in the menu bar. A cascading menu is a menu that appears to the side of an item in a dropdown menu.



Each choice in a menu is defined as a Menu object in PowerBuilder. The preceding window shows two Menu objects on the menu bar (File and Data), three Menu objects on the Data menu (Update, Delete, and Cancel), and two Menu objects on the cascading menu beside Update (Current Row and All Rows).

Using ellipsis points

Ellipsis points (...) following a menu item mean that clicking the item displays a dialog box.

Global functions

PowerBuilder lets you define two types of functions:

- ◆ **Object-level functions** are defined for a particular type of window, menu, or other object type and are encapsulated within the object for which they are defined
- ◆ **Global functions** are *not* encapsulated within another object, but instead are stored as independent objects

Unlike object-level functions, global functions do not act on particular instances of an object. Instead, they perform general-purpose processing such as mathematical calculations or string handling.

Queries

A **query** is a SQL statement that is saved with a name so that it can be used repeatedly as the data source for a DataWindow object. Queries enhance developer productivity, because they can be coded once but reused as often as necessary.

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*.

What structures do Structures allow you to refer to related entities as a unit rather than individually. For example, you can define the user's ID, address, access level, and a picture (bitmap) of the employee as a structure called `user_struct`, and then refer to this collection of variables as `user_struct`.

Two types There are two kinds of structures:

- ◆ **Object-level structures**, which are associated with a particular type of object such as a window or menu. These structures can always be used in scripts for the object itself. You can also choose to make the structures accessible from other scripts.
- ◆ **Global structures**, which are not associated with any object in an application. You can declare an instance of the structure and reference it in any script in an application.

User objects

Applications often have features in common. For example, several applications might have a Close button that performs a certain set of operations and then closes the window. Or they might have DataWindow controls that perform the same type of error checking. Or several applications might all require a standard file viewer.

When to define a user object If you find yourself using the same application feature repeatedly, you should define a **user object**. You define the user object once and use it as many times as you need.

Two types There are two types of user objects:

- ◆ **Visual user objects** are reusable controls or sets of controls that have a consistent behavior. For example, a visual user object could consist of several buttons that function as a unit. The buttons could have scripts associated with them that perform standard processing. Once the object was defined, you could use it as often as you needed.
- ◆ **Class user objects** are reusable processing modules that have no visual component. You typically use class objects to define business rules and other processing that acts as a unit. For example, you might want to calculate commissions or perform statistical analysis in several applications. To do this, you could define a class user object. To use a class user object, you create an instance of the object in a script and call its functions.

Custom class user objects, which define functions and variables, are the foundation of PowerBuilder distributed computing.

Libraries

You save objects, such as windows and menus, in PowerBuilder libraries (PBL files). When you run an application, PowerBuilder retrieves the objects from the library. Applications can use as many libraries as you want. When you create an application, you specify which libraries it uses.

Projects

To allow users to execute your application the same way they execute other applications, you create a **project object**.

Packaging an application The project object can package an application in either of two ways:

- ◆ 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

Providing additional resources When you package an application, you may also need to provide some additional resources, such as bitmaps and icons. PowerBuilder allows you to include additional resources in an executable and/or dynamic libraries, or distribute them separately.

The PowerBuilder environment

There are many elements of PowerBuilder that you use to build an application.

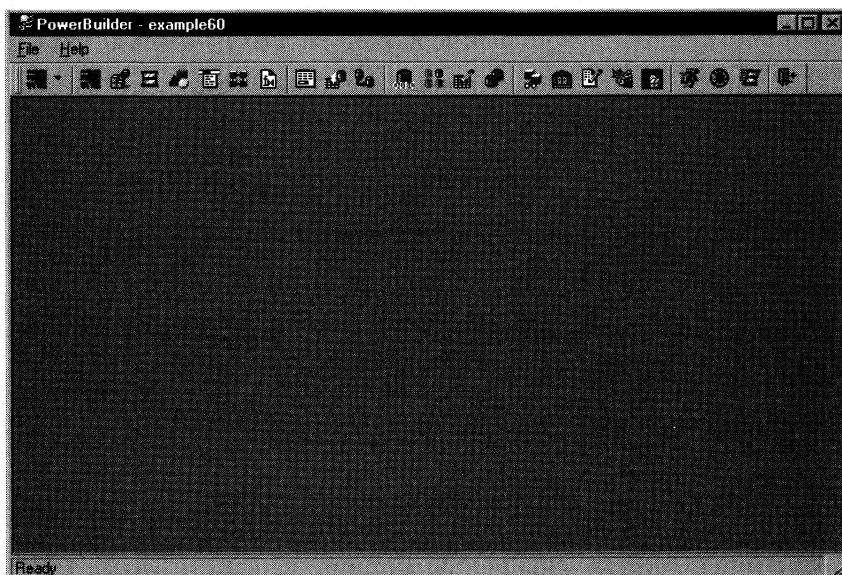
Painters

You build the components of an application using painters. Painters provide an assortment of tools for building objects.

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

The PowerBuilder window

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



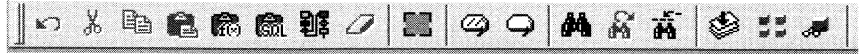
You can open painters and perform other tasks by clicking buttons in the PowerBar.

PowerBar

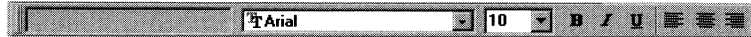
The PowerBar displays when you begin a PowerBuilder session. The PowerBar is the main control point for building PowerBuilder applications. From the PowerBar, you can open a PowerBuilder painter, debug or run the current application, display Help, or customize PowerBuilder to meet your needs.

PainterBar

When you open a painter, PowerBuilder displays a new window that has a workspace in which you design the object you're building. PowerBuilder also displays a PainterBar with buttons that provide easy access to the tools available in the painter.

**StyleBar**

The StyleBar displays when you open any painter that could contain text controls, such as the Window painter. Using buttons on the StyleBar, you can modify text properties such as the font and point size.

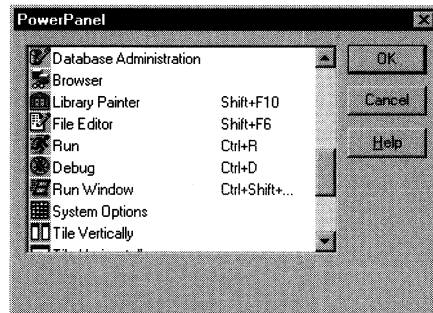
**PowerTips**

When you leave the mouse pointer over a button for a second or two, PowerBuilder displays a brief description of the button (a PowerTip).














**PowerPanel**










PowerBuilder also provides a PowerPanel, which like the PowerBar provides buttons for opening painters and performing other activities. The PowerPanel contains 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.

**PowerBar and PowerPanel buttons**

The buttons in the PowerBar and PowerPanel give you access to the most commonly used PowerBuilder tools.

Button	Tool	Use to
	Application painter	Specify application information, such as its name and the PowerBuilder libraries in which the application's objects reside
	Project painter	Create an executable that runs outside PowerBuilder and specify the components that go into the application
	Window painter	Build the windows that will be used in the application
	User Object painter	Build custom objects that you can save and reuse repeatedly in windows
	Menu painter	Build the menus that the windows will use
	Structure painter	Define structures (groups of variables) for use in an application
	Function painter	Build functions to perform processing specific to an application
	DataWindow painter	Build the interface to the database
	Query painter	Define and save SQL SELECT statements for reuse with DataWindow objects
	Pipeline painter	Copy data from one database to another
	Database profile	Specify how to connect to a database
	Table painter	Create and alter database tables
	Database painter	Maintain databases, control user access to databases, and manipulate data in databases

Button	Tool	Use to
	Browser	View object information (such as object properties or global variables) and copy, export, or print it
	Library painter	Create and maintain libraries of PowerBuilder objects
	Edit	Edit a file
	Web.PB wizard	Invoke the Web.PB wizard
	OLE Generate Registration	Invoke the OLE Automation Registry Generation tool
	Run	Run the current PowerBuilder application
	Debug	Set breakpoints, step through your code line by line, and look at variables during execution
	Run window	Run a specified window
	Exit	Exit from PowerBuilder

Customizing toolbars

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

You can also define toolbars and customize them to suit your needs.

The PowerBuilder Tutorial

This part is a tutorial that shows you how to get started with PowerBuilder. It provides step-by-step instructions for creating a simple database application.

About this chapter

This chapter describes what you will do in the PowerBuilder tutorial and how to get set up for it.

Contents

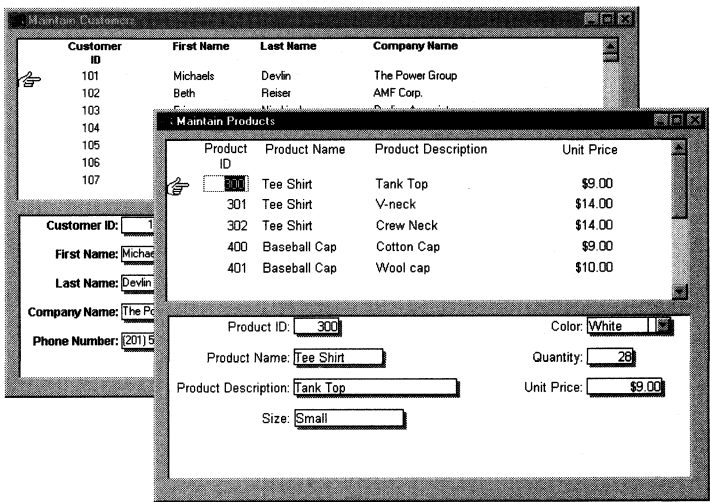
Topic	Page
What you will do	24
How you will proceed	26
How long it will take	26
What you will learn	26
Setting up	28

What you will do

The PowerBuilder tutorial is a series of 14 exercises in which you build a Multiple Document Interface (MDI) database application for a fictional company called SportsWear, Inc. The application allows you to retrieve customer and product information from the database and perform insert, update, and delete functions against the customer and product data.

Customer and Product windows

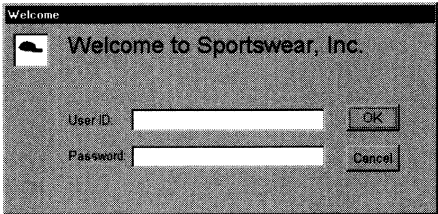
The tutorial application includes two windows that provide access to the Customer and Product tables in the Powersoft Demo Database.



Both windows are master/detail windows: each allows you to display a master list of rows in a particular table and also see detailed information for each row in the table. For example, the top half of the Maintain Products window contains a list of products with a pointer to a single product; the bottom half of the window displays extra detail for the current product.

Logon window

The tutorial application also includes a logon window that allows you to connect to the database at startup time.



A cross-platform application

This tutorial tells you how to build a cross-platform application. That means the objects you create using PowerBuilder for Windows will also run on UNIX or the Macintosh with little or no modification and will display almost identically in all environments.

How you will proceed

Chapter	What you will do
3	To begin the tutorial, you will start PowerBuilder, begin familiarizing yourself with the development environment, and use the Application painter to create the Application object
4	Next you will associate a library with the application
5, 6	Next you will create a logon window to allow the user to connect to the database. You will also see how database profiles are defined in the PowerBuilder environment
7	Next you will build some menus for the application
8, 9	Next you will create a standard master/detail window that will serve as the ancestor for two other descendent windows, one for viewing and updating rows in the Customer table and the other for viewing and updating rows in the Product table. You will also add scripts to allow users to retrieve data and perform insert, update, and delete operations against the database
10—14	Next you will build the DataWindow objects that retrieve customer and product information. After you build these objects, you will add them to the Customer and Product windows
15	Next you will run the tutorial application in debug mode. You will see how to set breakpoints in scripts, step through the code, and display the contents of variables
16	Finally, you will create an executable file that you can use to run the application outside the PowerBuilder environment

How long it will take

You can do the entire tutorial in about four to five hours, or you can stop after any chapter and continue at another time.

What you will learn

You will learn basic PowerBuilder techniques and concepts, including:

How to use the	To
Application painter	Define an Application object and application-level scripts
Window painter	Create SingleLineEdit controls, StaticText controls, CommandButton controls, DataWindow controls, window-level scripts, and control-level scripts
DataWindow painter	Define selection and display options
PowerScript painter	Define scripts for applications, windows, window controls, and menus
Menu painter	Define menus, menu items, accelerators, and shortcut keys
Debugger	Identify logic errors that may cause problems at execution time
Project painter	Create an executable version of an application

This tutorial will not make you an expert in PowerBuilder. Only experience building real-world applications can do that. But it will give you hands-on experience and provide a foundation for continued growth.

Setting up

Before you start the tutorial, you need to make sure you can connect to a database and that you have the tutorial files. The tutorial assumes you are connected to the Powersoft Demo Database.

The tutorial uses the following files:

File	Contents
TUTOR_PB.PBL	PowerBuilder library that contains several objects you will use in the tutorial
TUTORIAL.ICO	An icon
TUTSPORT.BMP	A bitmap

Make sure the files you need for the tutorial are on your hard disk in the PBTUTOR directory, a subdirectory of the PowerBuilder installation directory. When you have finished with the tutorial, you can delete the files.

Starting PowerBuilder

This chapter provides the information you need to start running PowerBuilder, create an Application object, and become familiar with the PowerBuilder environment.

In this chapter you will:

- ◆ Start PowerBuilder
- ◆ Create and save the Application object
- ◆ Control the toolbars
- ◆ Use online Help
- ◆ Use the Powersoft Online Books

How long will it take?

About 15 minutes.

Start PowerBuilder

Where you are

- > Start PowerBuilder
 - Create and save the Application object
 - Control the toolbars
 - Use online Help
 - Use the Powersoft Online Books
-

Now you will start PowerBuilder.



1 Read the installation notes for this release.

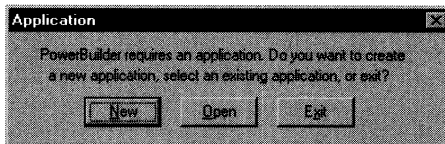
Any last-minute items will be documented there.

2 Double-click the PowerBuilder icon.

If the Welcome to PowerBuilder window displays, click Close.

If PowerBuilder prompts you for an application

If you are prompted for an application, click New and continue at step 3 in "Create and save the Application object" next.



If this is the first time you have started PowerBuilder and the Code Examples sample application has not been installed, you will need to create a new application now, as explained next.

Create and save the Application object

Where you are

- Start PowerBuilder
 - > Create and save the Application object
 - Control the toolbars
 - Use online Help
 - Use the Powersoft Online Books
-

Now you will create an Application object for the tutorial application and name it PBTUTOR. You will add objects to it later.

If PowerBuilder prompted you for an application

If you were prompted for an application, begin this exercise at step 3.



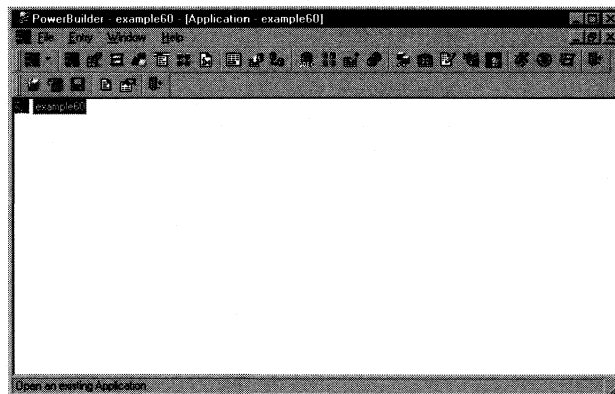
- 1 Click the *App!* button in the PowerBar.

Two Application painter buttons

When you first start PowerBuilder, there are two Application painter buttons. Clicking either button launches the Application painter.

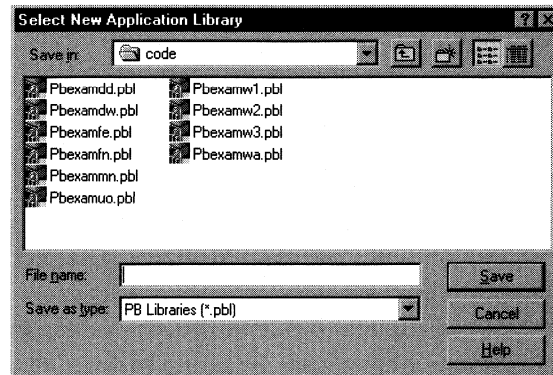
Clicking the down arrow next to the first button displays a dropdown list that shows all the PowerBar buttons. If you select a button from the dropdown list, it replaces the first button in the PowerBar.

The current Application object opens and displays in the workspace.



2 Select File>New from the menu bar.

The Select New Application Library dialog box displays.



About PowerBuilder libraries

The PBL extension stands for PowerBuilder library and is pronounced *pibble*. All PowerBuilder objects (applications, windows, DataWindows, menus, and user-defined functions and objects) are stored in PowerBuilder libraries.

3 Navigate to the pbtutor directory.

If you are in the code directory, navigate to the PowerBuilder install directory, then double-click the PBTUTOR directory.

**4 Type *pbtutor.pbl* in the File Name box.
Click Save.**

Windows NT

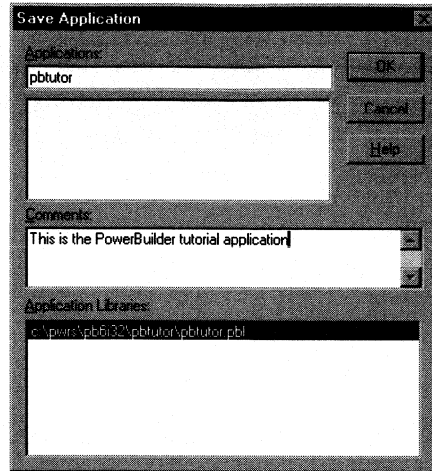
If you are running a version of Windows NT that has the Windows 3.1 interface, click OK.

Cross-platform naming conventions

Providing a filename in this format (name with 8 or fewer characters followed by 3-character extension) provides compatibility with platforms that restrict filenames to this format, such as Windows 3.1.

The Save Application dialog box displays.

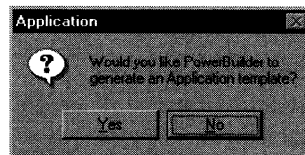
- 5 **Type *pbtutor* in the Applications box.**
Type *This is the PowerBuilder tutorial application* in the Comments box.



This names the application and associates a descriptive comment with the Application object. The comment you add here displays in the Library painter and helps to identify objects. Comments are optional.

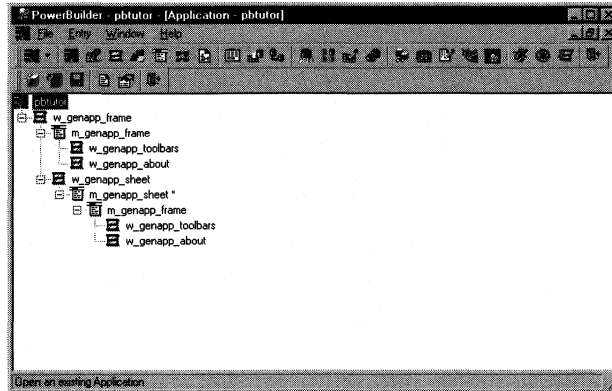
- 6 **Click *OK*.**

The Application dialog box displays.



7 Click Yes.

PowerBuilder creates a framework for the application and displays it in the Application painter workspace.



PowerBuilder creates a library named PBTUTOR.PBL and creates and saves the Application object named pbtutor in the library. PowerBuilder also creates a simple application framework that includes an MDI frame window, an MDI sheet window, some menus, and some scripts for opening the windows. The tree structure in the Application painter shows how the windows and menus reference each other.

At this point, the pbtutor application is the current application. Notice that the name pbtutor now displays in the PowerBuilder title bar and the Application painter title bar.

Using an application framework

To get the most out of PowerBuilder, you should take advantage of its object-oriented features. The best way to do that is to base each project you begin on an **application framework**. An application framework is a collection of **base classes** (ancestor objects) from which you can inherit most of the objects you need to develop for a project. When you're working on a production application, you will want to define your own application framework and customize it to suit your requirements.

Each time you start up, PowerBuilder opens the current application. Whenever you want, you can change the current application in the Application painter.

Control the toolbars

Where you are

Start PowerBuilder

Create and save the Application object

> Control the toolbars

Use online Help

Use the Powersoft Online Books

Now you will learn how to control the location and appearance of the PowerBuilder toolbars.

PowerBuilder has three toolbars: the PowerBar, the PainterBar, and the StyleBar. You can choose whether to display text in the buttons, add buttons for operations you perform frequently, move the toolbars around, and dock toolbars wherever you like.

FOR INFO For more information about the PowerBar, the PainterBar, and the StyleBar, see "The PowerBuilder environment" on page 16.

In addition, you can define toolbars and customize them to suit your individual requirements.

- 1 Move the pointer to any one of the buttons on the PainterBar. Do not click.**

PowerBuilder displays a **PowerTip** identifying the button's function.



PowerTips are displayed whenever the pointer pauses over a toolbar button.

- 2 Select *Window>Toolbars* from the menu bar.**

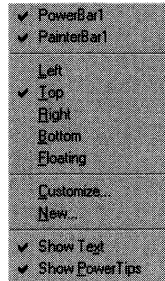
The Toolbars dialog box opens.

- 3 Select the *Show Text* checkbox, then click the *Close* button.**

PowerBuilder displays a label on each of the buttons in the PowerBar and the PainterBar.

- 4 **Move the pointer next to the PainterBar and click the right mouse button.**

The popup menu for the toolbars displays. The popup menu allows you to put toolbars in many different locations: left, top, right, bottom, and floating.



About popup menus

Throughout PowerBuilder, popup menus provide a fast way to do things. The menu changes depending on which painter you are using and where you are in the workspace when you click the right mouse button.

- 5 **Select *Floating* in the popup menu and release the mouse button.**

The PainterBar changes to a floating toolbar.

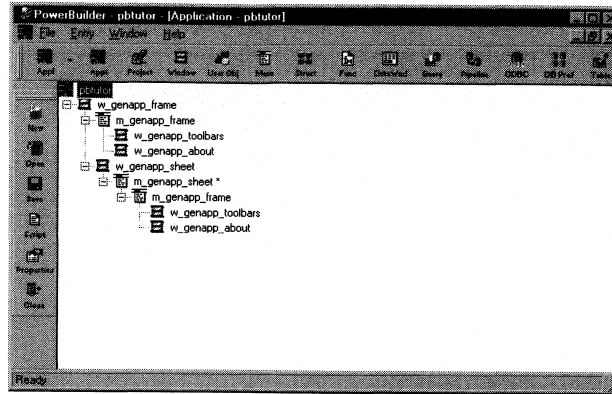


- 6 **Move the pointer to an empty space in the PainterBar and drag it to a different location.**

How to drag the toolbar

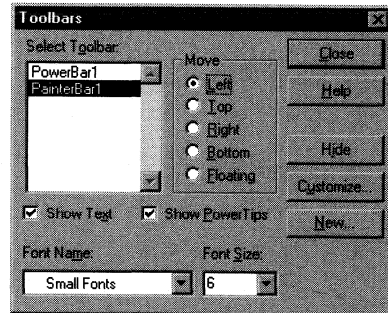
To drag the toolbar, press and hold the left mouse button while you move the mouse (an outline displays the location of the toolbar). When the toolbar is where you want it, release the mouse button.

- 7 Drag the floating toolbar toward the left side of the workspace. Release the mouse button when the toolbar becomes a vertical bar.



- 8 Select *Window>Toolbars* from the menu bar.

The Toolbars dialog box displays. The position of the currently selected toolbar shows as the selected radio button.



- 9 Click *PainterBar1*.
Click *Top*.

This positions the PainterBar at the top of the screen.

- 10 Click *Close*.

Now you know several ways to move the toolbars. For the rest of this tutorial, the illustrations show the toolbars at the top of the screen and text on the buttons.

Using the PowerPanel dropdown toolbar

When you show text on the buttons, some of the buttons on the PowerBar may not fit in your display. You can access all the painters from the PowerPanel dropdown toolbar—click the down-arrow next to the leftmost button on the PowerBar to display it.

Use online Help

Where you are

Start PowerBuilder

Create and save the Application object

Control the toolbars

> Use online Help

Use the Powersoft Online Books

PowerBuilder has an extensive online Help system that supplements the information in the PowerBuilder documentation. The online Help provides step-by-step directions for using the painters as well as reference information for all PowerBuilder features. Before you begin building the application, you should try using the Help system.

Now you will:

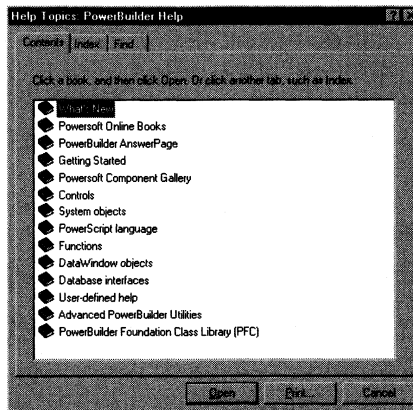
- ◆ Display a Help topic by navigating from the table of contents
- ◆ Use the Index facility to search for a Help topic
- ◆ Display context-sensitive Help for a dialog box

Display a Help topic by navigating from the table of contents

The table of contents gives you access to a variety of PowerBuilder topics. Now you will select a Help topic by searching from the table of contents.

- 1 **Select *Help>Help Contents* from the menu bar.**
or
Press F1.

PowerBuilder displays the PowerBuilder Help Topics window.



- 2 **Select *Getting Started*.**
Click *Open*.

PowerBuilder displays a list of subtopics that are available for this topic.

- 3 **Select *PowerBuilder environment*.**
Click *Display*.

PowerBuilder displays Help about the PowerBuilder environment.

You can view information on related topics by clicking highlighted text (or links). These links are highlighted with a solid underline (go to another Help topic) or a dashed underline (display a popup Help window).

You can access the Help contents or index, as well as the PowerBuilder Online Books, from buttons at the top of the Help screen.

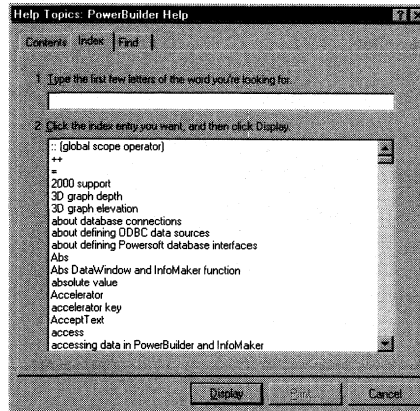
To return to the Help contents, click Contents.

Use the Index facility to search for a Help topic

The Index allows you to search for more specific Help topics. Now you will use the Index to find a Help topic.

1 Click the *Index* tab at the top of the Help window.

PowerBuilder displays a field for entering search strings and the list of index entries.



2 Type *report* to scroll the index to a list of related topics. Double-click *report object properties*.

PowerBuilder displays the *Properties for report objects* Help topic.

3 Close the Help window.

Display context-sensitive Help for a dialog box

When you display a dialog box in PowerBuilder, a Help button in the dialog box allows you to navigate directly to a Help topic on that dialog box. Now you will display context-sensitive Help from the Toolbars dialog box.

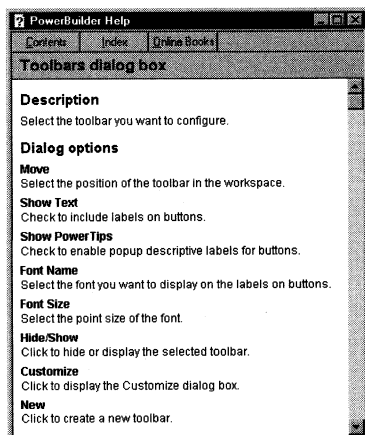
- 1 **Select** *Window>Toolbars*.

or

If there are no painters open, select *File>Toolbars*.

- 2 **Click** *Help*.

PowerBuilder displays context-sensitive Help for the Toolbars dialog box.



Other online resources In addition to the online Help, you can access other online resources:

- ◆ Powersoft Online Books

Some Help windows have direct links to relevant sections in an online book: Clicking one of these links opens a specific book and displays the appropriate page.

 [Click to see more information in the Powersoft Online Books.](#)

You can search the online books, print selected sections, copy to the clipboard, and add bookmarks and annotations.

FOR INFO For more information about using the online books, see the *DynaText Reader Guide* and *Guide to the PowerBuilder Documentation* in the *Using the Powersoft Online Books* collection.

◆ Books on the Web

You can access Powersoft documentation on the Web from the Sybase Web site: www.sybase.com. From this site, you can add bookmarks so you can find information quickly using a browser.

Use the Powersoft Online Books

Where you are

- Start PowerBuilder
 - Create and save the Application object
 - Control the toolbars
 - Use online Help
 - > Use the Powersoft Online Books
-

If you can't find the answer you need in online Help, you can always look in the PowerBuilder books, which are available in hardcopy and online. The Powersoft Online Books are online versions of the hardcopy books. You can get to the Powersoft Online Books directly from the Help system.

Now you will learn how to use the Powersoft Online Books.

- 1 **In the *Toolbars* Help window, click the *Online books* button in the Button bar.**

The library window opens showing all the Powersoft book collections on the left, and all the books in the first collection on the right.

Installing the Powersoft Online Books

If you see an error message, the Powersoft Online Books may not be installed.

FOR INFO For how to install them, see the *PowerBuilder Installation Guide*.

- 2 **Select a collection name in the left panel.**

The panel on the right displays all the books in that collection.

- 3 **Double-click the title of the book you want.**

PowerBuilder opens the book with a table of contents on the left and the full text of the book on the right. You can scroll through the book you have opened, or look at other books or collections.

- 4 **To continue with the tutorial, select *File>Exit* to close the Powersoft Online Books viewer.**

- 5 **Close the Help window in PowerBuilder.**

Modifying the Application Object

Before building the application, you need to modify some properties of the Application object.

In this chapter you will:

- ◆ Specify an icon for the application
- ◆ Add a library to the search path
- ◆ Run the application

How long will it take?

About 10 minutes.

Specify an icon for the application

Where you are

- > Specify an icon for the application
 - Add a library to the search path
 - Run the application
-

Now you will specify an icon for the application.

The icon displays in the Windows workspace when you minimize the application during execution. PowerBuilder includes the icon automatically when you create an executable file.



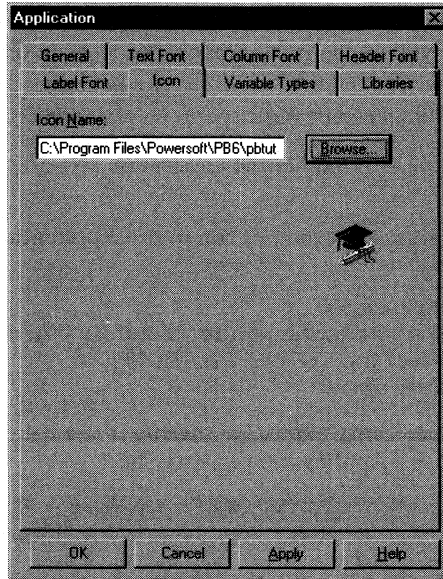
- 1 In the Application painter, click the Properties button in the PainterBar.**

The Application property sheet displays.

- 2 Select the *Icon* tab.**
- 3 Click Browse.**
Navigate to the pbtutor directory if necessary.
- 4 Select the *TUTORIAL.ICO* file.**

- ◆ **Windows NT 3.51** Click OK.
- ◆ **Windows 95 and Windows NT 4.0** Click Open. (You may not see the file extension ICO. For more information, see your Windows 95 documentation.)

The tutorial icon displays in the Application property sheet.



Add a library to the search path

Where you are

Specify an icon for the application

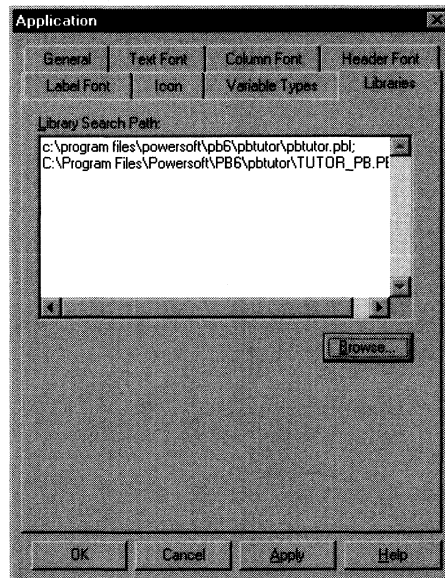
- > Add a library to the search path
 - Run the application
-

While you are defining properties for the tutorial application's application object, you need to add a library to the library search path. The library you will add contains some objects you will need later on in the tutorial.

Now you will modify the application properties to include another library.

- 1 **Select the *Libraries* tab in the Application property sheet.**
- 2 **Click *Browse*.**
Navigate to the *pbtutor* directory if necessary.
Select the *TUTOR_PB.PBL* file.
Click *Open*.

The library appears at the end of the list of libraries in the search path in the Application property sheet.



3 Click OK.

You return to the Application painter workspace.

4 Select *File>Save* from the menu bar.

This saves the updated Application object.



5 Click the *Close* button on the PainterBar.

You return to the PowerBuilder initial window.

Run the application

Where you are

Specify an icon for the application

Add a library to the search path

> Run the application

Now you will run the application to see how it works. At this point the application doesn't do very much. However, by running the application, you can see the windows and menus that were created for you when you generated the application template.

Showing the Run button

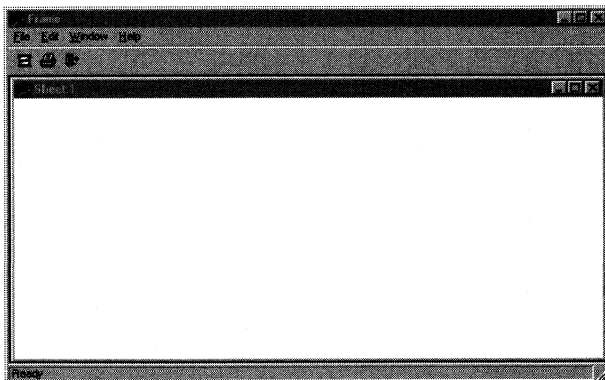
The Run button may not be visible in your window if you have text showing on the buttons in the PowerBar. If you can't see the Run button, click on the down arrow next to the Appl button to display a dropdown list that shows all the buttons available in the PowerBar.



1 Click the *Run* button on the PowerBar.

The application begins running.

The script PowerBuilder generated for the application Open event opens the MDI frame window.



The script generated for the Open event of the MDI frame window opens an MDI sheet inside the frame. The sheet has the title Sheet:1. Notice that the application has a menu bar and a toolbar, both of which were created for you when you generated the application template.

2 Select *File>New* to display a new MDI sheet.

The application opens the new sheet. Sheet:2 displays in the title bar of the new window to indicate that this is the second sheet.

3 Select *Window>Cascade* to cascade the windows you've opened so far.

The windows cascade inside the MDI frame.

4 Select *File>Toolbars* from the menu bar.

The application displays the Toolbars dialog box.

5 Select *Floating* to make the toolbar float within the MDI frame.

The toolbar floats within the frame.

6 Select *Top* to position the toolbar at the top of the MDI frame.

The toolbar is positioned at the top of the frame.

7 Click *Done* to close the *Toolbars* dialog box.

8 Select *File>Exit* to terminate the application.

PowerBuilder terminates the application and you return to the PowerBuilder initial window.

Building a Logon Window

Windows are the main interface between the user and PowerBuilder applications. Windows can display information, request information from a user, and respond to mouse or keyboard actions.

Windows are separate objects that you create using the Window painter. In PowerBuilder, you can create windows at any time during the application development process.

In this chapter you will:

- ◆ Build a logon window for the application
- ◆ Add a Picture control to the window
- ◆ Add three StaticText controls to the window
- ◆ Add two SingleLineEdit controls to the window
- ◆ Add two CommandButton controls to the window
- ◆ Write the PowerScript code required to open the window when the application starts

How long will it take?

About 30 minutes.

Create a new window

Where you are

- > Create a new window
 - Add controls to the window
 - Change the tab order on the window
 - Save the window
 - Preview the window
 - Write the script to open the window
-

Now you will create a new window for the application. The window you will create is a logon window that allows the user to enter a user ID and password and connect to the database. The logon window is a **response window**.

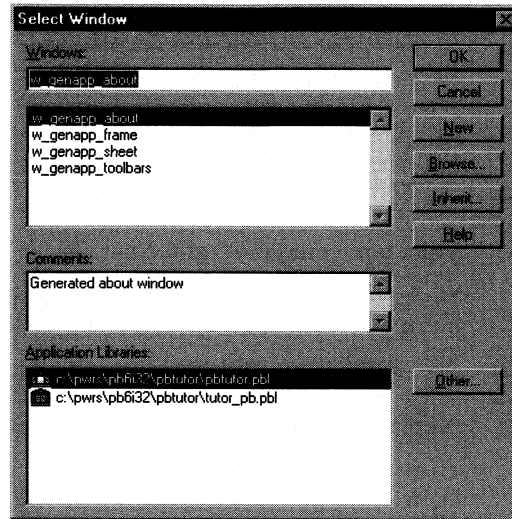
About response windows

Response windows are dialog boxes that request information from the user. Response windows are **application modal**. That means 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. The user can go to other applications, but when the user returns to the application, the response window is still active.



1 Click the *Window* button in the PowerBar.

The Select Window dialog box opens, listing several windows that were created for you when you generated the application template.

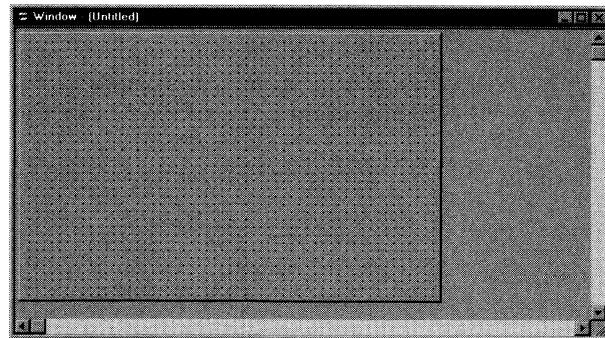


From this dialog box you can open an existing window and modify it, create a new window, or inherit from an existing window to create a new descendent window. You are going to create a new window.



2 Click the *New* button.

The Window painter workspace opens. The rectangle represents the window you are building.



Your background color may be different

If the window displays with a white background, the Default to 3D option has been disabled. When this option is enabled, the Window painter's default environment includes a colored background and 3D controls. The color of the background depends on the color palette you are using.

To change this setting, select Design>Options from the menu bar, then enable the Default to 3D option and reopen the Window painter.

3 Make the window rectangle smaller.

Move the pointer to the bottom edge of the rectangle. When it turns into a double-headed arrow, drag the arrow up to make the rectangle a little smaller.

4 Display the popup menu for the window.

To display the window's popup menu, right-click on the background of the window.

5 Select *Properties*.

The Window property sheet displays with the General tab selected.

6 With *Untitled* selected in the *Title* box, type *Welcome*.

7 Select *Response* in the *Window Type* dropdown listbox.

8 Click *OK*.

You return to the Window painter workspace. The changes you made will not be visible until you preview the design later.

Add controls to the window

Where you are

- Create a new window
 - > Add controls to the window
 - Change the tab order on the window
 - Save the window
 - Preview the window
 - Write the script to open the window
-

Now you will modify the window:

- ◆ Add a Picture control
- ◆ Add StaticText controls
- ◆ Add SingleLineEdit controls
- ◆ Add CommandButton controls

Controls allow users to interact with PowerBuilder objects, such as windows and DataWindows. Once you add a control, you can specify the properties of that control, such as the text and font that appears on a StaticText control or the command PowerBuilder executes when users click a CommandButton control.

Add a Picture control

Now you will add a Picture control to the logon window.



- 1 **Click the *Picture* button (not *PictureButton*) in the PainterBar.**
or
Select *Controls>Picture* from the menu bar.

If the *Picture* button isn't in the PainterBar

The *Picture* button isn't visible in the PainterBar when you first start PowerBuilder. You first need to click the down arrow next to the *Button* toolbar button. When the dropdown list displays, click the *Picture* button. The button you select replaces the *Button* toolbar button on the PainterBar.

- 2 **Click where you want the control in the window.**

A *Picture* control displays at the selected location.

How to delete controls

If you add a control to the window and later decide you don't want it, select the control and click the PainterBar's *Clear* button or press the *DELETE* key. This deletes the control and its scripts.

- 3 **Select *Properties* from the *Picture* control's popup menu.**

To display the *Picture* control's popup menu, right-click the control. When you select *Properties*, the *Picture* property sheet displays with the *General* tab selected.

- 4 **Select the text *p_1* in the *Name* box.**
Type *p_sports* in the *Name* box.

This names the *Picture* control. The prefix *p_* is standard for *Picture* controls.

- 5 **Click *Browse*.**

- 6 **Navigate to the *pbtutor* directory if necessary.**
Select the *TUTSPORT.BMP* file.
Click *Open*.

Windows NT

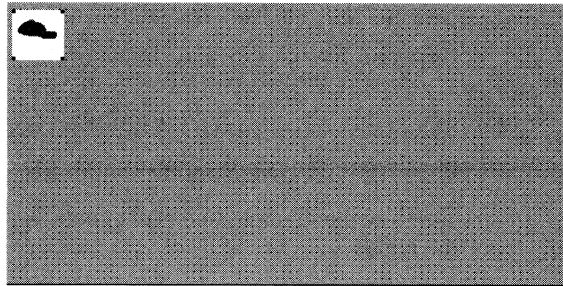
If you are using a Windows 3.1 interface, click OK.

The Select Picture dialog box closes and the Picture dialog box displays again.

- 7 Select the *Original Size* checkbox.
Click OK.**

You return to the Window painter workspace. The image of a baseball cap appears in the Picture control. The small black boxes in the corners show that this control is selected.

- 8 Drag the Picture control to the upper-left corner of the window.**



Add StaticText controls

Now you will add StaticText controls to the logon window.



- 1 Click the *Text* button in the PainterBar.
or
Select *Controls>StaticText* from the menu bar.**

The Text button is on the dropdown toolbar next to the fourth button in the PainterBar.

- 2 Click at the top of the window to the right of the Picture control you just added.**

A StaticText control displays at the selected location.

- 3 Select *Duplicate* from the StaticText control's popup menu.**

PowerBuilder creates a duplicate of the selected control.

- 4 Select *Duplicate* from the StaticText control's popup menu again.**

PowerBuilder creates another duplicate.

You now have three StaticText controls arranged vertically at the top of the window.

Specify properties of the StaticText controls

Now you will specify the properties of the StaticText controls to define how they should appear on the logon window.

- 1 Select the first StaticText control you added.
Select *Properties* from the control's popup menu.**

The StaticText property sheet displays with the General tab selected.

- 2 Select the text *st_1* in the *Name* box.
Type *st_welcome* in the *Name* box.
Select the text *none* in the *Text* box and type *Welcome to SportsWear, Inc.*
Click *OK*.**

This provides text for the StaticText control and gives it a name. The prefix *st_* is standard for StaticText controls.

You return to the Window painter workspace.

- 3 In the StyleBar at the top of the Window painter workspace, change the font size for this control to 18 points.**

You can select a different type face and font size if this one isn't available on your system.

PowerBuilder changes the size of the text you entered in the control.

If you don't see the StyleBar

If you do not see the StyleBar, select Window>Toolbars, click StyleBar in the Select Toolbar window, and click Show.

- 4 Adjust the size of the StaticText control to fit the text you entered.**

Drag the upper-right corner of the control toward the upper-right corner of the window. If necessary, adjust the size of the control until you see all of the text you entered.

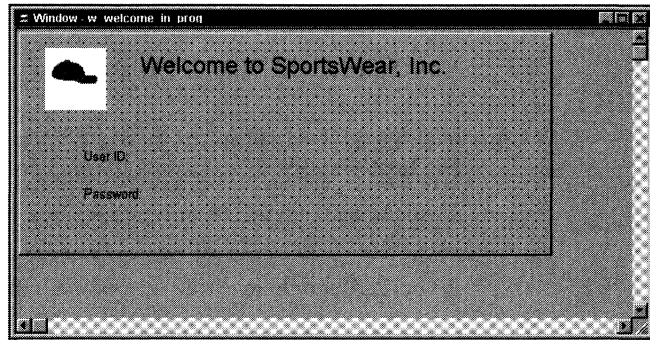
- 5 Adjust the location of the controls so that there is plenty of space between them.**

- 6 Display the property sheet for the second StaticText control you added.
Type *st_userid* in the *Name* box to change the control name.
Type *User ID:* in the *Text* box to change the text that will appear on the control.
Click *OK*.

You return to the Window painter workspace.

- 7 Display the property sheet for the third StaticText control you added.
Type *st_password* in the *Name* box to change the control name.
Type *Password:* in the *Text* box to change the text that will appear on the control.
Click *OK*.

You return to the Window painter workspace.



Add SingleLineEdit controls

Now you will add two SingleLineEdit controls to the window to allow the user to enter a user ID and password for connecting to the database. A SingleLineEdit control is a box in which the user can enter a single line of text. SingleLineEdits are typically used for input and output of data.



1 Click the *SingleEdit* button in the PainterBar.

The SingleEdit button is on the dropdown toolbar next to the fourth button in the PainterBar.

2 Click to the right of the User ID: StaticText control.

A SingleLineEdit control displays at the selected location.

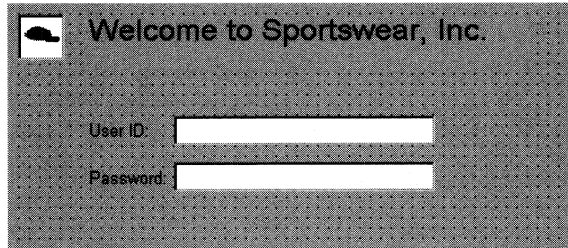
3 Select *Duplicate* from the SingleLineEdit control's popup menu.

PowerBuilder creates a duplicate of the selected control.

You now have two SingleLineEdit controls arranged vertically in the window.

4 Adjust the location of the controls as needed so that they are positioned next to the StaticText controls you added earlier.

5 Adjust the size of the SingleLineEdit controls to make them wider.



Specify properties of the SingleLineEdit controls

Now you will define the properties of the SingleLineEdit controls you just added to the logon window.

- 1 Display the property sheet for the top SingleLineEdit control. Type *sle_userid* in the *Name* box to specify the control name. Click *OK*.**

The prefix *sle_* is standard for SingleLineEdit controls.

- 2 Display the property sheet for the bottom SingleLineEdit control. Type *sle_password* in the *Name* box to specify the control name. Select the *Password* checkbox. Click *OK*.**

Because you checked the Password checkbox, the password the user types will display as a string of asterisks.

Add CommandButton controls

Now you will add CommandButton controls. Later you will define scripts that will execute when users click these buttons.



- 1 **Click the *Button* button in the PainterBar.**
- 2 **Click to the right of the *sle_userid* SingleLineEdit control.**
A CommandButton control displays at the selected location.
- 3 **Select *Duplicate* from the *CommandButton* control's popup menu.**
PowerBuilder creates a duplicate of the selected control.
- 4 **Adjust the location of the controls as needed so that there is some space between them.**

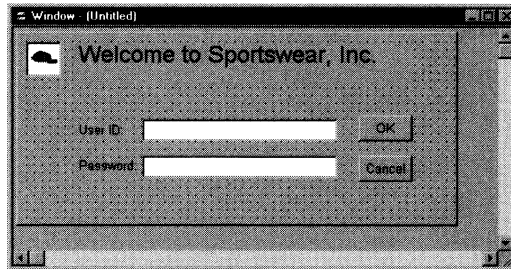
Specify properties of the CommandButton controls

Now you will define the properties of the CommandButton controls.

- 1 Display the property sheet for the top CommandButton control.**
Type *cb_ok* in the *Name* box to change the name of the control.
Type *OK* in the *Text* box to change the text that will appear on the control.
Select the *Default* checkbox.
Click *OK*.

This provides text for the CommandButton control and names the control. The prefix *cb_* is standard for CommandButton controls. Selecting the *Default* checkbox makes this the default button in the window.

- 2 Display the property sheet for the bottom CommandButton control.**
Type *cb_cancel* in the *Name* box to change the name of the control.
Type *Cancel* in the *Text* box to change the text that will appear on the control.
Select the *Cancel* checkbox.
Click *OK*.



Change the tab order on the window

Where you are

- Create a new window
 - Add controls to the window
 - > Change the tab order on the window
 - Save the window
 - Preview the window
 - Write the script to open the window
-

When you place controls in a window, PowerBuilder assigns them a default **tab order**. The tab order determines the sequence in which focus moves from control to control when the user presses the TAB key.

Now you will change the tab order for the window you created.

1 Select *Design>Tab Order* from the menu bar.

PowerBuilder displays the default tab order. The number at the right of each control indicates the control's relative position in the tab order. Controls that have the number zero are skipped when the user tabs in the window.

2 Change the tab values as needed.

Define the tab values as follows:

Set this control	To
SingleLineEdit control for entering a user ID (sle_userid)	10
SingleLineEdit control for entering a password (sle_password)	20
CommandButton control for the OK button (cb_ok)	30
CommandButton Control for the Cancel button (cb_cancel)	40

Changing a tab value

To change a tab value, select the value you want to change and enter a new value.

3 Deselect *Design>Tab Order* from the menu bar.

Save the window

Where you are

- Create a new window
- Add controls to the window
- Change the tab order on the window
- > Save the window
- Preview the window
- Write the script to open the window

Whenever you leave a painter, you are prompted to save any changes you have made. But as you refine the PowerBuilder objects, you should save them periodically to make sure you don't lose any work.

Now you will name the new window and save it.

1 Select *File>Save* from the menu bar.

The Save Window dialog box displays so you can name and save the window. Make sure pbtutor.pbl is selected in the Application Libraries box.

2 Type *w_welcome* for the window name.

The prefix *w_* is standard for windows.

3 (Optional) Enter comments in the *Comments* box.

This documents the window.

4 Click *OK*.

PowerBuilder saves the new window.

Preview the window

Where you are

Create a new window

Add controls to the window

Change the tab order on the window

Save the window

> Preview the window

Write the script to open the window

Now you will see the window as it will appear to users.

1 Select *Design>Preview* from the menu bar.

The window displays.

If you don't like the window layout, you can go back to the Window painter workspace and change the size, location, and fonts of the window controls.

2 Close the window.

You return to the Window painter workspace.

Write the script to open the window

Where you are

- Create a new window
 - Add controls to the window
 - Change the tab order on the window
 - Save the window
 - Preview the window
 - > Write the script to open the window
-

Now you will add a one-line script to open the logon window as soon as the application starts executing. You will add this script to the Open event of the MDI frame window called `w_genapp_frame`, which was created for you when you generated the application template.

To do this, you will:

- ◆ Modify a script in the PowerScript painter
- ◆ Compile the script

Modify a script in the PowerScript painter

Now you will modify a script to specify that the application open the logon window.

- 1 **Select *File>Open* from the menu bar.**
Select *w_genapp_frame* in the Windows box.

- 2 **Click *OK*.**

The Window painter workspace opens.

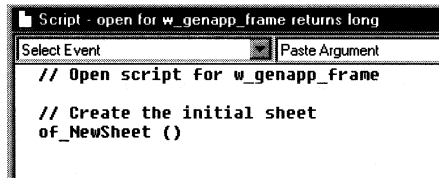
- 3 **Select *Script* from the window's popup menu.**

The PowerScript painter displays. It opens to the script for the *w_genapp_frame* window's Open event.

- 4 **Make sure you are looking at the script for the Open event for the window.**

The title bar of the PowerScript painter should read:

Script - open for w_genapp_frame returns long



If the title is incorrect

If the title is different, it means this is not the Open event. Click the Select Event dropdown listbox and select Open.

This event already has a script associated with it that PowerBuilder created for you when you generated the application template. The generated script causes a new MDI sheet to be displayed in the MDI frame automatically at application startup time. Next you are going to modify this script to cause the logon window to display instead.

About the PowerScript painter

The PowerScript painter is a text editor. It includes a PainterBar that has buttons that allow you to perform many common text-editing activities. For example, you can use these buttons to edit, comment and uncomment lines in scripts, and undo changes you've made.

- 5 Click anywhere in the line that reads:

```
of_NewSheet ()
```



- 6 Click the *Commnt* button in the PainterBar.

PowerBuilder converts the line to a comment.

```
//of_NewSheet ()
```

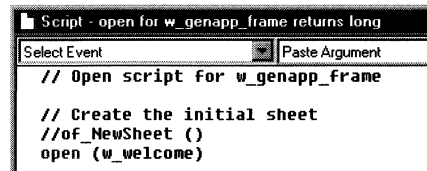
Using Script comments

A comment in PowerBuilder is indicated by either two forward slashes (//) at the start of a single-line comment, *or* a slash and an asterisk (/*) at the start and an asterisk and a slash (*/) at the end of a single-line or multiline comment.

- 7 Press ENTER at the end of the last line of the script to add a new line.
- 8 Type this line at the end of the script:

```
open(w_welcome)
```

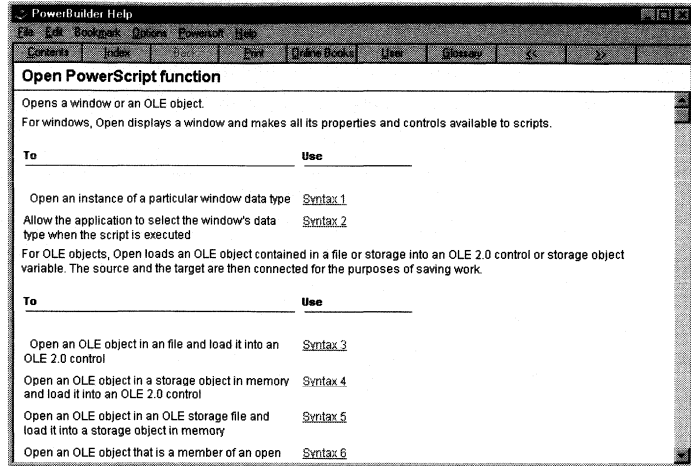
This calls the Open function to display the logon window you created.



As you type text in the PowerScript painter, you may see that PowerBuilder changes the text colors. The color coding PowerBuilder uses indicates what kind of syntax element you have entered. For example, PowerBuilder uses different colors to represent SQL keywords, variables, and comments. This allows you to tell at a glance whether PowerBuilder recognizes what you have typed.

- 9 Select the word *open* in the script and press SHIFT+F1. Select *Open PowerScript* function and click *Display*.

PowerBuilder displays Help for the Open function.



Accessing context-sensitive Help

To access context-sensitive Help for a function or reserved word, select the function or reserved word and press SHIFT+F1.

- 10 Close the Help window.

Compile the script



- 1 **Click the *Return* button in the PainterBar**
or
Select *File>Return* from the menu bar.

The script compiles and you return to the Window painter workspace. PowerBuilder automatically compiles the script when you close the PowerScript painter.

Handling errors in scripts

When there is an error in a script, an error window displays at the bottom of the PowerScript painter with the line number of the error and the error message.

To find an error Click on an error message to move the cursor to the line that contains that error. After you correct the error, click the Return button again.

Commenting out errors PowerBuilder does not save a script that has errors. If you want to save a script that has errors, select all the script and click the Comment button. You can come back later, uncomment the code, and fix the problem.

- 2 **Select *File>Save* from the menu bar.**

PowerBuilder saves the window.

- 3 **Select *File>Close* from the menu bar to close the window.**

Connecting to the Database

This chapter shows you how to make the application connect to the Powersoft Demo Database at execution time and how to use the Database painter to look at the table definitions and database profile for the Powersoft Demo Database.

In this chapter you will:

- ◆ Learn some terms and concepts that apply to database connectivity
- ◆ Look at the Powersoft Demo Database
- ◆ Write scripts to establish the execution time connection
- ◆ Run the application

How long will it take?

About 20 minutes.

Look at the Powersoft Demo Database

Where you are

- > Look at the Powersoft Demo Database
 - Establish the execution time connection
 - Run the application
-

In many organizations, database specialists maintain the database. If this is true in your organization, you may not need to create and maintain tables within the database. But to take full advantage of PowerBuilder, you may still want to know how to work with databases.

To maintain database definitions with PowerBuilder, you do most of your work using the Database and Table painters. The Database and Table painters allow you to:

- ◆ Create, alter, and drop tables
- ◆ Create, alter, and drop primary and foreign keys
- ◆ Create and drop indexes
- ◆ Define and modify extended attributes for columns
- ◆ Drop views

Now you will:

- ◆ Look at the table definitions in the Powersoft Demo Database
- ◆ Look at the database profile for the Powersoft Demo Database

Look at the table definitions in the Powersoft Demo Database

The tables, columns, indexes, and keys required to run the Powersoft Demo Database have been defined for you. You do not need to make any changes to the database to complete the tutorial.

Now you will look at the definitions for the Customer and Product tables in the Powersoft Demo Database. This will help you become familiar with the Database painter and with the tables you will be using in the tutorial.



1 Click the *Database* button in the PowerBar.

PowerBuilder connects to the database.

What happens when you connect

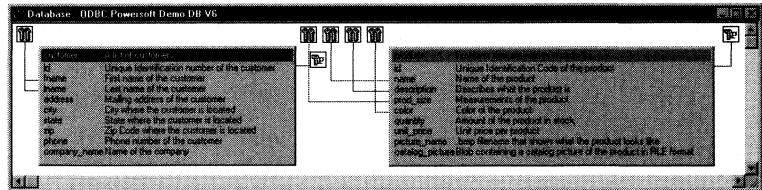
When you connect to a database in the development environment, PowerBuilder writes the connection parameters to the [Database] section of the PB.INI file. Each time you connect to a different database, PowerBuilder overwrites the existing parameters in the [Database] section with those for the new database connection.

When you open a PowerBuilder painter that accesses the database, you are automatically connected to the database you used last. PowerBuilder determines which database this is by reading the [Database] section of the PB.INI file.

The Select Tables dialog box displays, listing the tables in the database you are currently connected to.

- 2 In the **Select Tables** dialog box, click *customer*.
Scroll until you see *product*.
Click *product*.
Click *Open*.

The **Select Tables** dialog box closes and you go to the Database painter. The workspace shows the two tables you selected, and the workspace title bar identifies the database you are currently connected to.



In the Database painter workspace, you should see several icons that look like keys. These represent the keys and indexes that were defined for the tables.

- 3 Position the mouse cursor on the title bar for the **Customer** table. Right-click to display the popup menu. Select **Alter Table** from the popup menu.

PowerBuilder displays the Table painter's **Alter Table** window.

In the top part of the window you see the column definitions for the table. In the bottom part of the window you see the extended attributes. If you were going to create new tables for use with the application, you would use a window like this to define the columns and extended attributes.

About extended attributes

PowerBuilder stores extended attribute information in the database in a collection of system tables called the **repository**. Extended attributes include headers and labels for columns, initial values for columns, validation rules, and display formats.

- 4 Scroll through the list of columns for the *Customer* table.

Depending on which display resolution you are using, you may not need to scroll to see all of the columns. Later in the tutorial you will create a window that displays customer data.

- 5 Close the **Alter Table** window.

Look at the database profile for the Powersoft Demo Database

When database connections occur PowerBuilder establishes a connection to the database at various times, depending on whether you are developing or executing an application. PowerBuilder connects to the database when you open a painter that accesses the database, compile or save a PowerBuilder script that contains embedded SQL statements, or run a PowerBuilder application that accesses the database.

When you start working with PowerBuilder immediately after installing the software, you are automatically connected to the Powersoft Demo Database (unless you did not install this database).

Using database profiles to connect When you're developing an application in PowerBuilder, you can connect to a database by responding to dialog boxes that prompt you to supply connection information, or you can use a database profile. A **database profile** is a named set of parameters that specifies a connection to a particular data source or database. Database profiles provide an easy way for you to manage database connections that you use frequently. PowerBuilder stores database profile parameters in its initialization file (PB.INI).

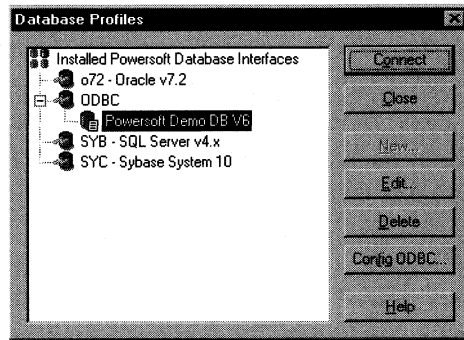
When you define an ODBC data source in PowerBuilder, PowerBuilder creates a database profile for the data source and stores the name of the associated data source. The definitions of ODBC data sources are stored in the Powersoft ODBC.INI file.

In addition, when you use PowerBuilder to create a new SQL Anywhere database, PowerBuilder creates both the data source definition and the database profile.

The Powersoft Demo Database is a SQL Anywhere database that is accessed through ODBC. To use this database, you do not need to define an ODBC data source or database profile; these definitions have been created for you. However, while you're in the Database painter, you should take the opportunity to look at the database profile for the Powersoft Demo Database. This will give you an idea what a database profile looks like.

1 Select *File>Connect>Setup* from the menu bar.

PowerBuilder displays the Database Profiles dialog box, which includes a tree view of the installed database interfaces and defined database profiles for each interface.

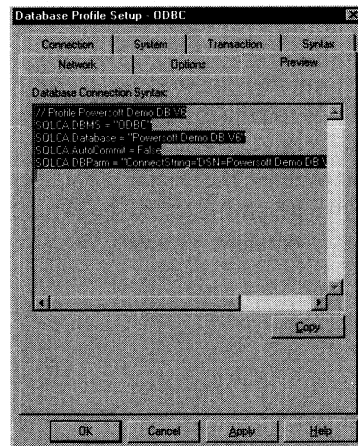


**2 Select *Powersoft Demo DB V6*.
Click *Edit*.**

PowerBuilder displays the Database Profile Setup dialog box with the Connection tab selected.

3 Select the *Preview* tab.

The PowerScript connection syntax for each selected option is shown on the Preview tab. If you change any options, the syntax changes.



- 4 **Click *Cancel* to close the Database Profile Setup dialog box. Click *Close*.**

You return to the Database painter workspace.

- 5 **Close the Database painter.**

Establish the execution time connection

Where you are

- Look at the Powersoft Demo Database
 - > Establish the execution time connection
 - Run the application
-

Establishing a connection To make it possible for an application to connect to the database at execution time, you must establish the connection in an application script. To do this, you use a **Transaction object**, a nonvisual object that serves as a communication area between the application and the database. Each Transaction object has properties (variables) for sending connection information to the database and for obtaining information from the database about the success or failure of database activities.

If the application connects to only one database, you specify connection parameters as properties of the default Transaction object, SQL Communications Area (SQLCA). The SQLCA object has several properties, including database name, login ID, and password. If an application communicates with multiple databases, you can create additional Transaction objects as needed, one for each database connection.

You will set some of the SQLCA properties.

Now you will:

- ◆ Modify the script for the application Open event
- ◆ Add a window function
- ◆ Write scripts for the buttons

As is often the case when you are developing production applications, you will get some of the connection parameters from an initialization file and some from user input.

Modify the script for the application Open event

The script for the application Open event reads database connection parameters from the PB.INI file. You need to modify this script, because you do not want to read all of the parameters from PB.INI. In particular, you do not want to get the login ID and password or some of the database information from PB.INI. You will be getting the login ID and password directly from the user in the logon window; you will provide the database information in a script.

Using an application-specific file instead of PB.INI

The connection parameters in the [Database] section of the PB.INI file match the database you connected to last in PowerBuilder. When you develop and test the pbtutor application, the application will use the connection parameters currently in your PB.INI.

When you are developing a production application and want to read some of your connection parameters from a file, you create an application-specific file, such as MYAPP.INI. Then you use the ProfileString function to read information from the application-specific file instead of the PB.INI file. You will set up an application initialization file later in the tutorial.

FOR INFO For more information about the ProfileString function, see the *PowerScript Reference*.



- 1 Click the *Appl* button in the PowerBar.



- 2 Click the *Script* button in the PainterBar.

The PowerScript painter displays the script for the Open event for the application.

PowerBuilder created this script when you generated the application template in Chapter 3, "Starting PowerBuilder". The script first determines the name of the startup file, which depends on the operating system. Then the ProfileString function looks in the [Database] section of the startup file to get database connection information.

- 3 Scroll down to the bottom of the script to see the ProfileString function calls.

- 4 Select these lines:

```
sqlca.database = ProfileString (ls_startupfile,
                              "database", "database", "")
```

```
sqlca.userid = ProfileString (ls_startupfile,
    "database", "userid", "")
sqlca.dbpass = ProfileString (ls_startupfile,
    "database", "dbpass", "")
sqlca.logid = ProfileString (ls_startupfile,
    "database", "logid", "")
sqlca.logpass = ProfileString (ls_startupfile,
    "database", "LogPassWord", "")
sqlca.servername = ProfileString (ls_startupfile,
    "database", "servername", "")
sqlca.dbparm = ProfileString (ls_startupfile,
    "database", "dbparm", "")
```

Make sure that you have *not* selected the line:

```
slqca.DMBS = ProfileString(startupfile,
    "database", "dbms", "")
```



5 Click the *Commnt* button in the PainterBar.

PowerBuilder comments out the lines.

6 Notice that the following lines are already commented out:

```
//connect;
//
//if sqlca.sqlcode <> 0 then
// MessageBox ("Cannot Connect to Database",
//     sqlca.sqlerrtext)
// return
//end if
```

Leave these lines as they are. You will be performing this logic in the logon window.

**7 Click the *Select Event* listbox.
Select *close*.**

When you select the script for the Close event, PowerBuilder compiles the script for the Open event.

8 Type this line:

`Disconnect using SQLCA;`

This line disconnects from the database when the user closes the application.



9 Click the *Return* button in the PainterBar.

or

Select *File>Return* from the menu bar.

PowerBuilder compiles the script. If there are errors, correct them and select Return again.

10 Select *File>Save* from the menu bar.

PowerBuilder saves the updated Application object.

11 Close the Application painter.

Add a window function

PowerBuilder gives you the ability to create functions. This way you can centralize processing logic that's required in multiple situations, organize code so that it's easier to maintain, or simply break a long body of code into more manageable subroutines.

Now you will create a function for the `w_welcome` window to perform the logic required to connect to the database. To define the function, you will specify:

- ◆ The arguments it requires
- ◆ The value it is to return
- ◆ A script of the code it is to execute

**1 Open the Window painter.
Open the `w_welcome` window.**

2 Select *Declare>Window Functions* from the menu bar.

PowerBuilder displays the Select Function In Window dialog box.

3 Click *New*.

PowerBuilder displays the New Function dialog box.

4 Type `of_connect` in the *Name* box.

The default values for the Access and Returns boxes are what you want. The function will return a value of type integer, and access to the function will not be restricted.

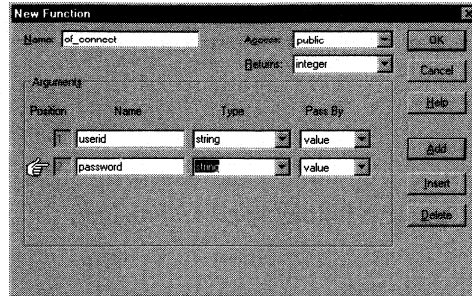
You do not need to change the Access and Returns boxes.

**5 Type `userid` in the *Name* box inside the *Arguments* groupbox.
Select *string* in the *Type* dropdown listbox.**

You do not need to change the Pass By box.

- 6 Click **Add** to add a second argument. Type *password* in the **Name** box inside the **Arguments** groupbox. Select *string* in the **Type** dropdown listbox.

You do not need to change the Pass By box.



- 7 Click **OK**.

PowerBuilder takes you to the PowerScript painter for the window function.

- 8 Type this line in the **PowerScript** painter:

```
string ls_userid, ls_password, ls_database
```

This line declares the local variables (*ls_userid*, *ls_password*, and *ls_database*), all of which have the string data type.

Naming conventions for variables

To make scripts easier to read and understand, you should adopt a standard for naming variables. One standard that works well is to give each variable a 2-letter prefix followed by an underscore (`_`). The first letter identifies the scope of the variable (for example, *g* for global, *l* for local) and the second letter identifies the data type (for example, *s* for string, *l* for long).

- 9 Type these lines:

```
ls_userid = Trim(userid)
ls_password = Trim(password)
```

```
ls_database = "ConnectionString='DSN=Powersoft Demo  
DB V6;'"
```

The first two lines use the Trim function to remove leading and trailing spaces from the user ID and password values passed as arguments to the function. The resulting values are assigned to the two local variables ls_userid and ls_password.

The third line assigns a literal string to the variable ls_database. This string specifies the name of the ODBC data source for the Powersoft demo database.

10 Type the following all on one line:

```
sqlca.DbParm=ls_database + "UID=" + ls_userid +  
";PWD=" + ls_password + "''
```

This line assigns the completed connect string to the DbParm property of SQLCA. The connect string specifies the ODBC data source name, user ID, and password for the connection.

11 Type these lines:

```
CONNECT using sqlca;  
Return sqlca.SQLCode
```

The first line uses the SQLCA Transaction object to connect to the database. The last line returns the value of the SQLCode property of the Transaction object to the script that calls the function. The SQLCode property indicates whether the SQL operation succeeded.



**12 Click the *Return* button in the PainterBar.
or
Select *File>Return* from the menu bar.**

PowerBuilder compiles the script and returns you to the Window painter workspace.

If there are errors, an error window opens at the bottom of the screen. Correct the errors, then select Return again.

Write scripts for the buttons

Now you will write the scripts for the buttons on the logon window.

- 1 **Select Script from the `cb_ok` `CommandButton` control's popup menu.**

PowerBuilder displays the script for the Clicked event for the `cb_ok` control.

- 2 **Make sure you are looking at the script for the Clicked event for the `cb_ok` control.**

The title bar of the PowerScript painter should read:

Script - clicked for `cb_ok` returns long

If the title is incorrect

If the object is not `cb_ok`, select Design>Select Object from the menu bar and select `cb_ok`.

If the event is not the Clicked event, click the Select Event listbox and select Clicked.

- 3 **Type this line:**

```
SetPointer(hourglass!)
```

This line uses the `SetPointer` function to set the mouse pointer to the shape of a hourglass when the user clicks OK on the logon window.

- 4 **Press ENTER and type these lines exactly as shown:**

```
IF parent.of_connect(sle_userid.text, &
    sle_password.text) = -1 THEN
    MessageBox ("Database Connection Error", &
        "Unable to connect.")
    HALT
ELSE
    close (parent)
END IF
```

These lines connect to the database by calling the `of_connect` function, passing the user ID and password values entered by the user as arguments.

After calling the `of_connect` function, the script checks to see whether the connection was successful by checking the function's return value. If the connection fails, a message box is displayed and the application is terminated. If the connection is successful, the script closes the logon window.

The pronoun `Parent` is used to make a general reference to the object with which the `cb_ok` control is associated. In this case, that object is the `w_welcome` window.



- 5 Click the *Return* button in the PainterBar.**
or
Select *File>Return* from the menu bar.

PowerBuilder compiles the script and returns you to the Window painter workspace.

If there are errors, an error window opens at the bottom of the screen. Correct the errors, then select `Return` again.

- 6 Select *Script* from the *cb_cancel* CommandButton control's popup menu.**

PowerBuilder displays the script for the `Clicked` event for the `cb_cancel` control.

- 7 Type this one-line script:**

```
HALT
```

This statement terminates the application immediately when the user clicks `Cancel` on the logon window.



- 8 Click the *Return* button in the PainterBar.**
or
Select *File>Return* from the menu bar.

PowerBuilder compiles the script and returns you to the Window painter workspace.

If there are errors, an error window opens at the bottom of the screen. Correct the errors, then select `Return` again.

Run the application

Where you are

Look at the Powersoft Demo Database

Establish the execution time connection

> Run the application

Now you will run the application.



1 Click the *Run* button in the PowerBar.

2 Click *Yes to save your changes*.

The logon window displays.

**3 Type *dba* in the *User ID* box.
Type *sql* in the *Password* box.**

4 Click *OK*.

The database connection is established and the MDI frame for your application displays.

5 Select *File>Exit* from the menu bar.

The application terminates and you return to the Window painter.

6 Select *File>Close* to close the Window painter.

Setting Up the Menus

In this chapter you will set up the menus for the application. You will:

- ◆ Modify the menu that was created for the MDI frame window in the application template
- ◆ Create a new menu, which you will attach to the MDI sheets in the next chapter
- ◆ Preview the new menu

Menus are separate objects that you create using the Menu painter. After you create a menu, you can attach it to as many windows as you want. You can create menus at any time during the application development process.

How long will it take?

About 30 minutes.

Open the menu for the MDI frame

Where you are

- > Open the menu for the MDI frame
 - Add items to the File dropdown menu
 - Preview the menu
 - Create a new menu
 - Add items to the new menu
 - Add toolbar buttons for the new menu items
 - Save the new menu
 - Preview the new menu
-

Now you will open the menu that was created for you in the application template.



- 1 **Click the *Menu* button in the PowerBar.**

The Select Menu dialog box displays.

- 2 **Select *m_genapp_frame*.
Click *OK*.**

The Menu painter displays the menu associated with the MDI frame window in the application.

This menu was created automatically when you generated the application template. The *m_genapp_frame* menu is the ancestor of all the other menus you work with in the tutorial. Therefore, the changes you will make to this menu will automatically be propagated to the descendants of *m_genapp_frame*.

Add items to the File dropdown menu

Where you are

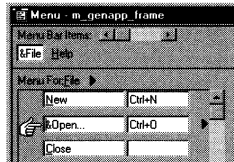
- Open the menu for the MDI frame
 - > Add items to the File dropdown menu
 - Preview the menu
 - Create a new menu
 - Add items to the new menu
 - Add toolbar buttons for the new menu items
 - Save the new menu
 - Preview the new menu
-

Now you will modify the File dropdown menu on the MDI frame menu.

In the Menu painter, the menus you create look similar to the way they will appear when the application is running. You put the menu bar items across the top of the painter, and you add items for each dropdown menu in a list underneath.

1 Click *Open* in the Menu painter workspace.

The insertion point moves to the Open menu item under Menu For:File. PowerBuilder displays an ampersand character (&) in front of the O.



The ampersand character defines accelerator keys

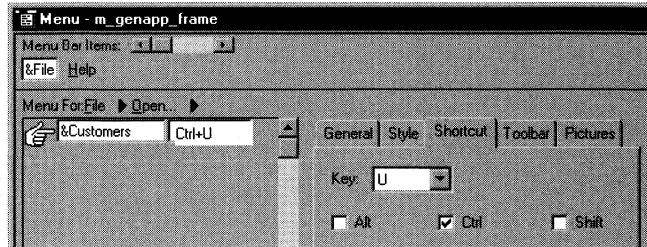
The ampersand (&) is the standard for defining accelerator keys, which provide mnemonic access to menus and window controls.

In the tutorial application, for example, the user can access the Open item on the File menu by pressing ALT+F+O.



2 Click the *Next Lvl* button in the PainterBar.

- 3 **Type *&Customers* at the cursor.**
Select the *Shortcut* tab on the right side of the screen.
Type *U* in the *Key* field.
Select the *Ctrl* checkbox.



- 4 **Press TAB.**

This establishes the Customers menu item under the Open submenu. C is the accelerator key and CTRL+U is the shortcut key. If you select the General tab, you will see that PowerBuilder set m_customers as the menu name (you use this name to reference the menu item in scripts). The insertion point moves to the menu item under Customers.

- 5 **Type *&Products* at the cursor.**
Select the *Shortcut* tab if it's not already selected.
Type *R* in the *Key* field.
Select the *Ctrl* checkbox.

- 6 **Press TAB.**

This establishes the Products menu item under the Open submenu. P is the accelerator key, CTRL+R is the shortcut key, and m_products is the menu name. The insertion point moves to the menu item under Products.



- 7 **Click the *Prior Lvl* button in the PainterBar.**
Select the *Style* tab.
Select the *Enabled* checkbox.

This enables the Open menu item at execution time so the user can select it.

- 8 **Select *File>Save* from the menu bar.**

PowerBuilder saves the m_genapp_frame menu.

Preview the menu

Where you are

- Open the menu for the MDI frame
 - Add items to the File dropdown menu
 - > Preview the menu
 - Create a new menu
 - Add items to the new menu
 - Add toolbar buttons for the new menu items
 - Save the new menu
 - Preview the new menu
-

Now you will preview the menu to see what it will look like at execution time.

1 Select *Design>Preview* from the menu bar.

The menu displays in a new window called `m_genapp_frame`.

2 Navigate through the menu using the mouse or keyboard.

Select `File>Open` and check that the Customers and Products items are on the submenu.

3 Close the window to return to the Menu painter workspace.

4 Close the `m_genapp_frame` menu.

If you are prompted to save changes, select Yes.

Create a new menu

Where you are

- Open the menu for the MDI frame
 - Add items to the File dropdown menu
 - Preview the menu
 - > Create a new menu
 - Add items to the new menu
 - Add toolbar buttons for the new menu items
 - Save the new menu
 - Preview the new menu
-

Now you will create a new menu that will display whenever the user opens an MDI sheet to look at customer or product information. The menu you create is a descendant of the `m_genapp_sheet` menu, created for you when you generated the application template. It has some of the characteristics of `m_genapp_sheet`, so you want to inherit these characteristics. But you need to add some more menu items in the Menu painter.



1 Click the *Menu* button in the PowerBar.

The Select Menu dialog box displays.

2 Click *Inherit* to create a new menu that is a descendant of another menu.

PowerBuilder displays the Inherit From Menu dialog box.

**3 Select *m_genapp_sheet*.
Click *OK*.**

PowerBuilder displays an untitled menu that has all of the characteristics of `m_genapp_sheet`.

In addition to the File and Help dropdown menus, the new menu has the Edit and Window menus, which provide access to many of the activities you would typically perform on an open MDI sheet. For example, these menus have options for cutting and pasting text and for tiling and cascading windows.

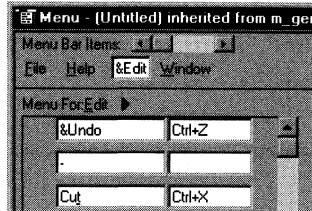
Add items to the new menu

Where you are

- Open the menu for the MDI frame
 - Add items to the File dropdown menu
 - Preview the menu
 - Create a new menu
 - > Add items to the new menu
 - Add toolbar buttons for the new menu items
 - Save the new menu
 - Preview the new menu
-

Now you will add items to the new menu.

- 1 **Select the *Edit* dropdown menu in the Menu painter workspace.**



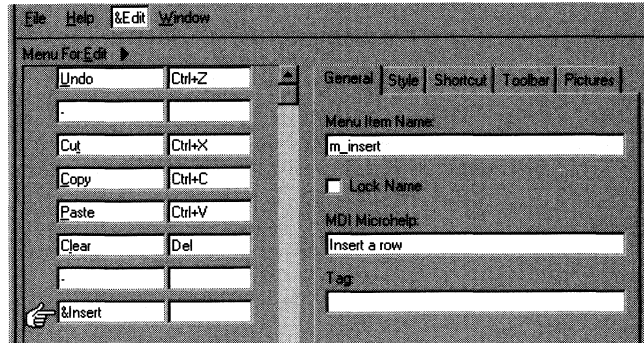
The insertion point moves to the Edit menu.

- 2 **Press TAB until the insertion point is in the empty menu item under *Clear*.**
- 3 **Type – (hyphen).**
Press TAB.

If PowerBuilder displays a message that the default name conflicts with one of the other menu names, click OK to accept the suggested name.

The hyphen creates a separator line that spans the width of the menu. You will see this line later when you preview the menu.

- 4 **Type *&Insert* in the last menu item (under the hyphen).**
Type *Insert a row* in the MDI Microhelp box.
Select the *Insert* menu item in the left panel.



The menu item name is set automatically to m_insert. The text *Insert a row* will display in the MicroHelp line at the bottom of the application window whenever the user selects the menu item.

- 5 **Press TAB.**

This establishes the Insert menu item.

- 6 **Type *&Update* at the cursor.**
Type *Update the database* in the MDI Microhelp box.
Select the *Update* menu item.
Press TAB.

This establishes an Update menu item. The menu item name is m_update and the MicroHelp text is *Update the database*.

- 7 **Type *&Delete* at the cursor.**
Type *Delete the current row* in the MDI Microhelp box.
Select the *Delete* menu item.
Press TAB.

This establishes a Delete menu item. The menu item name is m_delete and the MicroHelp text is *Delete the current row*.

Add toolbar buttons for the new menu items

Where you are

- Open the menu for the MDI frame
- Add items to the File dropdown menu
- Preview the menu
- Create a new menu
- Add items to the new menu
- > Add toolbar buttons for the new menu items
- Save the new menu
- Preview the new menu

Now you will add toolbar buttons for the menu items you just defined.

1 Select *Insert* in the Menu painter workspace.

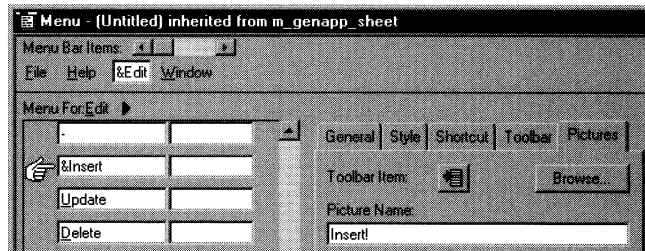
The insertion point moves to the Insert menu item.

2 Select the *Toolbar* tab.

Type *Insert* in the *Text* box.

Select the *Pictures* tab.

Type *Insert!* in the *Picture Name* box.



This defines a toolbar button for the Insert menu item that uses the stock picture called Insert!. When the Show Text option is enabled for toolbars, the text Insert appears on the button.

3 Select *Update* in the Menu painter workspace.

The insertion point moves to the Update menu item.

- 4 **Select the *Toolbar* tab.**
Type *Update* in the *Text* box.
Select the *Pictures* tab.
Type *Update!* in the *Picture Name* box.

This defines a toolbar button for the Update menu item that uses the stock picture called Update!. The button text is Update.

- 5 **Select *Delete* in the **Menu painter workspace.****

The insertion point moves to the Delete menu item.

- 6 **Select the *Toolbar* tab.**
Type *Delete* in the *Text* box.
Select the *Pictures* tab.
Type *Clear!* in the *Picture Name* box.

This defines a toolbar button for the Delete menu item that uses the stock picture called Clear!. The button text is Delete.

Save the new menu

Where you are

- Open the menu for the MDI frame
 - Add items to the File dropdown menu
 - Preview the menu
 - Create a new menu
 - Add items to the new menu
 - Add toolbar buttons for the new menu items
 - > Save the new menu
 - Preview the new menu
-

Now you will save the new menu.

1 Select *File>Save* from the menu bar.

The Save Menu dialog box opens. The insertion point is in the Menus box.

2 Type *m_fileopen* as the name of the new menu.

or

Click Browse and select *m_fileopen*.

Type a comment in the *Comments* box.

Click *OK*.

This names the menu. The prefix *m_* is standard for menus.

The name you just assigned to the menu displays in the title bar of the Menu painter workspace.

Preview the new menu

Where you are

- Open the menu for the MDI frame
- Add items to the File dropdown menu
- Preview the menu
- Create a new menu
- Add items to the new menu
- Add toolbar buttons for the new menu items
- Save the new menu
- > Preview the new menu

Now you will preview the new menu to see how it will appear at execution time.

1 Select *Design>Preview* from the menu bar.

The menu displays in a new window called `m_fileopen`.

2 Navigate through the menus using the mouse or keyboard.

Make sure the Insert, Update, and Delete items are on the Edit menu.

3 Close the window to return to the Menu painter workspace.

4 Close the *m_fileopen* menu.

Building an Ancestor Window

In this chapter you will:

- ◆ Create and save a new window
- ◆ Add DataWindow controls to the window
- ◆ Add user events to the window
- ◆ Write the scripts required to retrieve, insert, update, and delete data
- ◆ Write the scripts for the menu items to trigger the user events you created

How long will it take?

About 45 minutes.

Create a new window

Where you are

- > Create a new window
 - Add DataWindow controls
 - Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - Save the window
 - Add menu scripts to trigger the user events
-

Now you will create a window that handles retrieval and update operations against the database. The window is a master/detail window that allows you to display a master list of rows in a particular table and see detailed information for each row in the table. This window will become an ancestor to descendent windows you will create later.



- 1 Click the *Window* button in the PowerBar.**

The Select Window dialog box opens.



- 2 Click the *New* button.**

The Window painter workspace opens. The rectangle represents the window you are building.

- 3 Make the window rectangle larger.**

Add DataWindow controls

Where you are

- Create a new window
 - > Add DataWindow controls
 - Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - Save the window
 - Add menu scripts to trigger the user events
-

Now you will add two DataWindow controls to the new window that inherit their definitions from a **user object**. You can build a user object in PowerBuilder to perform processing that you use frequently in applications. Once you've defined a user object, you can reuse it as many times as you need without any additional work.

The user object you will use, `u_dwstandard`, was created for the tutorial application. It is a customized DataWindow control that includes scripts to perform standard database error checking. The user object is provided for you, so all you need to do to use it is place it in the window.

In this exercise, you will:

- ◆ Add a DataWindow control for the master DataWindow
- ◆ Add a DataWindow control for the detail DataWindow
- ◆ View the scripts inherited from the user object

Add a DataWindow control for the master DataWindow

Now you will add a DataWindow control that will be the master DataWindow in the application.



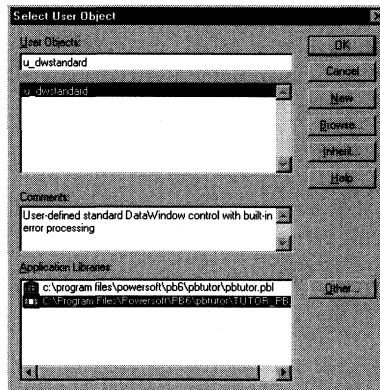
- 1 Click the *UserObj* button in the PainterBar.
or
Select *Controls>UserObject* from the menu bar.

Select the UserObj button in the PainterBar

Make sure you select the UserObj button in the PainterBar. If you select the button on the PowerBar you launch the User Object painter.

The Select User Object dialog box displays.

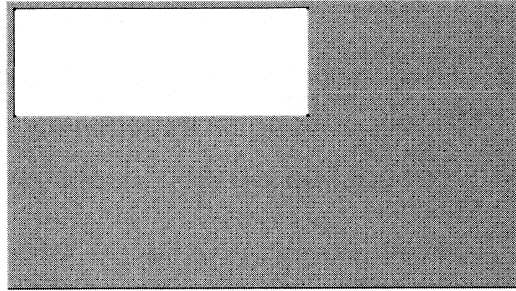
- 2 Select the **TUTOR_PB.PBL** file in the *Application Libraries* box.
Select *u_dwstandard* if it isn't already selected.



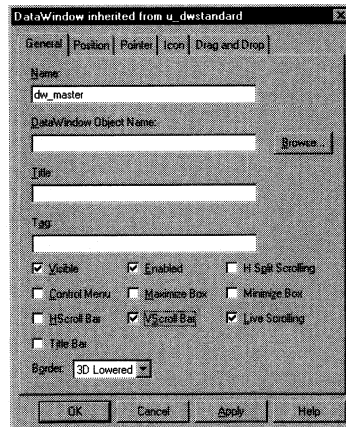
- 3 Click *OK*.
Click in the upper-left corner of the window to place the control.

PowerBuilder creates a descendent DataWindow control that inherits its definition from the user object.

If necessary, move the DataWindow control so that it is completely visible inside the window.



- 4 Select *Properties* from the DataWindow control's popup menu.
- 5 Select the text *dw_1* in the *Name* box.
Type *dw_master* in the *Name* box.
Select the *Vscroll Bar* checkbox.
Click *OK*.



PowerBuilder adds a vertical scrollbar to the control. It also changes its name to *dw_master*; the name change isn't visible in the control.

The prefix *dw_* is standard for DataWindow controls.

Add a DataWindow control for the detail DataWindow

Now you will add a second DataWindow control that will be the detail DataWindow in the application.



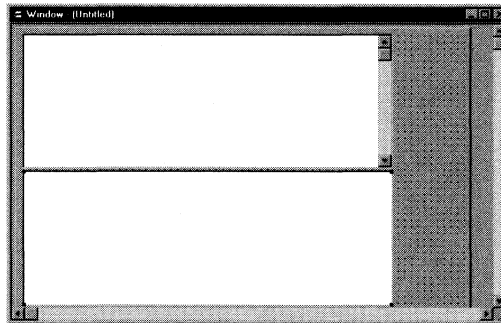
- 1 **Click the *UserObj* button in the PainterBar.**
or
Select *Controls>UserObject* from the menu bar.

- 2 **Select the *TUTOR_PB.PBL* file in the *Application Libraries* box if it's not already selected.**
Select *u_dwstandard* if it's not already selected.
Click *OK*.
Click in the left side of the window underneath the *dw_master* control.

PowerBuilder creates another DataWindow control that inherits its definition from the user object *u_dwstandard*.

- 3 **If possible, move the DataWindow control so that it is completely visible inside the window.**

If you need to, you can enlarge the window object to make more room.



- 4 **Select *Properties* from the DataWindow control's popup menu.**
- 5 **Select the text *dw_1* in the *Name* box.**
Type *dw_detail* in the *Name* box.
Click *OK*.

PowerBuilder changes the name of the control to *dw_detail*.

View the scripts inherited from the user object

Now you will view the scripts the DataWindow controls inherited from `u_dwstandard`.

- 1 **Select the `dw_detail` control if it's not already selected.**

- 2 **Select *Script* from the DataWindow control's popup menu.**

The PowerScript painter displays. It automatically opens to the script for the `dw_detail` control's `Itemchanged` event. This script is empty.

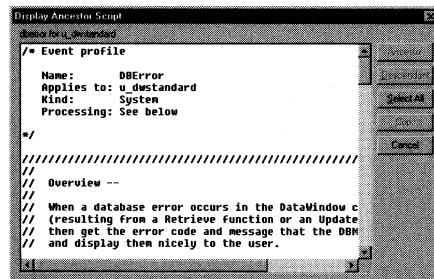
- 3 **Click the *Select Event* listbox.
Select `dberror`.**

This script is also empty.

You can look at the scripts defined for any event by selecting a listing in the *Select Event* listbox. A script icon next to an event name means the event has an associated script.

- 4 **Select *Design>Display Ancestor Script* from the menu bar.**

PowerBuilder displays the script for the `DBError` event in the Display Ancestor Script dialog box.



- 5 **Scroll through the window to view the database error-handling logic defined for the `DBError` event.**

The logic suppresses the default error message the `DBError` event normally displays and instead causes an appropriate message to be displayed for each database error that might occur. Because you used the `u_dwstandard` object to define both DataWindow controls in the window, this logic is automatically reused in both controls.

6 Click *Cancel*.

PowerBuilder closes the window and returns you to the PowerScript painter workspace.

7 Close the PowerScript painter.

You return to the Window painter workspace.

Add user events

Where you are

- Create a new window
 - Add DataWindow controls
 - > Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - Save the window
 - Add menu scripts to trigger the user events
-

Windows, user objects, and controls have predefined events associated with them. Most of the time, the predefined events are all you need. But there are times when you will want to declare your own events. Events that you define are called **user events**.

User events have many purposes. One reason to define a user event is to reduce coding in situations where an application provides several ways to perform a particular task. For example, suppose the user can update the database from a particular window by clicking a button, selecting a menu item, or simply closing the window. Instead of writing the code to update the database in each of these places, you could define a user event, then trigger that user event in each place you update the database.

Now you will define some user events to handle retrieval, insert, update, and delete operations against the database.

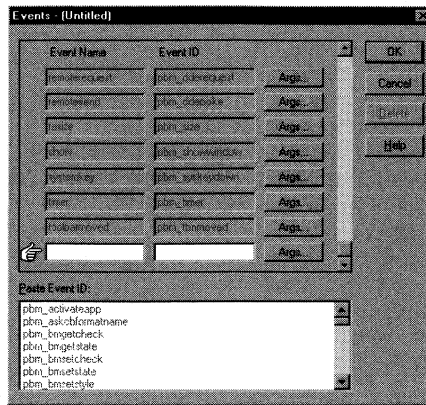
1 Make sure that neither of the controls in the window is selected.

To clear a selection

To ensure that nothing is selected, click the window background to select the window object.

- 2** Select *Declare>User Events* from the menu bar.

The Events dialog box displays.



- 3 Make sure the title bar of the dialog box reads:**

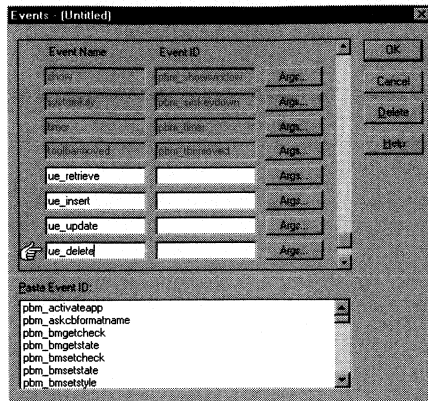
Events - (Untitled)

If the title is incorrect

If the title bar is incorrect, click the Cancel button and start this exercise over.

- 4 **Click in the empty Event Name box below** *toolbarmoved*.
Type *ue_retrieve* **in the Event Name box.**
Press the DOWN ARROW **key.**
- 5 **Type** *ue_insert*.
Press the DOWN ARROW **key.**
- 6 **Type** *ue_update*.
Press the DOWN ARROW **key.**

- 7 Type `ue_delete` in the *Event Name* box at the end of the list.



- 8 Click **OK**.

You return to the Window painter workspace.

Add scripts for the user events

Where you are

- Create a new window
 - Add DataWindow controls
 - Add user events
 - > Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - Save the window
 - Add menu scripts to trigger the user events
-

Now you will define scripts for the new user events.

- 1 Make sure that neither of the controls in the window is selected.**



- 2 Click the *Script* button in the PainterBar.**
or
Select *Script* from the window's popup menu.

- 3 Click the *Select Event* listbox and select *ue_retrieve*.
Type these lines:**

```
IF dw_master.Retrieve() <> -1 THEN
    dw_master.SetFocus()
    dw_master.SetRowFocusIndicator(Hand!)
END IF
```

These lines execute the Retrieve function, which performs the SQL Select statement associated with the DataWindow object and places the retrieved rows in the dw_master DataWindow control.

If the retrieval operation succeeds, it sets focus to the first row in the DataWindow control and establishes the hand pointer as the current row indicator. If the retrieval fails, PowerBuilder automatically triggers the DBError event and the script associated with that event is executed. In the tutorial application, the logic for the DBError event is handled in the user object u_dwstandard.

- 4 Click the *Select Event* listbox and select *ue_insert*.**

The script compiles and PowerBuilder opens the script for ue_insert. The script is empty.

5 Type these lines:

```
dw_detail.Reset()  
dw_detail.InsertRow(0)  
dw_detail.SetFocus()
```

The first line clears (resets) the dw_detail DataWindow control. The second line inserts a new row after the last row. The third line positions the cursor in the dw_detail control.

The InsertRow function does not insert a new row in the database. The row is not actually added to the database until the Update function is executed.

6 Click the *Select Event* listbox and select *ue_update*.

The script compiles and PowerBuilder opens the script for ue_update. The script is empty.

7 Type these lines:

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

These lines update the database with all the changes made to the dw_detail DataWindow.

If the update succeeds, a database commit is performed and a message tells you the operation succeeded. If the update fails, a database rollback is performed and PowerBuilder automatically triggers the DBError event. The script associated with that event is executed, causing PowerBuilder to display a message that the operation failed.

8 Click the *Select Event* listbox and select *ue_delete*.

The script compiles and PowerBuilder opens the script for ue_delete. The script is empty.

9 Type this line:

```
dw_detail.DeleteRow(0)
```

The argument zero in the DeleteRow function specifies that the current row in the dw_detail control will be deleted. The DeleteRow function does not delete the row from the database; the row is not actually deleted from the database until the Update function is executed.

10 Compile the script by selecting *Design>Compile Script* from the menu bar.

Add a script to retrieve data for the master DataWindow

Where you are

- Create a new window
- Add DataWindow controls
- Add user events
- Add scripts for the user events
- > Add a script to retrieve data for the master DataWindow
- Add a script to retrieve data for the detail DataWindow
- Attach a menu to the window
- Save the window
- Add menu scripts to trigger the user events

The script you just compiled has no effect on the dw_master DataWindow control. But you could add lines to the script to delete the currently selected row in the dw_master DataWindow. Now that you have a script for the ue_retrieve event, all you need to do to retrieve data into the dw_master DataWindow is trigger the event.

So now you will write a script to trigger the ue_retrieve event from the Open event for the window. This will retrieve data into the dw_master DataWindow as soon as the window opens.

- 1 **Click the *Select Event* listbox and select *Open*. Type these lines:**

```
dw_master.settransobject ( sqlca )
dw_detail.settransobject ( sqlca )
this.EVENT ue_retrieve()
```

The first two lines tell the dw_master and dw_detail DataWindows to look in the SQLCA Transaction object for the values of the database variables. The third line triggers the ue_retrieve event. The pronoun *This* refers to the current object. In this example, the window you're working on is the current object.



- 2 **Click the *Return* button.
or
Select *File>Return* from the menu bar.**

The script compiles and you return to the Window painter workspace.

Add a script to retrieve data for the detail DataWindow

Where you are

- Create a new window
 - Add DataWindow controls
 - Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - > Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - Save the window
 - Add menu scripts to trigger the user events
-

Now you will add a script for the RowFocusChanged event of dw_master to retrieve data into the dw_detail DataWindow.

Here's how it will work when you're done:

- ◆ When the window first opens, a list of all customers (or products) displays in the top DataWindow control and detail information for the first customer (or product) displays in the bottom DataWindow control.
- ◆ When a user moves through the list in the top DataWindow control using the UP ARROW and DOWN ARROW keys or by clicking in a row, the details for the current row display in the bottom DataWindow control.

To accomplish this, you will add a script for the RowFocusChanged event of the top DataWindow control (dw_master). That script will send a retrieval request and the ID number of the selected customer (or product) to the bottom DataWindow control (dw_detail).

RowFocusChanged also occurs upon DataWindow display

The RowFocusChanged event also occurs before the DataWindow is displayed. This allows the application to retrieve and display detail information for the first customer (or product) in the master DataWindow.

- 1 **Select the dw_master DataWindow control.**
Select Script from the popup menu.
Click the Select Event listbox and select rowfocuschanged.
Type this line:

```
long ll_itemnum
```

This line declares the local variable ll_itemnum, which has the long data type.

2 Type this line:

```
ll_itemnum = this.object.data[currentrow, 1]
```

Use square brackets

The expression shown above requires square brackets, not parentheses.

This line uses a DataWindow data expression to obtain the item number in column 1 of the currently selected row of dw_master. It stores the number in the variable ll_itemnum. CurrentRow is an argument passed to the RowFocusChanged event that specifies the current row in the DataWindow control. The current row is the row the user has selected by clicking or by scrolling with the arrow or tab keys.

3 Type these lines:

```
IF dw_detail.Retrieve(ll_itemnum) = -1 THEN  
    MessageBox("Retrieve", "Retrieve error-detail")  
END IF
```

This group of lines sends a retrieval request to the dw_detail DataWindow along with the argument the DataWindow expects (a Customer ID or Product ID, depending on which window the user is working with). The IF statement that encloses the Retrieve function checks for successful completion. If the retrieval operation fails, it displays a message box.

**4 Click the *Return* button.
or
Select *File>Return* from the menu bar.**

The script compiles and you return to the Window painter workspace.

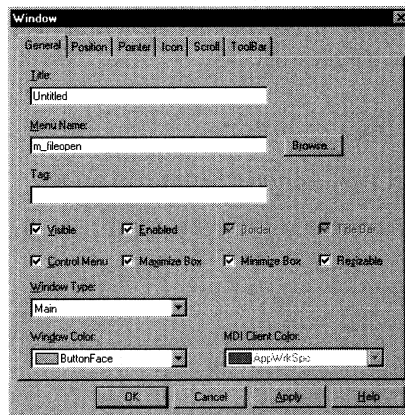
Attach a menu to the window

Where you are

- Create a new window
 - Add DataWindow controls
 - Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - > Attach a menu to the window
 - Save the window
 - Add menu scripts to trigger the user events
-

The window you've just created still needs a menu. Now you will attach `m_fileopen` (the menu you created earlier) to the window.

- 1 **Make sure neither DataWindow control is selected.**
- 2 **Select *Properties* from the window's popup menu.**
- 3 **Type `m_fileopen` in the Menu Name box.**



- 4 **Click *OK*.**

PowerBuilder attaches the menu to the window.

Whenever this window (or any of its descendants) is displayed at execution time, the menu bar will automatically change to `m_fileopen`.

Save the window

Where you are

- Create a new window
 - Add DataWindow controls
 - Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - > Save the window
 - Add menu scripts to trigger the user events
-

Now you will save the window.

1 Select *File>Save* from the menu bar.

The Save Window dialog box opens with the insertion point in the Windows box.

2 If necessary, select the *TUTOR_PB.PBL* file in the *Application Libraries* box.

Type *w_master_detail* as the name of the new window.

Type a comment in the *Comments* box and click *OK*.

The name you just assigned to the window displays in the title bar of the Window painter workspace.

3 Close the *w_master_detail* window.

Add menu scripts to trigger the user events

Where you are

- Create a new window
 - Add DataWindow controls
 - Add user events
 - Add scripts for the user events
 - Add a script to retrieve data for the master DataWindow
 - Add a script to retrieve data for the detail DataWindow
 - Attach a menu to the window
 - Save the window
 - > Add menu scripts to trigger the user events
-

Now you will add scripts that will trigger the new user events.

- 1 **Open the Menu painter and select *m_fileopen*, then click *OK*. If no menus are listed, select *pbtutor.pbl* in the Applications Libraries box.**

- 2 **Select the *Edit* dropdown menu in the Menu painter workspace.**

The insertion point moves to the Edit menu.

- 3 **Select the *Insert* menu item.**



Click the *Script* button in the PainterBar. (You should be in the Clicked event.)

Type these lines:

```
w_master_detail lw_activesheet  
lw_activesheet = w_genapp_frame.GetActiveSheet()  
lw_activesheet.EVENT ue_insert()
```

The first two lines determine which sheet in the MDI frame is currently active. The third line triggers the user event *ue_insert* for the active sheet.



- 4 **Click the *Return* button in the PainterBar.**

or

Select *File>Return* from the menu bar.

The script compiles and you return to the Menu painter workspace.

- 5 **Select the *Update* menu item in the Menu painter workspace.**



Click the *Script* button in the PainterBar. (You should be in the Clicked event.)

Type these lines:

```
w_master_detail lw_activesheet  
lw_activesheet = w_genapp_frame.GetActiveSheet()  
lw_activesheet.EVENT ue_update()
```



- 6 Click the *Return* button.**
or
Select *File>Return* from the menu bar.

The script compiles and you return to the Menu painter workspace.

- 7 Select the *Delete* menu item.**

You may need to scroll down to see it.



- 8 Click the *Script* button in the PainterBar.**
Type these lines:

```
w_master_detail lw_activesheet  
lw_activesheet = w_genapp_frame.GetActiveSheet()  
lw_activesheet.EVENT ue_delete()
```



- 9 Click the *Return* button.**
or
Select *File>Return* from the menu bar.

The script compiles and you return to the Menu painter workspace.

- 10 Select *File>Save* from the menu bar.**

PowerBuilder saves the menu.

- 11 Close the *m_fileopen* menu.**

Building Two Descendent Windows

In this chapter you will:

- ◆ Create a new window to display Customer information that inherits characteristics from `w_master_detail`
- ◆ Create another window to display Product information that also inherits characteristics from `w_master_detail`
- ◆ Add a menu script to open the Customer window
- ◆ Add a menu script to open the Product window
- ◆ Run the application

How long will it take?

About 15 minutes.

Create the Customer window

Where you are

- > Create the Customer window
- Create the Product window
- Add a menu script to open the Customer window
- Add a menu script to open the Product window
- Run the application

Now you will create the Customer window for the application.



- 1 **Click the *Window* button in the PowerBar.**

The Select Window dialog box opens.

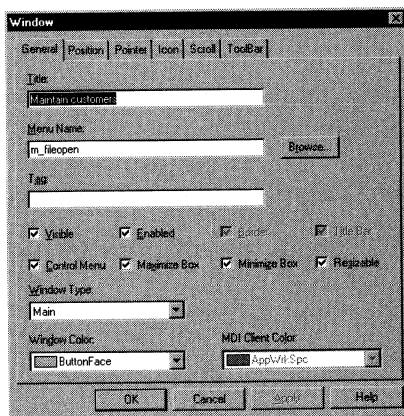
- 2 **Click *Inherit*.**

PowerBuilder displays the Inherit From Window dialog box.

- 3 **Select *w_master_detail*.
Click *OK*.**

PowerBuilder displays an untitled window that has all the characteristics of *w_master_detail*.

- 4 **Make sure neither of the controls in the window is selected.
Select *Properties* from the window's popup menu.
Type *Maintain Customers* in the *Title* box.
Click *OK*.**



5 Select *File>Save* from the menu bar.

The Save Window dialog box opens with the insertion point in the Windows box.

**6 Type *w_customers* as the name of the new window.
Type a comment in the *Comments* box.
Click *OK*.**

The name you just assigned to the window displays in the title bar of the Window painter workspace.

Create the Product window

Where you are

- Create the Customer window
 - > Create the Product window
 - Add a menu script to open the Customer window
 - Add a menu script to open the Product window
 - Run the application
-

Now you will create the Product window for the application.

1 Select *File>Inherit* from the menu bar.

PowerBuilder displays the Inherit From Window dialog box.

**2 Select *w_master_detail*.
Click *OK*.**

PowerBuilder displays an untitled window that has all the characteristics of *w_master_detail*.

**3 Select *Properties* from the window's popup menu.
Type *Maintain Products* in the *Title* box.
Click *OK*.**

4 Select *File>Save* from the menu bar.

The Save Window dialog box opens with the insertion point in the Windows box.

**5 Type *w_products* as the name of the new window.
Type a comment in the *Comments* box.
Click *OK*.**

The name you just assigned to the window displays in the title bar of the Window painter workspace.

6 Close the Window painter.

Add a menu script to open the Customer window

Where you are

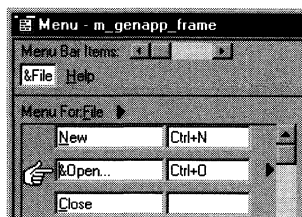
- Create the Customer window
- Create the Product window
- > Add a menu script to open the Customer window
- Add a menu script to open the Product window
- Run the application

Now you will add a script that will open the Customer window from the menu.



- 1 **Open the Menu painter.**
Select *m_genapp_frame*.
Click *OK*.

- 2 **Select the *File* dropdown menu if it's not already selected.**
Click the *Open* menu item.



- 3 **Click the *Next Lvl* button in the PainterBar.**
Select the *Customers* menu item if it's not already selected.
The insertion point moves to the *Customers* menu item.



- 4 **Click the *Script* button in the PainterBar.**
Make sure this is the script for the clicked event.

- 5 **Type this line:**

```
OpenSheet (w_customers, ParentWindow, 0,
          original!)
```

This line creates a menu item on the Window menu:

- ◆ *W_customers* is the window PowerBuilder opens an MDI sheet for
- ◆ *ParentWindow* means the sheet should be opened within the parentwindow of *m_genapp_frame* (*w_genapp_frame*)

- ◆ Zero means the sheet name should be added to the next-to-last menu on the menu bar, which in this case is the Window menu
- ◆ Original! means the position should be opened in its original size



6 Click the *Return* button.

or

Select *File>Return* from the menu bar.

The script compiles and you return to the Menu painter workspace.

Add a menu script to open the Product window

Where you are

- Create the Customer window
 - Create the Product window
 - Add a menu script to open the Customer window
 - > Add a menu script to open the Product window
 - Run the application
-

Now you will add a menu script to open the Product window in the application.

- 1 **While you still have the `m_genapp_frame` menu open, click the *Products* menu item.**

The insertion point moves to the Products menu item.



- 2 **Click the *Script* button in the PainterBar.**

Make sure this is the Clicked event.

- 3 **Type this line:**

```
OpenSheet (w_products, ParentWindow, 0,  
original!)
```

This line opens an MDI sheet for the `w_products` window.



- 4 **Click the *Return* button.**
or
Select *File>Return* from the menu bar.

The script compiles and you return to the Menu painter workspace.

- 5 **Select *File>Save* from the menu bar.**

PowerBuilder saves the `m_genapp_frame` menu.

- 6 **Close the Menu painter.**

Run the application

Where you are

- Create the Customer window
 - Create the Product window
 - Add a menu script to open the Customer window
 - Add a menu script to open the Product window
 - > Run the application
-

Now you will run the application again.



1 Click the *Run* button in the PowerBar.

The application logon window displays.

**2 Type *dba* in the *User ID* box.
Type *sql* in the *Password* box.
Click *OK*.**

The database connection is established and the MDI frame for the application displays.

3 Select *File>Open>Customers* from the menu bar.

Notice that the menu bar changes to include the Edit and Window dropdown menus.

These menus are part of the definition of `m_fileopen`. The `OpenSheet` call you used in the Open menu event script attaches the `m_fileopen` menu to the `w_master_detail` window, the ancestor window for `w_customers`. Since you didn't override the menu assignment for `w_customers`, this window uses `m_fileopen`.

4 Select the *Edit* menu.

The Edit menu has the Insert, Update, and Delete options you added. These options do not function yet, because the `DataWindow` controls in the Customer window do not have `DataWindow` objects associated with them.

5 Select the *Window* menu.

Notice that a new menu item has been added for the sheet you just opened.

6 Select *File>Open>Products* from the menu bar.

A second MDI sheet opens. This sheet cascades relative to the first sheet. The menu bar does not change. That's because `m_fileopen` is the menu for both `w_customers` and `w_products`.

7 Select the *Edit* menu.

Because the `w_products` window uses the `m_fileopen` menu, the Insert, Update, and Delete options are also available when the Product window is open.

8 Select the *Window* menu.

Another menu item has been added for the second sheet you opened.

9 Select *File>Exit* from the menu bar.

The application terminates and you return to the Menu painter workspace.

Run the application

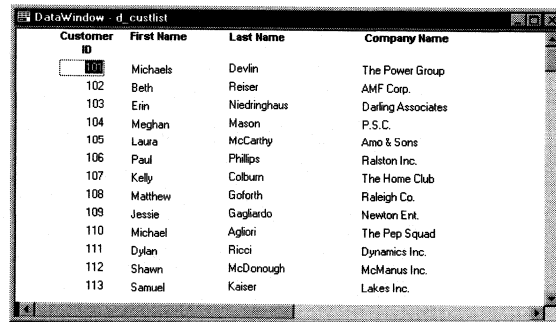
Building the First DataWindow Object

The DataWindow object is one of the most powerful and useful features of PowerBuilder. A DataWindow object can connect to a database, retrieve rows, display the rows in various presentation styles, and update the database.

In this chapter you will:

- ◆ Create the new DataWindow object
- ◆ Preview the DataWindow object
- ◆ Save the DataWindow object
- ◆ Make cosmetic changes

When you are done and preview the DataWindow object, it will look like this.



The screenshot shows a DataWindow object titled "DataWindow - d_custlist". It displays a table with four columns: "Customer ID", "First Name", "Last Name", and "Company Name". The table contains 12 rows of data. The "Customer ID" column has a search box with the value "100" entered. The table is styled with a simple border and a light background.

Customer ID	First Name	Last Name	Company Name
100	Michael	Devlin	The Power Group
102	Beth	Reiser	AMF Corp.
103	Elin	Niedringhaus	Darling Associates
104	Meghan	Mason	P.S.C.
105	Laura	McCarthy	Amo & Sons
106	Paul	Phillips	Ralston Inc.
107	Kelly	Colburn	The Home Club
108	Matthew	Goforth	Raleigh Co.
109	Jessie	Gagliardo	Newton Ent.
110	Michael	Aglioti	The Pep Squad
111	Dylan	Ricci	Dynamics Inc.
112	Shawn	McDonough	McManus Inc.
113	Samuel	Kaiser	Lakes Inc.

How long will it take?

About 15 minutes.

Create the new DataWindow object

Where you are

- > Create the new DataWindow object
- Preview the DataWindow object
- Save the DataWindow object
- Make cosmetic changes

Now you will create a new DataWindow object.



1 Click the *DataWnd* button in the PowerBar.

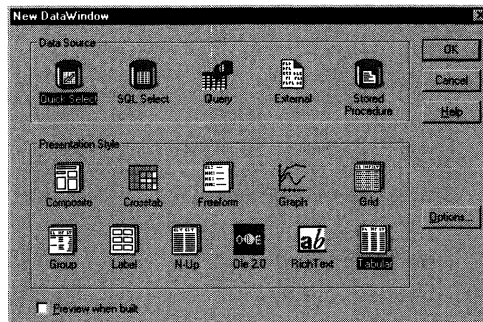
PowerBuilder connects to the Powersoft Demo Database, the DataWindow painter opens, and the Select DataWindow dialog box displays.

In the Select DataWindow dialog box, you can select an existing DataWindow object and modify it, or you can create a new one. You will create a new DataWindow object.

2 Click *New*.

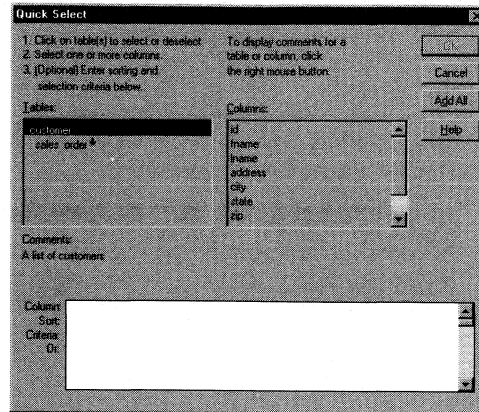
The New DataWindow dialog box displays.

- ### 3 Select *Quick Select* as the data source. Select *Tabular* as the presentation style. Make sure that the *Preview when built* checkbox is not checked.



4 Click OK.

PowerBuilder displays the Quick Select dialog box. The Tables box automatically lists all tables in the current database.

**5 Click *customer*.**

This opens the table and lists its columns. For this DataWindow, you select four columns.

6 Click *id* first.

Then click *fname* and *lname*.

Then scroll down the list and click *company_name*.

PowerBuilder displays the selected columns in a grid at the bottom of the Quick Select dialog box.

Selection order determines display order

The order in which you select the columns determines the left-to-right display order in the DataWindow object.

You can use the grid area at the bottom of the dialog box to specify sort criteria (for the SQL ORDER BY clause) and selection criteria (for the SQL WHERE clause). Now you will specify sort criteria only.

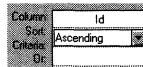
You will sort on the *id* column in ascending order.

7 In the grid, click in the cell beside *Sort* and below *Id*.

A dropdown listbox displays.

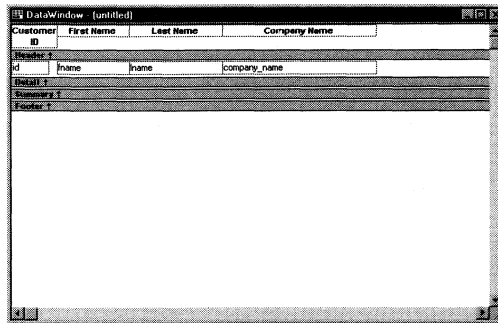
8 Select *Ascending* from the dropdown listbox.

This specifies that the id column will be sorted in ascending order.



9 Click *OK*.

PowerBuilder creates the new DataWindow object and displays it in the DataWindow painter workspace.



In the header band, PowerBuilder displays the headings that were specified in the extended attributes information for the table in the Table painter. They are just starting points; you can override them if you want.

About the DataWindow painter workspace

The DataWindow painter workspace is divided into four areas called **bands**: header, detail, summary, and footer. You can modify these bands. For example, you can change their sizes; add text, lines, boxes, or ovals; and change colors and fonts. If a band is not open, you can create as much space as needed by pointing at the bar (the gray area with an up arrow), pressing and holding the left mouse button, and dragging down until you see the amount of space you want.

Preview the DataWindow object

Where you are

- Create the new DataWindow object
 - > Preview the DataWindow object
 - Save the DataWindow object
 - Make cosmetic changes
-

You can preview the new DataWindow object immediately.

1 Select *Design>Preview* from the menu bar.

PowerBuilder displays the DataWindow as it will appear during execution. PowerBuilder displays data and header information in the DataWindow. If you had specified footer information, it would be displayed too.

All customers in the table display. Customers are sorted by customer ID, just as you specified.



Customer ID	First Name	Last Name	Company Name
100	Michaels	Devlin	The Power Group
102	Beth	Reiser	AMF Corp.
103	Erin	Niedringhaus	Darling Associates
104	Meghan	Mason	P.S.C.
105	Laura	McCarthy	Arno & Sons
106	Paul	Phillips	Ralston Inc.
107	Kelly	Colburn	The Home Club
108	Matthew	Goforth	Raleigh Co.
109	Jessie	Gagliardo	Newton Ent.
110	Michael	Agliori	The Pep Squad
111	Dylan	Ricci	Dynamics Inc.
112	Shawn	McDonough	McManus Inc.

Refreshing the display

Sometimes if you change a DataWindow's format, you may need to click the Retrieve button on the PainterBar to refresh the display.

2 Deselect *Design>Preview* from the menu bar.

You return to the DataWindow painter workspace.

Save the DataWindow object

Where you are

- Create the new DataWindow object
 - Preview the DataWindow object
 - > Save the DataWindow object
 - Make cosmetic changes
-

Now you will name the DataWindow object and save it.

1 Select *File>Save* from the menu bar.

The Save DataWindow dialog box displays with the insertion point in the DataWindows box.

2 Type *d_custlist*.

This names the DataWindow object. The prefix *d_* is standard for DataWindow objects.

3 Enter comments in the *Comments* box.

4 Click *OK*.

PowerBuilder saves the DataWindow object and closes the Save DataWindow dialog box.

Make cosmetic changes

Where you are

- Create the new DataWindow object
 - Preview the DataWindow object
 - Save the DataWindow object
 - > Make cosmetic changes
-

Now you need to make some cosmetic changes to the DataWindow. You need to reposition the columns and column headings to make room for the hand pointer, which displays to the left of the currently selected row. You also need to move some of the columns to make them line up with their headings.

- 1 **Select** *Edit>Select>Select All* from the menu bar.

or

Press CTRL+A.

All of the items in the DataWindow object should now be selected.

- 2 **Position the mouse pointer over one of the selected objects.**
Drag the object to the right about one inch.
Clear the selection.

All of the selected objects move together.

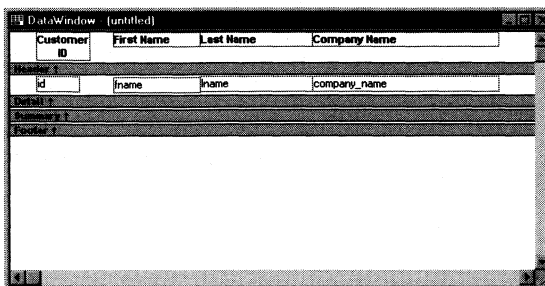


- 3 **Click on the *Customer ID* header.**
Hold down the CTRL key and click on the *id* column.
Position the mouse pointer over one of the selected objects.
Drag the objects to the left about one-half inch.
Click the *Center* button in the StyleBar.
Clear the selection.



- 4 **Click on the *First Name* header.**
Hold down the CTRL key and click on the *Last Name* and *Company Name* headers.
Release the CTRL key.
Click the *Left* button in the StyleBar.

When you have finished, the workspace should look something like this.



- 5 **Select *Design>Preview* from the menu bar to see how the DataWindow looks.**
- 6 **Deselect *Design>Preview* from the menu bar.**
You return to the DataWindow painter workspace.
- 7 **Select *File>Close* from the menu bar.**
- 8 **Click *Yes*.**

PowerBuilder saves the DataWindow object and closes the DataWindow painter.

Adding the First DataWindow

After you create and save a DataWindow *object*, you can use it in a window. To do this, you associate the DataWindow object with a DataWindow *control* in the window.

In this chapter you will:

- ◆ Add the DataWindow object you just built to the Customer window
- ◆ Run the application to see the DataWindow object at work

How long will it take?

About 5 minutes.

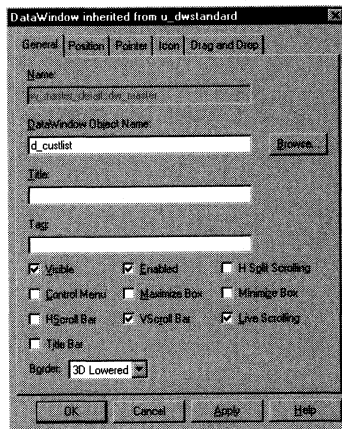
Attach the DataWindow object to the control

Where you are

- > Attach the DataWindow object to the control
 - Run the application
-

Now you will attach the DataWindow object to a DataWindow control in the `w_customers` window.

- 1 **Open the Window painter.**
Select `w_customers`.
Click `OK`.
- 2 **Select the `dw_master` DataWindow control.**
Select *Properties* from the `dw_master` DataWindow control's popup menu.
- 3 **Type `d_custlist` in the *DataWindow Object Name* box.**



- 4 **Click `OK`.**

PowerBuilder associates the `d_custlist` DataWindow object with the `dw_master` DataWindow control. The Window painter workspace now shows the `d_custlist` DataWindow object inside the `dw_master` control. You see the headings for the DataWindow object. But you do not see any data yet. The DataWindow will not execute its `SELECT` statement until you run the application.

Run the application

Where you are

- Attach the DataWindow object to the control
 - > Run the application
-

Now you will run the application again.



- 1 Click the *Run* button in the PowerBar.**

PowerBuilder prompts you to save your changes.

- 2 Click *Yes*.**

The application begins running and the logon window displays.

- 3 Type *dba* in the *User ID* box.
Type *sql* in the *Password* box.
Click *OK*.**

The database connection is established, and the MDI frame for the application displays.

- 4 Select *File>Open>Customers* from the menu bar.**

The Customer window now displays. The top DataWindow control (dw_master) shows data retrieved from the Customer table. The hand pointer means the first row is selected. The bottom DataWindow control (dw_detail) is empty.

- 5 Click inside the DataWindow control to select a different customer row.**

The hand pointer moves to that row.

- 6 Select *File>Exit* from the menu bar.**

The application closes and you return to the Window painter.

- 7 Close the Window painter.**

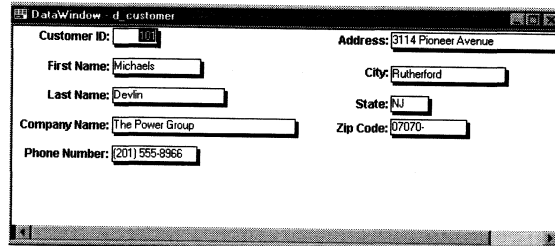
Building the Second DataWindow Object

In this chapter you will create a second DataWindow object to show the details for a selected customer.

You will:

- ◆ Create the new DataWindow object
- ◆ Preview the DataWindow object
- ◆ Save the DataWindow object
- ◆ Enhance the DataWindow object

When you finish this chapter and preview the DataWindow object, it will look like this.



The screenshot shows a DataWindow object titled "d_customer" with a form layout. The form contains the following fields and values:

Field	Value
Customer ID:	005
Address:	3114 Pioneer Avenue
First Name:	Michaels
City:	Rutherford
Last Name:	Devin
State:	NJ
Company Name:	The Power Group
Zip Code:	07070
Phone Number:	(201) 555-9966

How long will it take?

About 20 minutes.

Create the new DataWindow object

Where you are

- > Create the new DataWindow object
 - Preview the DataWindow object
 - Save the DataWindow object
 - Enhance the DataWindow object
-

When you built the first DataWindow object, you used Quick Select to specify the table and columns. This let you retrieve all the customers without having to use the Select painter.

But to build the second DataWindow object, you need to use the Select painter. You need to define a retrieval argument and WHERE criteria so you can pass an argument to the DataWindow object during execution. In this case, you will be passing the customer ID.

Now you will:

- ◆ Select the data source and style
- ◆ Select the table and columns
- ◆ Define a retrieval argument
- ◆ Specify a WHERE clause

Select the data source and style

Now you will select a data source and define how the data is to be presented.



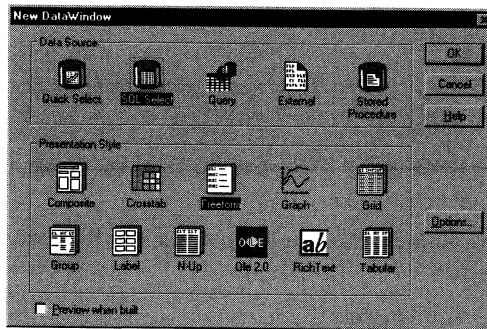
1 Click the *DataWnd* button in the PowerBar.

The DataWindow painter opens and the Select DataWindow dialog box displays. It lists the *d_custlist* DataWindow object, which you created in the preceding chapter.

2 Click *New*.

The New DataWindow dialog box displays.

3 Click *SQL Select* in the *Data Source* box. Click *FreeForm* in the *Presentation Style* box.



4 Click *OK*.

Since the data source is *SQL Select*, you go to the Select painter and the Select Tables dialog box displays.

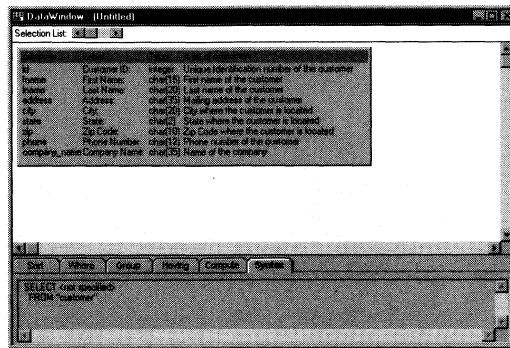
Select the table and columns

Now you will select the table and the columns from that table to use in the DataWindow object.

1 **Select *customer* in the list of tables.**

2 **Click *Open*.**

The Select painter displays the table and its columns.



3 **Select these column names in this order:**

id
fname
lname
company_name
phone
address
city
state
zip

After you click a column name, the name is highlighted and displays in the Selection List area along the top of the workspace. The order in which you select the columns determines the order in which they appear in the Selection List area.

Selection List: **id** | fname | lname | company_name | phone | address | city | state | zip

Selecting all columns in a table

There is an easy way to select all of the columns in a table. Point to the heading area, click the right mouse button, and then click Select All. PowerBuilder highlights the columns in the table and displays them in the selection list. The columns are added in the order in which they appear in the table definition. After adding the columns, you can change the order in the selection list.

- 4 **Click the *Syntax* tab at the bottom of the DataWindow painter if it's not already selected.**

The generated SELECT statement displays in the tab area.

Define a retrieval argument

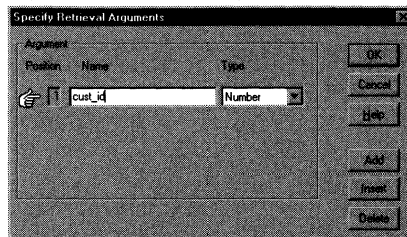
Now you will define a retrieval argument.

- 1 **Select *Design>Retrieval Arguments* from the menu bar.**

The Specify Retrieval Arguments dialog box displays.

- 2 **Type *cust_id* in the Name box.**

The default data type is Number, which is what you want.



About retrieval argument names

You can choose any name you want for the retrieval argument; it is just a placeholder for the value you will pass during execution. Nonetheless, it is a good idea to make the name meaningful.

- 3 **Click *OK*.**

The retrieval argument is defined.

Specify a WHERE clause

Now you will specify a WHERE clause using the retrieval argument to retrieve a specific customer.

1 Click the *Where* tab.

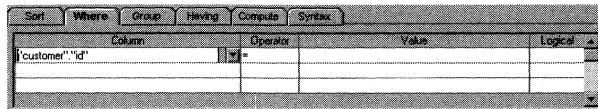
The Where tab displays.

2 Click in the box below *Column*.

A down arrow displays.

**3 Click the *down arrow*.
Select "*customer*".*id*".**

It displays immediately below the Column heading. An equal sign (=) appears in the Operator box. This is what you want, so do not change it.



**4 Position the mouse pointer on the box below *Value*.
Click the right mouse button.**

The popup menu displays.

5 Select *Arguments*.

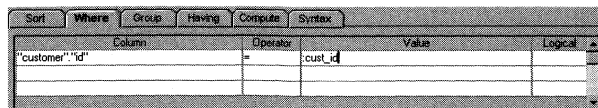
The arguments dialog box displays.

The arguments dialog box contains the retrieval argument you specified earlier (cust_id).

The colon before cust_id means it is a PowerBuilder variable that can be used as an argument in a SQL statement.

**6 Select *:cust_id*.
Click the *Paste* button.**

The Where tab displays with the retrieval argument.



7 Click the *Syntax* tab.

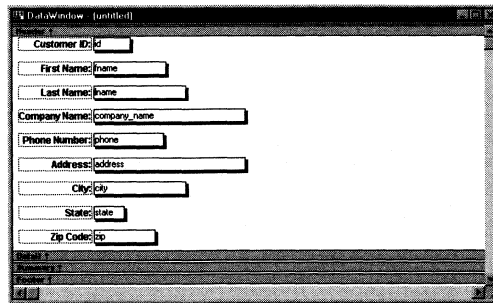
The SELECT statement displays in the tab area.

8 Scroll down until you see the generated WHERE clause.

You have now created a complete SQL SELECT statement that retrieves data in several columns in the customer table where the id column is equal to an argument that will be supplied during execution.

9 Deselect the *SQL Data* button in the PainterBar.

PowerBuilder creates the new DataWindow object and displays it in the DataWindow painter workspace.



PowerBuilder uses the labels specified in the extended attribute information in the Table painter. If your DataWindow has different labels, you can either continue with the tutorial or change them to match the labels shown here.

Preview the DataWindow object

Where you are

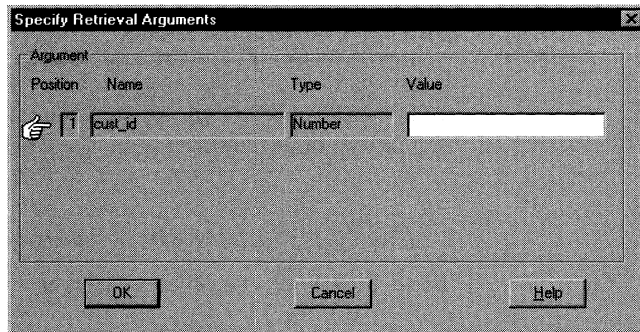
- Create the new DataWindow object
 - > Preview the DataWindow object
 - Save the DataWindow object
 - Enhance the DataWindow object
-

Now you will preview the new DataWindow object.

1 Select *Design>Preview* from the menu bar.

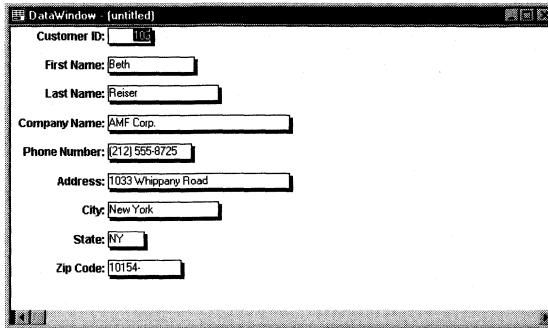
Since this DataWindow object requires a retrieval argument, the Specify Retrieval Arguments dialog box displays.

PowerBuilder prompts for the argument. When you put this DataWindow object into the application, you will write a script that will pass the required argument to the DataWindow object automatically.



- 2 **Type a customer ID in the value field (such as 101, 102, or 103). Click OK.**

The DataWindow object retrieves the requested customer data.



The screenshot shows a window titled "DataWindow - (untitled)". Inside the window, there are several text fields with labels to their left. The labels and their corresponding values are: "Customer ID:" with "101", "First Name:" with "Beth", "Last Name:" with "Fleiser", "Company Name:" with "AMF Corp.", "Phone Number:" with "(212) 555-9725", "Address:" with "1033 Whippary Road", "City:" with "New York", "State:" with "NY", and "Zip Code:" with "10154".

PowerBuilder remembers the retrieval argument

The next time you preview the DataWindow, PowerBuilder will go directly to the DataWindow display, using the retrieval argument you specified the first time. PowerBuilder stores the argument until you leave the DataWindow painter.

If you wanted to access information for another customer, you would click the Retrieve button, which would display the Specify Retrieval Arguments dialog box.

- 3 **Deselect *Design>Preview* from the menu bar.**

You return to the DataWindow painter workspace.

Save the DataWindow object

Where you are

- Create the new DataWindow object
 - Preview the DataWindow object
 - > Save the DataWindow object
 - Enhance the DataWindow object
-

Now you will name the DataWindow object and save it. You could wait to save it until you leave the painter, but it's good practice to save your work frequently.

- 1 Select *File>Save* from the menu bar.**

The Save DataWindow dialog box displays.

- 2 Type *d_customer* in the *DataWindows* box.
Type a comment in the *Comments* box.**

- 3 Click *OK*.**

You return to the DataWindow painter.

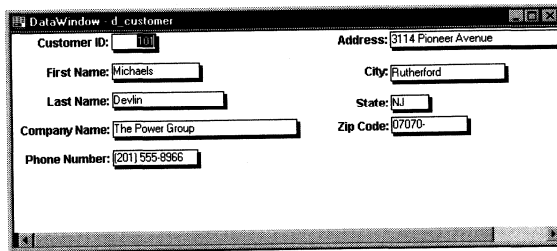
Enhance the DataWindow object

Where you are

- Create the new DataWindow object
 - Preview the DataWindow object
 - Save the DataWindow object
 - > Enhance the DataWindow object
-

Now you will modify the DataWindow object. You will:

- ◆ Rearrange the columns and text
- ◆ Align the labels and columns



The screenshot shows a DataWindow object titled "d_customer". It contains a form with the following fields and values:

Field	Value
Customer ID:	101
Address:	3114 Pioneer Avenue
First Name:	Michaels
City:	Rutherford
Last Name:	Devlin
State:	NJ
Company Name:	The Power Group
Zip Code:	07070
Phone Number:	(201) 555-8966

Columns on freeform DataWindows

Data fields on freeform DataWindow objects are still called *columns*, even though they're shown in a nontabular display.

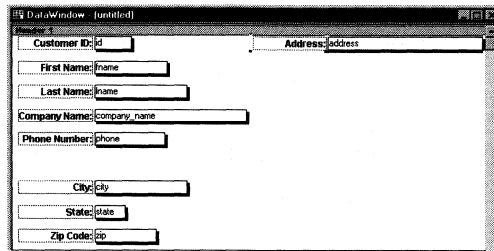
Rearrange the columns

Now you will rearrange the columns in the new DataWindow object.

- 1 **Click the *Address:* label.**
Hold the CTRL key and click the *address* column.

The two items are selected.

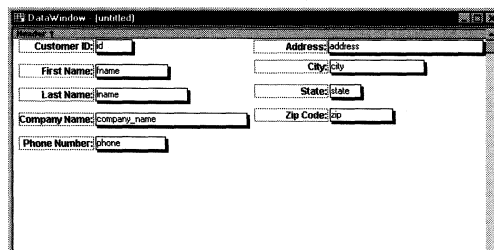
- 2 **Release the CTRL key.**
Position the cursor on one of the selected objects and press the left mouse button.
Drag the objects to the top-right corner of the DataWindow object.



- 3 **Use the CTRL+click technique to select the *City:*, *State:*, and *Zip:* labels and the *city*, *state*, and *zip* columns.**
Drag these objects under the *Address:* label and *address* column.
Clear the current selection.

If necessary, scroll up until you can see all the columns in the DataWindow.

The DataWindow should look like this.

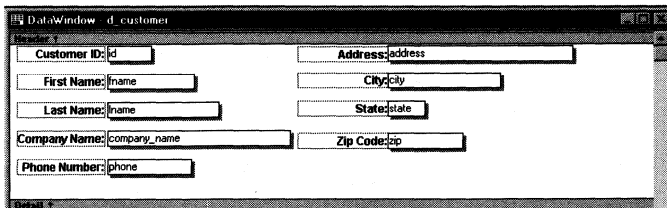


- 4 **Remove any extra space you see in the detail area by dragging the Detail bar up until the columns fill the area.**

Align the labels and columns

Now you will align the labels and columns on the new DataWindow.

- 1 **Select the *Zip Code: label*.**
Make sure that the *Zip Code: label* is the only object selected.
- 2 **Move the *Zip Code: label* as close as possible to the *company_name* column.**
- 3 **While the *Zip Code: label* is still selected, use the CTRL+click technique to select the *Address:*, *City:*, and *State:* labels.**
- 4 **Select *Edit>Align Objects* from the menu bar.**
The align options display.
- 5 **Select the first option (*left*) to left-align the labels.**
PowerBuilder aligns the objects with the *first* item you selected (the *Zip Code: label*).
- 6 **Move the *zip* column so that it is right next to the *Zip Code: label*. Align the *address*, *city*, and *state* columns with the *zip* column, just as you aligned the column labels.**



Preview again

Now you will preview the cosmetic changes you made.

1 Select *Design>Preview* from the menu bar.

The DataWindow uses the argument you specified the last time Preview was requested to retrieve the specified customer.

You may need to specify a retrieval argument

If you have closed the DataWindow painter since the last Preview was requested, PowerBuilder displays the Specify Retrieval Arguments dialog box. In this case, type a customer number (such as 101, 102, or 103) and click OK.

2 Deselect *Design>Preview* from the menu bar.

You return to the DataWindow painter workspace.

At this point you could make many other changes to the DataWindow object. For example, you could change colors, add lines, remove columns, and change labels and display formats.

For this tutorial, the DataWindow object is ready to be used in the window you created earlier.

3 Select *File>Close* from the menu bar.

PowerBuilder prompts you to save your changes.

4 Click Yes.

PowerBuilder saves the DataWindow object and closes the DataWindow painter.

Adding the Second DataWindow

In this chapter you will:

- ◆ Add the second DataWindow object you built to the Customer window
- ◆ Run the application again to test the insert, update, and delete capabilities of the second DataWindow

How long will it take?

About 5 minutes.

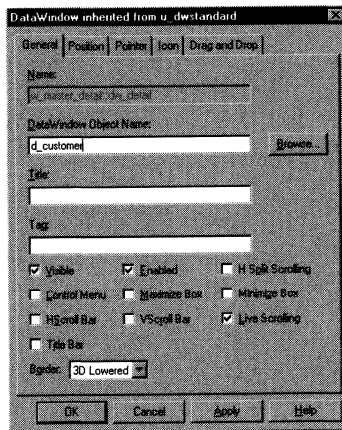
Attach the DataWindow object to the control

Where you are

- > Attach the DataWindow object to the control
 - Run the application
-

Now you will attach the new DataWindow object to the control.

- 1 **Open the Window painter.**
Select *w_customers*.
Click **OK**.
- 2 **Select *Properties* from the *dw_detail* DataWindow control's popup menu.**
- 3 **Type *d_customer* in the *DataWindow Object Name* box.**



- 4 **Click **OK**.**

PowerBuilder associates the *d_customer* DataWindow object with the *dw_detail* DataWindow control. The Window painter workspace now shows the *d_customer* DataWindow object inside the *dw_detail* control.

- 5 If necessary, make the dw_detail control larger so you can see all of the columns you added to the DataWindow object.

If you need to, you can also enlarge the window to make more room. If you make the dw_detail control wider, you may also want to make the dw_master control the same width.

The screenshot shows a DataWindow object titled "Window w customers inherited from w master detail". It contains a table with the following columns: Customer ID, First Name, Last Name, and Company Name. Below the table, there are input fields for the following fields: Customer ID, Address, First Name, City, Last Name, State, Company Name, Zip Code, and Phone Number. The input fields are arranged in a grid-like fashion, with some fields having a small icon next to them.

Run the application

Where you are

Attach the DataWindow object to the control

> Run the application

Now you will run the application again to test the insert, update, and delete capabilities of the second DataWindow.



1 Click the *Run* button in the PowerBar.

PowerBuilder prompts you to save your changes.

2 Click Yes.

The application begins running and the logon window displays.

**3 Type *dba* in the *User ID* box.
Type *sql* in the *Password* box.
Click *OK*.**

The database connection is established, and the MDI frame for the application displays.

4 Select *File>Open>Customers* from the menu bar.

The Customer window displays.

Customer ID	First Name	Last Name	Company Name
100	Michaels	Devlin	The Power Group
102	Beth	Reiner	AMF Corp.
103	Elin	Niedringhaus	Darling Associates
104	Meghan	Mason	P.S.C.
105	Laura	McCarthy	Amo & Sons
106	Paul	Phillips	Ralston Inc.
107	Kelly	Colburn	The Home Club

Customer ID: 100 Address: 3114 Pioneer Avenue

First Name: Michaels City: Rutherford

Last Name: Devlin State: NJ

Company Name: The Power Group Zip Code: 07070

Phone Number: 201 555-8966

The top DataWindow control (dw_master) shows all of the rows retrieved from the Customer table. The hand pointer shows which row is selected.

The bottom DataWindow control (dw_detail) shows information about the first customer.

5 Click the *Insert* button in the toolbar.

or

Select *Edit>Insert* from the menu bar.

This clears (resets) the dw_detail DataWindow and adds a new row to the DataWindow. The cursor is in the Customer ID box in the dw_detail control.

6 Add a new customer row by entering information in the boxes in the lower DataWindow.

Specifying the customer's state

To specify the state in which the new customer resides, you need to use a dropdown listbox.

- 7 Click the *Update* button in the toolbar.**

or

Select *Edit>Update* from the menu bar.

This sends the new customer data to the database and displays a confirmation message, as coded in the script for the `ue_update` event.

In the tutorial application, the new customer does not display in the top DataWindow. (You could code the script to include this feature.)

- 8 Click a customer in the top DataWindow.**

That customer's data displays in the lower DataWindow.

- 9 Change the customer's address.**

- 10 Click the *Update* button in the toolbar.**

or

Select *Edit>Update* from the menu bar.

This sends the revised customer data to the database and displays another confirmation message.

- 11 Select another customer in the top DataWindow.**

That customer's data displays in the lower DataWindow.

- 12 Click the *Delete* button in the toolbar.**

or

Select *Edit>Delete* from the menu bar.

The customer is deleted from the DataWindow immediately but will not be deleted from the database unless you select the Update option on the Edit menu. In this particular situation, the Update operation may fail, because rows in other tables in the Powersoft Demo Database may refer to the row you're trying to delete.

You should be able to delete any row that you have added to the database.

- 13 Select *File>Exit* from the menu bar.**

The application terminates and you return to the Window painter.

- 14 Close the Window painter.**

Adding DataWindows to the Product Window

In this chapter you will:

- ◆ Add two DataWindow objects to the Product window
- ◆ Run the application again to test the Product window

How long will it take?

About 15 minutes.

Attach the DataWindow objects

Where you are

- > Attach the DataWindow objects
 - Run the application
-

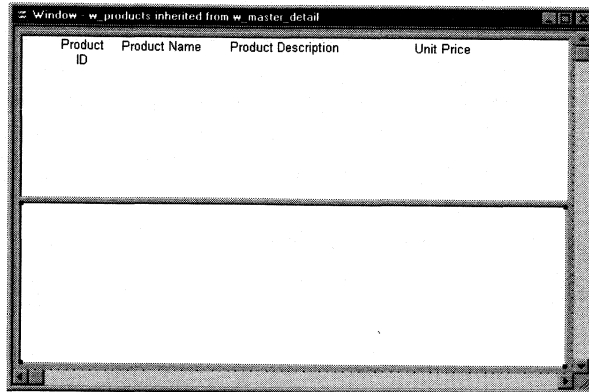
Now you will add two DataWindow objects to the w_products window. The DataWindow objects you will use are provided for you in the tutor_pb.pbl library.

- 1 Open the Window painter.**
Select *w_products*.
Click *OK*.
- 2 Select *Properties* from the dw_master DataWindow control's popup menu.**
- 3 Click *Browse*.**
PowerBuilder displays the Select DataWindow dialog box.
- 4 Select the *TUTOR_PB.PBL* file in the *Application Libraries* box.**
Select *d_prodlst*.
Click *OK*.

You return to the DataWindow property sheet.

5 Click OK.

PowerBuilder associates the `d_prodlist` DataWindow object with the `dw_master` DataWindow control. The Window painter workspace now shows the `d_prodlist` DataWindow object inside the `dw_master` control. You see the headings for the DataWindow object. You may need to resize the control and/or the window.

**6 Display the `dw_detail` DataWindow Property sheet. Click *Browse*.**

The Select DataWindow dialog box displays.

7 Select the `TUTOR_PB.PBL` file in the *Application Libraries* box. Select `d_product`. Click *OK*.

You return to the DataWindow properties sheet.

8 Click *OK*.

PowerBuilder associates the `d_product` DataWindow object with the `dw_detail` DataWindow control. The Window painter workspace now shows the `d_product` DataWindow object inside the `dw_detail` control. The `d_product` DataWindow object has seven columns labeled Product ID, Product Name, Product Description, Size, Color, Quantity, and Unit Price.

If necessary, make the `dw_detail` control larger so you can see all of the columns in the DataWindow object. You can also enlarge the window to make more room.

Run the application

Where you are

Attach the DataWindow objects

> Run the application

Now you will run the application again to test the Product window.

At this point, the Product window should have all of the capabilities of the Customer window. Like the Customer window, the Product window functions as a master/detail window, providing support for retrieval, insert, update, and delete operations against the database.



1 Click the *Run* button in the PowerBar.

PowerBuilder prompts you to save your changes.

2 Click *Yes*.

The application begins running and the logon window displays.

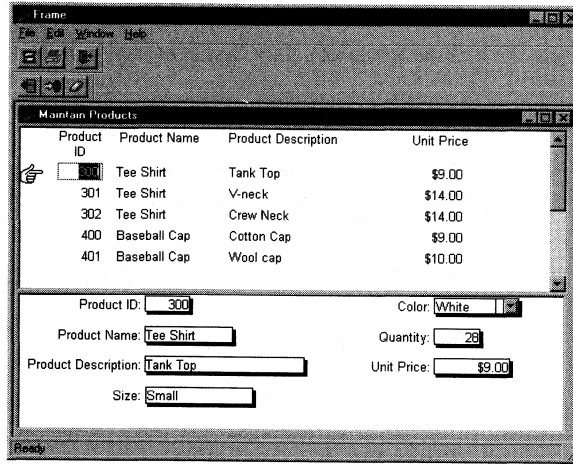
**3 Type *dba* in the *User ID* box.
Type *sql* in the *Password* box.
Click *OK*.**

The database connection is established, and the MDI frame for the application displays.

4 Select *File>Open>Products* from the menu bar.

The Product window displays. The top DataWindow control (dw_master) shows all of the rows retrieved from the Product table. The hand pointer means the first product row is selected.

The bottom DataWindow control (dw_detail) shows information about the first product.



5 Select *Edit>Insert* from the menu bar.

This clears (resets) the dw_detail DataWindow and adds a new row to the DataWindow. The cursor is in the Product ID box in the dw_detail control.

6 Add a new product row by entering information in the boxes in the lower DataWindow. Use the TAB key to move from box to box.

7 Select *Edit>Update* from the menu bar.

This sends the new product data to the database and displays a confirmation message, as coded in the script for the ue_update event.

In the tutorial application, the new product does not display in the top DataWindow. (You could code the script to include this feature.)

8 Click a product in the top DataWindow.

That product's data displays in the lower DataWindow.

9 Change the product's unit price.

10 Select *Edit>Update* from the menu bar.

This sends the revised product data to the database and displays another confirmation message.

11 Select another product in the top DataWindow.

That product's data displays in the bottom DataWindow.

12 Select *Edit>Delete* from the menu bar.

The product is deleted from the DataWindow immediately but will not be deleted from the database until you select the Update option on the Edit menu.

You could program this update to occur automatically.

13 Select *File>Exit* from the menu bar.

The application closes and you return to the Window painter.

14 Close the Window painter.

Running the Debugger

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.

In debug mode, you can set breakpoints (stops) in scripts and functions, step through the code line by line, and display the contents of variables to locate logic errors and mistakes that will result in errors during execution. When you run your application in debug mode, PowerBuilder suspends execution just before it hits a line containing a breakpoint. You can then look at (and change) the values of variables.

In this chapter you will:

- ◆ Add simple breakpoints to the tutorial application
- ◆ Run the tutorial application in debug mode
- ◆ Step through the application examining variables
- ◆ Set a watch on a variable and a conditional breakpoint

How long will it take?

About 10 minutes.

Add a breakpoint for the Application object

Where you are

- > Add a breakpoint for the Application object
 - Add breakpoints for the Welcome and Customer windows
 - Run in debug mode
 - Step through the code line by line
 - Stop at breakpoints and examine variables
 - Set a watch and a conditional breakpoint
-

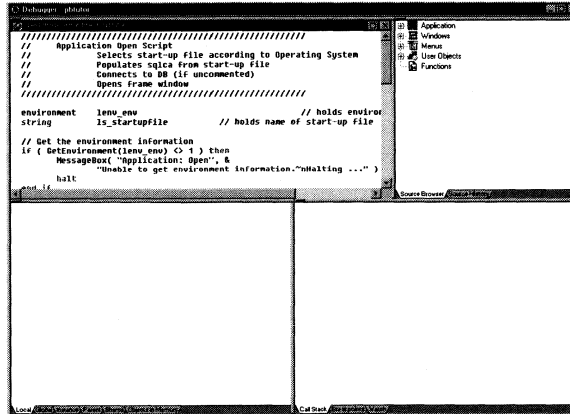
Now you will open the Debug window and add a breakpoint for the Open event of the Application object.



1 Click the **Debug** button in the PowerBar.

PowerBuilder opens the Debug window. This window contains several views, each displaying in a separate pane and showing a different kind of information. Several of the panes are overlapped with a tab for each pane.

The source code for the application's open event displays in the Source pane at top left.



If the Debug window looks different

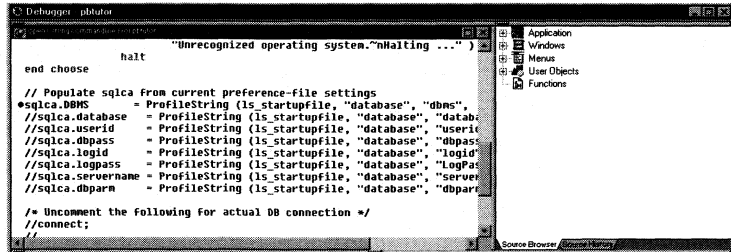
If you have opened the Debug window before and opened, moved, or closed any panes, your display may look different. To restore the default layout, select Debug>Options from the menu bar and select the Default button on the Layout tab page.

2 Scroll down through the source code until you see this statement:

```
sqlca.DBMS = ProfileString (ls_startupfile,
                           "database", "dbms", "")
```

3 Double-click the line containing the statement.

A red symbol displays at the start of the line to show that a breakpoint has been set on the statement.

**Select executable lines**

When you're inserting breakpoints in a script, select lines that contain executable statements. If you try to set a breakpoint in variable-declaration lines, comment lines, or blank lines, PowerBuilder sets the breakpoint at the next executable line.

When PowerBuilder runs the application in debug mode, it stops just before executing a line containing a breakpoint.

Add breakpoints for the Welcome and Customer windows

Where you are

- Add a breakpoint for the Application object
 - > Add breakpoints for the Welcome and Customer windows
 - Run in debug mode
 - Step through the code line by line
 - Stop at breakpoints and examine variables
 - Set a watch and a conditional breakpoint
-

Now you will add breakpoints so you can examine the behavior of the Welcome and Customer windows. To do this, you need to add a breakpoint to a function in the w_welcome window so that you can verify that the correct values are being assigned to variables when a user logs in. You also need to add a breakpoint to one of the scripts for w_master_detail, the ancestor window from which the Customer window inherits characteristics, so that you can verify what happens when a user selects a different row in the window.

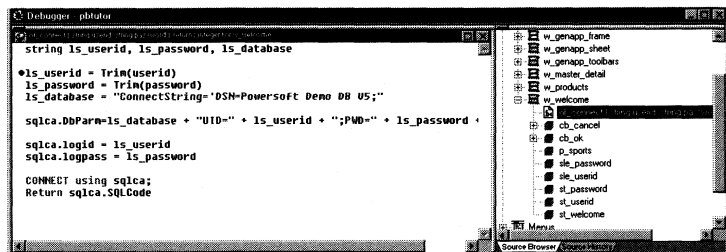
- 1 Double-click *Windows in the Source Browser view.***
Double-click *w_welcome.*
Double-click *of_connect.*

PowerBuilder displays the script for the of_connect function of the w_welcome window in the Source view.

- 2 Double-click this line:**

```
ls_userid = Trim(userid)
```

A breakpoint symbol displays at the start of the line.



- 3 **Double-click `w_master_detail` in the Source Browser view. Double-click `dw_master` and then `rowfocuschanged`.**

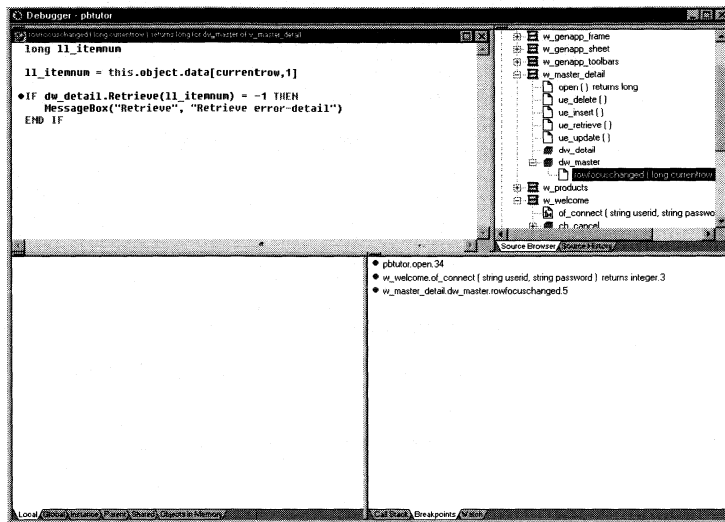
PowerBuilder displays the script for the RowFocusChanged event of the `dw_master` DataWindow control in the Source view.

- 4 **Double-click this line:**

```
IF dw_detail.Retrieve(ll_itemnum) = -1 THEN
```

A breakpoint symbol displays at the start of the line.

- 5 **Select the *Breakpoints* tab in the lower-right pane.**



You should see the three breakpoints you set in the Breakpoints view. To complete this chapter, you need to have these breakpoints, and only these breakpoints, set correctly. You can clear any excess breakpoints using the popup menu in the Breakpoints view.

Run in debug mode

Where you are

Add a breakpoint for the Application object

Add breakpoints for the Welcome and Customer windows

> **Run in debug mode**

Step through the code line by line

Stop at breakpoints and examine variables

Set a watch and a conditional breakpoint

Now you will run the application in debug mode.



1 Click the *Start* button in the PainterBar.

or

Select *Debug>Start* from the menu bar.

The application starts and runs until it hits a breakpoint. You return to the Debug window, with the line containing the breakpoint displayed. The yellow arrow cursor means that this line contains the next statement to be executed. You can now examine the application using Debug window views and tools.

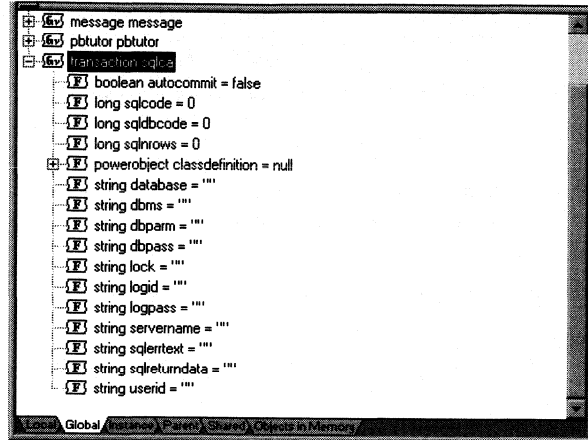
2 Click the *Global* tab in the lower-left pane.

The Global Variables view displays.

3 Double-click *transaction sqlca*.

4 Find the *dbms* property.

Notice that this property does not yet have a value associated with it (because the debugger interrupted execution before the `ProfileString` function executed).



Step through the code line by line

Where you are

- Add a breakpoint for the Application object
 - Add breakpoints for the Welcome and Customer windows
 - Run in debug mode
 - > Step through the code line by line
 - Stop at breakpoints and examine variables
 - Set a watch and a conditional breakpoint
-

Now you will step through the code line by line. You begin stepping from the line where you just stopped execution of the application.



- 1 **To execute the next statement, click the *Step In* button in the PainterBar.**

or

Select *Debug>Step In* from the menu bar.

Now the dbms property has a value. It was assigned when the previous statement was executed. The value depends on which DBMS you are using.

```
...[F] string database = ""  
...[F] string dbms = "ODBC"  
...[F] string dbparm = ""
```

About the Step buttons

You can use either Step In or Step Over to step through an application one statement at a time. They have the same result except when the next statement contains a call to a function.

Use Step In if you want to step into a function and examine the effects of each statement in the function. If you've stepped into a function, you can use Step Out to execute the rest of the function as a single step. You return to the next statement in the script that called the function.

Use Step Over to execute the function as a single statement.

- 2 **Use Step In or Step Over to step through the code until you reach this statement:**

```
open(w_welcome)
```

When PowerBuilder executes this statement, the logon window displays and the PowerBuilder window is minimized.

- 3 **Type *dba* in the *User ID* box.**
Type *sql* in the *Password* box.
Click *OK*.

Execution resumes at the next executable statement. The Source view shows the clicked script for the `cb_ok` command button with the pointer on the first line.

Stop at breakpoints and examine variables

Where you are

- Add a breakpoint for the Application object
 - Add breakpoints for the Welcome and Customer windows
 - Run in debug mode
 - Step through the code line by line
 - > Stop at breakpoints and examine variables
 - Set a watch and a conditional breakpoint
-

Now you will continue execution without stepping through each statement.



1 Click the *Continue* button in the PainterBar.

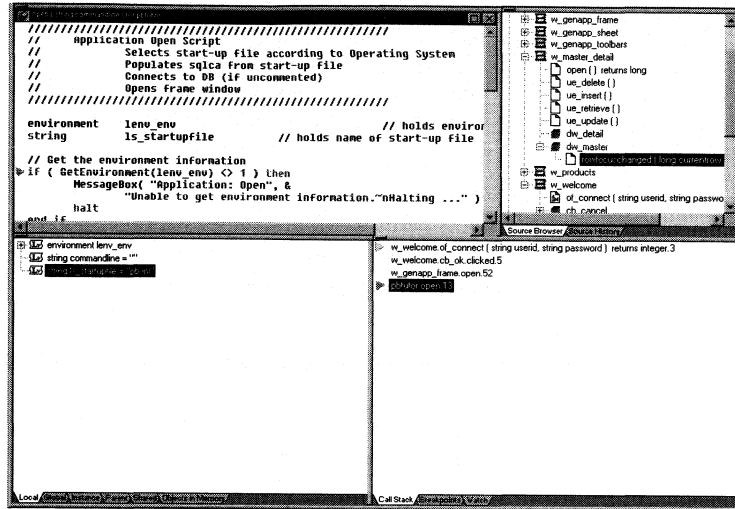
The application resumes execution and continues to the next breakpoint. The `of_connect` function displays in the Source view.

2 Select the *Local* tab in the lower-left pane and the *Call Stack* tab in the lower-right pane.

The yellow arrow in the Call Stack view indicates the current location in the call stack.

3 Double-click another line in the *Call Stack* view.

A green arrow indicates the line you selected, and the Source and Variables views show the context at that location in the call stack. You can try double-clicking other lines to show different contexts.



4 Step through the next few lines of code.

When you resume execution, the Source and Variables views revert to the current context. As you step through each statement, you can check that the values assigned to the local variables are what you expected.



5 Click the *Continue* button in the PainterBar.

The application resumes execution. The database connection is established, and the MDI frame for your application displays. The application is waiting for user input.

6 Select *File>Open>Customers* from the menu bar.

The application continues until it reaches the line in the RowFocusChanged event that contains the next breakpoint you added.

The RowFocusChanged event for a DataWindow occurs before the DataWindow is displayed. For this reason, execution stops before the Customer window is opened.

Set a watch and a conditional breakpoint

Where you are

- Add a breakpoint for the Application object
 - Add breakpoints for the Welcome and Customer windows
 - Run in debug mode
 - Step through the code line by line
 - Stop at breakpoints and examine variables
 - > Set a watch and a conditional breakpoint
-

Next you will set a watch on a variable whose value changes when the user selects a row in the Customer window. Then you will change one of the simple breakpoints you have set into a conditional breakpoint that is triggered only when a variable has a specific value.

1 Select the *Watch* tab in the lower-right pane.

2 Select the *ll_itemnum* variable in the *Local Variables* view and drag it to the *Watch* view.

The *ll_itemnum* variable is set to 101, the ID of the first customer retrieved. Displaying it in the Watch view makes it easier to observe when its value changes. You can also drag Global, Instance, and Parent variables to the Watch view so that you can easily keep track of several variables of different types.



3 Click the *Continue* button.

The application resumes execution. The Customer window displays and shows the list of customers retrieved from the database. The detail DataWindow shows information about customer 101.

4 Select a different row in the *Customer* window.

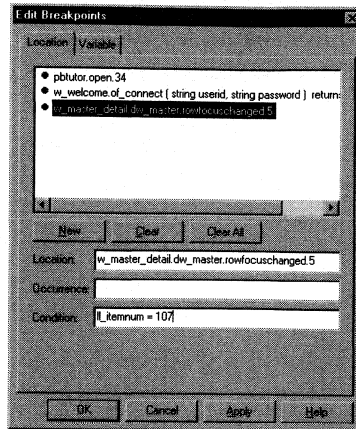
The new value of *ll_itemnum* displays in both the Local Variables view and the Watch view.

5 Select the *Breakpoints* tab in the lower-right pane. Double-click the *rowfocuschanged* breakpoint.

The Edit Breakpoints dialog box opens with the breakpoint in the RowFocusChanged event selected.

6 Type this line in the *Condition* textbox and click *OK*:

```
ll_itemnum = 107
```



The breakpoint in the RowFocusChanged event script is now a conditional breakpoint. PowerBuilder only suspends execution when it reaches this statement *and* the value of ll_itemnum is 107.



7 Click the *Continue* button.

The application resumes execution. Now you can select different rows in the Customer window, and the Debug window will only open at the breakpoint if you select the customer whose ID is 107.

If you select customer 107, click the Continue button again to return to the application.

8 Select *File>Exit* from the application's menu bar.

The application terminates and you return to the Debug window.

9 Select *File>Close* from the menu bar to close the Debug window.

You return to the PowerBuilder initial screen.

Creating the Executable File for the Application

In this chapter you will create an executable version of the application for distribution to users. Users can run this executable version of the application just as they can any other application.

You will:

- ◆ Specify an application initialization file
- ◆ Create the executable file
- ◆ Create an icon
- ◆ Test the executable file

How long will it take?

About 10 minutes.

Specify an application initialization file

Where you are

- > Specify an application initialization file
 - Create the executable file
 - Create an icon
 - Test the executable file
 - What to do next
-

Now you will create an application initialization file (PBTUTOR.INI). An initialization file keeps track of settings that control the appearance and behavior of the application. It can also store default parameters for connecting to the database, which is what you will use it for in this chapter.

When you deploy your application's executable file, the initialization file is among the files you deploy with it, usually in the same directory as the executable file.

This is a two-step process:

- ◆ Create the application initialization file
- ◆ Modify the application's Open event script

Create the application initialization file

First you will create an application initialization file.



- 1 Click the *Edit* button to open the file editor.

- 2 Open the PB.INI file.

You may have to select Initialization files from the Files of Type dropdown listbox.

- 3 Scroll down to the [DATABASE] section.

- 4 Select the [DATABASE] and DBMS lines:

```
[Database]
DBMS=odbc
```

- 5 Select *Edit>Copy*.
Select *File>New*.
Select *Edit>Paste*.

- 6 Select *File>Save As* from the menu bar.
Type PBTUTOR.INI in the File Name box.

Save the file in the same directory where you will save the executable file.

- 7 Close the file editor.

Modify the application's Open event script

Now you will modify the script for the application's Open event to specify the application initialization file.



- 1 **In the Application painter, click the *Script* button in the PainterBar.**

Make sure you are looking at the script for the Open event.

- 2 **Scroll down until you see these lines:**

```
// Select start-up file by operating system
choose case lenv_env.ostype
  case windows!, windowsnt!
    ls_startupfile = "pb.ini"
  case sol2!, aix!, hpux!
    ls_startupfile = ".pb.ini"
  case macintosh!
    ls_startupfile = "PowerBuilder Preferences"
```

- 3 **Modify the appropriate *ls_startupfile* line to read:**

```
ls_startupfile = "pbtutor.ini"
```



- 4 **Click the *Return* button to compile and save the script.**

If any errors are reported, correct them and click Return again.

Create the executable file

Where you are

- Specify an application initialization file
 - > Create the executable file
 - Create an icon
 - Test the executable file
 - What to do next
-

Now you will create the executable file.

- 1 If you have any windows open, close them now.



- 2 Click the *Project* button in the PowerBar.

The Select Project dialog box displays.

- 3 Click *New*.

The New Project dialog box displays.

- 4 Select the *Application* icon and click *OK*.

- 5 Type *pbtutor.exe* in the **Executable File Name** box.

- 6 Make sure the *Machine Code* checkbox is not selected.

About machine code

If you are running PowerBuilder Enterprise, you can choose between Pcode (pseudocode) and machine code.

When you deploy an application to users, you may want to take advantage of the execution speed of machine code. While you are developing the application, you will usually use Pcode because it is faster to generate.

7 Look at (but do not change) the contents of the *Library* box at the bottom of the screen.

The Library box allows you to create dynamic libraries for the application. Dynamic libraries can be used to store the objects in the application. At execution time, objects that are not contained in the application's executable file are loaded dynamically. By using dynamic libraries, you can break the application into smaller units that are easier to manage and also reduce the size of the executable file.

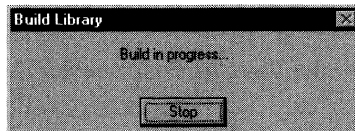
For small applications like the one you're working on now, you do not need to use dynamic libraries. So do not select anything in the Library box at the bottom of the screen.



8 Click the *Build* button in the PainterBar.

PowerBuilder creates an executable file that contains definitions for the objects (such as windows, menus, and DataWindows) in the application. The executable file also includes the bitmap file used in the logon window and the compiled scripts.

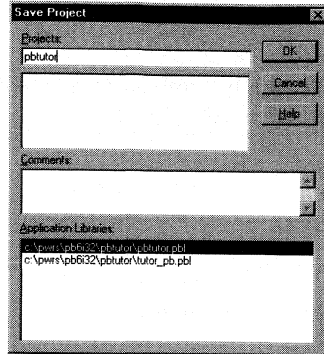
The process of creating the executable file may take a few moments. While PowerBuilder is working, you can look at the status area at the bottom of the screen to see what PowerBuilder is doing. If you wanted to stop the build process, you could click the Stop button in the Build Library popup window. The Build Library window closes when PowerBuilder has finished.



9 Select *File>Save* from the menu bar.

The Save Project dialog box displays.

10 Type *pbtutor* in the *Projects* box.



11 Click *OK* and then close the Project painter.

PowerBuilder saves the new project so you can easily create a new executable version of the application without having to redefine the application components or respecify the options you want. Using a project saves time when you're working on an application that includes dynamic libraries that you expect to rebuild often.

Executables are not binary compatible

While PowerBuilder libraries (PBLs) are binary compatible across platforms, executable files are not. For example, an executable file created on Windows will not run on UNIX. To create an executable file that runs on UNIX, you must copy the PBLs to a UNIX platform and use PowerBuilder for UNIX to create the executable file.

Create an icon

Where you are

Specify an application initialization file

Create the executable file

> Create an icon

Test the executable file

What to do next

Now you will create an icon for the executable file.

How you create the icon depends on the Windows interface you are using:

- ◆ **On Windows 95** You create a shortcut on the desktop.
- ◆ **On Windows NT** You create an icon for a program group.

On Windows 95

1 Minimize PowerBuilder.

PowerBuilder is minimized to an icon on the taskbar.

2 Right-click on the desktop.

Select *New>Shortcut* from the popup menu.

The Create Shortcut dialog box displays.

3 Enter the full path to the EXE file in the *Command Line* box.

For example, if you accepted the default installation folder, type:

`C:\Program Files\Powersoft\PB6\PBTUTOR\PBTUTOR.EXE`

If you're not sure where the executable is, click the Browse button and locate PBTUTOR.EXE.

4 Click Next.

5 Type *SportsWear, Inc.* in the *Select A Name For The Shortcut* box.

6 Click *Finish*.

The icon and description display in the location where you created the shortcut.

Now you must modify a property of the shortcut so that you can run the application.

7 Display the property sheet for the *SportsWear, Inc.* icon.

Select the *Shortcut* tab.

Type the path to the Powersoft shared modules in the *Start In* box.
Click *OK*.

About the location of the Powersoft shared modules

When you install PowerBuilder, the installation process puts the DLLs in a shared folder. The default location is C:\Program Files\Powersoft\Shared.

On Windows NT

1 Minimize PowerBuilder.

PowerBuilder is minimized to an icon at the bottom of the screen.

2 Open the Windows Program Manager.

3 Open the program group where you want the application icon.

4 Select *File>New* from the menu bar.

The New Program Object dialog box displays. The default radio button is Program Item, which is what you want.

5 Click *OK*.

The Program Item Properties dialog box displays.

6 Type the description *SportsWear, Inc.*

7 Enter the full path to the EXE file in the *Command Line* box.

For example, if you are using Windows NT 3.51 and you accepted the default location when you installed PowerBuilder, type:

```
C:\PQRS\PB6\PBTUTOR\PBTUTOR.EXE
```

If you are using Windows NT 4.0 and you accepted the default installation folder, type:

```
C:\Program Files\Powersoft\PB6\PBTUTOR\PBTUTOR.EXE
```

If you're not sure where the executable is, click the **Browse** button and locate PBTUTOR.EXE.

- 8 Enter the full path to the location of the Powersoft shared modules in the *Working Directory* box.**

About the location of the Powersoft shared modules

When you install PowerBuilder, the installation process automatically puts the DLLs in a shared folder.

On Windows NT 3.51. The default folder path is C:\PWRS\SHARED.

On Windows NT 4.0. The default folder path is C:\Program Files\Powersoft\Shared.

- 9 Click OK.**

The icon and description display in the selected program group in the Program Manager.

Test the executable file

Where you are

Specify an application initialization file

Create the executable file

Create an icon

> Test the executable file

What to do next

Now you will test the new executable file.

- 1 **Make sure the PBTUTOR.INI file you created is in the same directory as the executable file.**
- 2 **Double-click the icon for the application.**
The application begins running.
- 3 **Test the application.**
- 4 **When you've finished testing the application, select *File>Exit* from the menu bar.**

What to do next

Congratulations. You have completed the tutorial. Now you know the basics of application development with PowerBuilder.

PowerBuilder books to consult

To further your education, you should continue with the *Feature Guide*, *User's Guide*, and *Application Techniques* books. Also, look at the roadmap at the beginning of this book; the roadmap is a guide to the PowerBuilder documentation.

Finding platform-specific documentation

You use PowerBuilder in much the same way on Windows as you do on UNIX or Macintosh platforms, but there are some differences in look and feel among platforms, and some platform-specific features and limitations. Throughout the documentation, wherever there is a different behavior on a particular platform, you will find notes like this one:

On Windows

The PowerBuilder initialization file is called PB.INI.

Here's where you'll find specific information about using PowerBuilder on Windows:

For this information	Look here
Supported features	For a list of the features supported on each platform, see the matrix of feature support in the <i>Feature Guide</i>
Startup files	For a description of the PB.INI initialization file that each user needs, see the <i>PowerBuilder User's Guide</i>
Database access	The process of connecting to the database in PowerBuilder is identical on all supported platforms For information about which databases you can use, see <i>Connecting to Your Database</i>
Using OLE, DDE, and MAPI	For how to use OLE, DDE, and MAPI, see <i>Application Techniques</i>
Calling external functions	For information about calling external functions, see <i>Application Techniques</i>
Providing online Help	For an overview of how you can develop online Help, see <i>Application Techniques</i>
Deploying the application	For information about deploying the application, see <i>Application Techniques</i>

For this information	Look here
-------------------------	-----------

Developing cross- platform applications	For what to consider when developing an application that will run on different platforms, see <i>Application Techniques</i>
---	--

Installing the
Powersoft Online
Books

All the PowerBuilder manuals are available online and on the Web.

FOR INFO For how to install the Powersoft Online Books, see the
PowerBuilder Installation Guide.

Glossary

accelerator key	<p>In PowerBuilder and InfoMaker, a key used in combination with another key that allows you to select a control or menu item.</p> <p>On Windows 3.1 and Windows 95, these keys are called <i>mnemonic access characters</i> and are used in combination with ALT. On Motif, these keys are called <i>mnemonics</i> and are used in combination with the Meta key. There is no equivalent on the Macintosh.</p> <p>See also shortcut key.</p>
access key	<p>See accelerator key.</p>
ActiveX control	<p>See window ActiveX and Sync ActiveX.</p>
ancestor	<p>An object that defines basic functionality for a class of objects from which descendent objects inherit.</p>
any	<p>A data type that temporarily takes the data type of the value currently assigned to it.</p>
application framework	<p>A collection of interrelated base classes (ancestor objects) from which you inherit the objects you need for an application.</p>
Application object	<p>The entry point into the windows of an application. The Application object is a discrete object that is saved in a PowerBuilder library—just like a window, menu, function, or DataWindow object.</p> <p>The Application object defines application-level behavior, such as which libraries contain the objects used in the application, which fonts are used by default for text, and what processing should occur when the application begins and ends.</p>

asynchronous request	In a distributed application, a function call made by a client that instructs the server to perform processing at a later time. When a client issues an asynchronous function call, it can continue to do other work while the server handles the request.
base class object	<p>The object at the top of a hierarchy (tree structure) of ancestor objects and descendent objects. The base class object typically performs generalized processing, and each descendant modifies the inherited processing as needed.</p> <p>The PowerObject is the base class object for all PowerBuilder objects.</p>
blob	A data type that is used to store an unbounded amount of data (for example, generic binary, image, or large text).
boolean	A data type that is either TRUE or FALSE.
breakpoint	A point in your application code where you want to break (interrupt) normal execution while you are debugging. For example, you might set a breakpoint at a statement that calls a function, so that the application temporarily stops executing when that function is called. A breakpoint can also be called a <i>stop</i> .
cascading menu	Another menu that appears when an item on a dropdown menu is selected. A cascading menu can be nested, so that a menu selection on the cascading menu displays yet another cascading menu.
char	A standard data type that is a single letter, number, or symbol. Can also be spelled out as <i>character</i> .
child DataWindow	See DataWindow child .
child class	A class that is contained within a parent class. Also called a <i>nested class</i> . See also class .
child window	<p>A window that is dependent on a main (parent) window and can exist only within the parent window. The initial position of the child is relative to the parent and not to the workspace. You can move the child within the parent, but not outside the parent.</p> <p>When you move part of the child beyond the parent, PowerBuilder clips the window so that only the portion within the parent is visible. When you move the parent window, the child moves with the parent and maintains the same position relative to the parent.</p> <p>When you minimize a child window, the icon displays within the parent.</p>

class	A definition for a particular type of object. The definition includes the properties, events, functions, structures, and scripts that make up the object. All instances of a given class are identical in form and behavior, but their variables and properties contain different data.
class hierarchy	The relationship among a set of classes, usually represented in a tree structure. A class hierarchy has a single top node (the base class) and can have an unlimited number of levels.
class library	A reusable and extensible collection of classes. In PowerBuilder, classes are called <i>objects</i> (for example, window objects, user objects, and Menu objects).
class user object	<p>An object you build in the User Object painter that can be used to perform processing that has no visual component. To use a class user object, you create an instance of the object in a script and call its functions.</p> <p>You typically use class objects to define business rules and other processing that acts as a unit.</p> <p>There are two kinds of class user object: standard and custom.</p>
code table	A table of display values and corresponding data values used in an edit style. The display values are those the user sees. The data values are those that are stored in the database.
composite report	A report that consists entirely of one or more nested reports. A composite report serves as a container for other reports and has no data source of its own.
computed field	<p>A field in a DataWindow, report, or form consisting of an expression that yields a value calculated on the client at execution time. The expression can use columns, constants, functions, and operators.</p> <p>For example, a DataWindow that includes the Salary and Benefits columns could also include a computed field using the expression <code>salary + benefits</code> to calculate the total remuneration for each employee.</p>
Connection object	The object you create to establish a connection from a client to a server in a distributed application. The Connection object is instantiated on the client and contains several properties.
Context object	A PowerBuilder object that allows applications to access certain host (non-PowerBuilder) services.
control	A graphical object that allows users to interact with your application or that you use to enhance the design of a window.

Controls can also have events that can trigger scripts to perform actions or call functions.

cursor type

The representation of the cursor on the screen.

custom class user object

A class user object you design that encapsulates properties and functions but is not visible to the user.

You define custom class user objects to create units of processing that have no visual component. You typically use custom class user objects to define processing that acts as a unit (such as business rules).

A custom class user object is derived from the PowerBuilder NonVisualObject system object.

custom visual user object

A user object you design that has several controls that function as a unit. (You can think of a custom visual user object as a window that is a single unit and is used as a control.)

For example, if you frequently group controls together in a window and always use the controls to perform the same processing, you can build a custom visual user object that contains these controls and their scripts.

database interface

A connection to your data in PowerBuilder or InfoMaker. You can access data in two ways: through the Powersoft ODBC interface or through one of the Powersoft native database interfaces. (For availability, see your Powersoft product literature.)

database profile

A named set of parameters stored in your PowerBuilder or InfoMaker initialization file that defines a connection to a particular database in the PowerBuilder or InfoMaker development environment.

data source

The source of the data to be retrieved into the DataWindow object.

In PowerBuilder and InfoMaker, the data source sometimes means the method for specifying the data to be retrieved into the DataWindow object. Typically, you specify a data source in the DataWindow painter. These are the five PowerBuilder and InfoMaker data sources:

- Quick Select
- SQL Select
- Query
- External
- Stored Procedure

DataStore

A nonvisual DataWindow that is manipulated entirely through scripts.

data type	<p>A specific type of data. PowerScript has standard data types (such as integer and string) as well as system object data types (such as window) and enumerated data types (such as Alignment). Each DBMS has its own set of data types that correspond to PowerBuilder data types but may not match them exactly.</p> <p>The objects you define in the painters are data types in your application.</p>
data value	<p>A value stored in the database. Compare to display value.</p>
DataWindow child	<p>A DropDownDataWindow in a DataWindow object. A DropDownDataWindow behaves as a child of the DataWindow object that contains it.</p>
DataWindow control	<p>An object you place in a window or in a custom user object that acts as a container for a DataWindow object that has been created in the DataWindow painter.</p>
DataWindow object	<p>An object you create to retrieve, present, and manipulate data from a relational database or other data source (such as an Excel worksheet or a dBASE file).</p> <p>The DataWindow object contains database information that can be viewed and updated by users without using SQL statements.</p>
DataWindow plug-in	<p>A feature that lets you display a Powersoft report (PSR) on a Web page viewed in a browser that supports Netscape plug-ins.</p>
date	<p>A standard data type that is a calendar date in <i>yyyy-mm-dd</i> format where:</p> <p style="padding-left: 40px;"><i>yyyy</i> is the year (a number 1000 to 3000) <i>mm</i> is the month (a number 01 to 12) <i>dd</i> is the day (a number 01 to 31)</p> <p>For example, 1990-01-31 is January 31, 1990.</p>
DateTime	<p>A standard data type that is a date and time in a single data type. DateTime is used only for reading and writing DateTime values to and from a database.</p>
DBMS	<p>Database management system. Software that provides complete database management features.</p>
DDE	<p>Dynamic Data Exchange. A procedure for passing data between Windows applications.</p>

debug mode	An application run mode in which you can insert breakpoints, single-step through a script, and display the contents of variables to help find bugs in your application.
decimal	A standard data type that is a signed decimal number with up to 18 digits. The decimal point can occur anywhere within the 18 digits and is not counted as a digit. The abbreviated form for decimal is <i>dec</i> .
declaration	A specification that associates a name and properties with a variable or a function.
descendant	<p>An object that inherits functionality (properties, variables, functions, and event scripts) from an ancestor object.</p> <p>All objects you define in painters and store in libraries are descendants of PowerBuilder system classes.</p>
display case	The case of the type (upper or lower) used to display data in columns that have a data type of character. The default is Any, the case in which the data was entered.
display format	<p>A mask that determines how data in a database column will be formatted for display. For example, you can display currency values preceded by a dollar sign, show dates with month names spelled out, or use a special color for negative numbers.</p> <p>PowerBuilder comes with many predefined display formats. You can use them as is or define your own.</p>
display value	A value that displays in a form or report. A display value corresponds to a data value stored as specified in a code table. The display value is what the user sees.
distributed computing	<p>Building and running applications in a client/server environment such that:</p> <ul style="list-style-type: none">◆ Business logic is centralized on servers.◆ Application functions are partitioned between the client and the server, thereby reducing the client workload.◆ Applications are more flexible and scalable.

PowerBuilder distributed computing offers a natural way to separate user interface components from the business logic required by an application. In a distributed application, a client can invoke services provided by remote objects. A client can invoke methods (functions and events) that are associated with a remote object just as if they were defined on a local object.

DLL	<p>Dynamic link library. On Windows, a library of software code that is automatically loaded and unloaded as needed. DLL routines can be run from any number of Windows applications. DLLs are also memory efficient. On UNIX and the Macintosh, the equivalent of a DLL is a shared library.</p> <p>In PowerBuilder, you have the option of compiling your application to generate DLLs or shared libraries for deployment.</p>
dot notation	<p>The syntax for identifying objects and controls in order to call their functions and change or obtain information about their properties. The syntax format is:</p> <p style="margin-left: 40px;"><i>object.function</i> <i>control.property</i></p> <p>The syntax can be more complex, including parent objects and properties of other properties.</p>
double	<p>A standard data type that is a signed floating-point number with 15 digits of precision in the range 2.2E-308 to 1.7E+308.</p>
drawing object	<p>A control that has no events and therefore no scripts. There are four drawing objects: Line, Oval, Rectangle, and RoundedRectangle.</p>
drop area	<p>The area within a draggable object in which a control in drag mode can be dropped. When you or the user drags any portion of a draggable control into another draggable control, the control is within the drop area.</p>
dropdown menu	<p>A list of commands or options under a menu item in the menu bar that appears when the user clicks the menu item.</p>
drop lines	<p>The lines on a graph that connect the data points back to the axis.</p>
dynamic linking	<p>The type of linking that occurs when an application uses functions from a dynamic link library and the functions are not included in the executable module. Instead, the executable module contains references to these functions that are resolved at execution time.</p>
dynamic link library	<p>See DLL.</p>

dynamic lookup	Instructs PowerBuilder to wait until execution time to find the specified function or event. Allows you to call functions and events in descendants that don't exist in the ancestor.
edit mask	A format or control used to validate user-entered data and control the display of that data.
edit style	<p>A style that specifies how column data is presented in a DataWindow, report, or form.</p> <p>Named, reusable edit styles are stored in the Powersoft repository. An edit style can be used by any column of the appropriate data type in the database. You can also specify edit styles in the DataWindow, but then they aren't reusable.</p>
encapsulation	Packaging related data and procedures in an object. Through encapsulation, an object contains a collection of variables plus methods (events and functions) for operating on those variables.
enumerated data type	A special data type that provides a set of values specific to the property being set. Enumerated values end with an exclamation point; for example, the Alignment data type has values such as Left! and Right!.
ErrorReturn	An enumerated data type that provides values that identify various PowerBuilder execution-time errors.
event	An action in an object or control that can trigger a script. An event can be triggered by a user action (such as clicking an object or control or entering data) or by execution of a statement in a script.
EXE file	On Windows, the executable file that you create in the Application painter and use to deliver your PowerBuilder application to users.
export name	The default filename created in the Library painter when you export a library entry. The export name has the extension SRx (where x identifies the type of entry).
expression	<p>A combination of values, operators, and functions that can be evaluated. An expression has a value and a data type, which is determined by the expression's components. In most cases, you can use expressions wherever you need a value—for example, on the right side of an assignment statement or as a function argument.</p>

extended attributes	<p>Information about database columns that extends their definitions. Extended attributes include text for labels and headings, display formats, edit styles, validation rules, column widths, heights, and justification. They are defined in the Database or Table painter and stored in special Powersoft system tables in the database itself. PowerBuilder and InfoMaker use extended attributes as default values when creating new DataWindows, reports, and forms.</p>
external visual user object	<p>An object you build that contains controls from objects in the underlying windowing system (external objects) that were created outside PowerBuilder.</p>
floating toolbar	<p>A toolbar that displays on top of the painter workspace and any other windows opened in the painter. It floats above the window and is not within a toolbar dock.</p>
focus	<p>Identifies the control, column, or graphic object that currently can receive input from the keyboard.</p> <p>When a DropDownListBox or ListBox has focus, a dotted rectangle typically surrounds the first item in the box. When a column of a DataWindow object has focus, a dotted rectangle typically surrounds the first row of the column.</p>
foreign key	<p>A column or set of columns that contains primary key values from another table. Each item in the column or columns must correspond to an item in the column of the other table.</p>
form	<p>An electronic document for entering and modifying data in a database. The form can display existing data from the database; you can change or delete the existing data. The form can be blank; you can add new data using a blank form.</p>
function	<p>In PowerBuilder, a routine that performs specific processing. You commonly use functions to act on the objects and controls in your application. PowerBuilder provides a rich assortment of functions (built-in functions). You can also define your own functions (user-defined functions).</p> <p>In object-oriented programming, functions are called <i>methods</i>.</p>
garbage collection	<p>In PowerBuilder, the process of marking objects that are no longer accessible, clearing those objects from memory, and reclaiming memory for objects that the application is actively using.</p>

global function	<p>A user-defined function that is not associated with any object in the application and is always accessible anywhere in the application.</p> <p>Global functions correspond to the PowerBuilder built-in functions that are not associated with an object, such as the mathematical and string-handling functions.</p>
global structure	<p>A structure not associated with any object in your application. You can declare an instance of a global structure and reference the instance anywhere in your application.</p>
global variable	<p>A variable that is accessible anywhere in the application. It is independent of any object definition.</p>
graph	<p>A graphical representation of data. PowerBuilder provides two ways of defining a graph: a graph control in a window or user object that you populate with data and a graph object within a DataWindow that uses the data retrieved from the DataWindow.</p>
IM.INI file	<p>The InfoMaker initialization file on Windows. The file containing the preferences you defined to customize InfoMaker. The equivalent on Macintosh is InfoMaker Preferences. See also INI file.</p>
index	<p><i>In a database.</i> A way to order a table logically to speed the retrieval of data. You index a column or set of columns if information from the columns will be needed frequently. Primary and foreign keys are special examples of database indexes.</p> <p><i>In an array.</i> A number identifying a specific element within the array.</p>
indicator variables	<p>Integers used to identify NULL values or conversion errors after a database retrieval. Specified in the HostVariableList of a FETCH or SELECT statement.</p>
InfoMaker initialization file	<p>See IM.INI file.</p>
inheritance	<p>The feature that enables you to build windows, user objects, and menus that are derived from existing objects. Using inheritance has a number of advantages:</p> <ul style="list-style-type: none">◆ When you change an ancestor object, the changes are reflected in all the descendants.◆ The descendant inherits the ancestor's scripts.◆ You get consistency in the code and objects in your applications.

inheritance hierarchy	An object and all its ancestors.
INI file	On Windows, an initialization file that stores variables that define your preferences. See also PB.INI file and IM.INI file .
instance	An object, rather than the definition of the object (called the <i>object class</i>). An object instance exists in memory and has values assigned to its properties and variables. Object instances exist only when you run an application.
instance variable	<p>A variable that belongs to an object and is associated with an instance of that object (you can think of it as a property of the object). Instance variables have access keywords that determine whether scripts of other objects can access them.</p> <p>Instance variables can belong to the application, a window, a user object, or a menu.</p>
instantiate	In object-oriented programming, to create an object (an instance) of a specific class.
integer	A standard data type that is a 16-bit signed integer, in the range -32768 to +32767. The abbreviated form for integer is <i>int</i> .
keyword	A word used in PowerScript syntax that is not a variable. In most situations, keywords are reserved words.
library	<p>Where PowerBuilder and InfoMaker store objects. A library is a PBL file (it has the extension PBL in DOS).</p> <p>A library is a collection of compiled object definitions (compiled binary representation) and source objects (including scripts) stored in the same location. The name of the library identifies the location.</p> <p>The PowerBuilder and InfoMaker painters store the following objects in libraries: applications, DataWindows, forms, functions, menus, pipelines, projects, proxies, queries, reports, structures, user objects, windows.</p> <p>See PBL file.</p>
library search path	A list of the libraries that contain entries that can be used by scripts in the application. 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.

local variable	A temporary variable that is accessible only in the script in which you define it. When the script is finished, the variable ceases to exist.
long	A standard data type that is a 32-bit signed integer in the range -2,147,483,648 to +2,147,483,647.
machine code	A platform-specific language that produces a fast executable. Compare to pseudocode .
main window	<p>A standalone window that can be independent of all other windows.</p> <p>You use the main window as the anchor for your application. The first window your application opens is a main window (unless you are building an MDI application, in which case the first window is an MDI frame).</p>
MAPI	Message Application Program Interface. A set of Microsoft functions that you can use to add messaging to your application (make your application mail-enabled) on Windows. The PowerBuilder mail functions use MAPI.
MDI application	<p>Multiple Document Interface application. An application in which users work within a frame window that lets them perform activities on multiple sheets of information. This is useful in applications where users require the ability to do several different things at a time.</p> <p>An MDI frame is a window with several parts: a menu bar, a client area, sheets, and (usually) a toolbar plus a status area that can display MicroHelp. On the Macintosh, the whole desktop is the MDI frame.</p> <p>An MDI sheet is a window that can be opened in the client area of an MDI frame. You can use any type of window except an MDI frame as an MDI sheet.</p> <p>Compare to SDI application.</p>
menu bar	A bar at the top of a window that displays the first level of menu items the user can select in the window or application.
menu-level variable	An instance or shared variable, SQL cursor, or SQL procedure belonging to the menu. See scope .
method	<p>In object-oriented programming, the processing that an object performs.</p> <p>In PowerBuilder, methods are <i>functions</i> or <i>events</i>.</p>
MicroHelp	A brief description of a menu item or toolbar icon that displays in the status bar at the bottom of the window when you click the item or icon. The MicroHelp displays as long as you press the mouse button.

nested class	A child class.
object	<p>In object-oriented programming, a self-contained module of data and its associated methods.</p> <p>In PowerBuilder, an object is usually a graphic entity that can be saved in a library (an application, DataWindow object, window, menu, or user object). But some objects (such as the Transaction object, custom class user objects, and structures) have no visual aspect.</p> <p>Each object has a set of properties that describe its appearance and behavior. You can use functions, statements, or the assignment operator to test and set the properties of an object. You change the appearance of an object by setting the properties of the object. You obtain information about the object by testing the properties.</p> <p>Objects also have events that can trigger scripts to perform actions or initiate functions.</p> <p>An object's definition, as stored in a library, is called a <i>class</i>. When you set properties in a painter, you are changing the class. In a running application, an object is instantiated so that it exists in memory.</p> <p>When scripts call functions and set properties for the object, they affect the object instance, not the original definition.</p>
object-level function	A function associated with a particular type of window, menu, or user object—or with the Application object itself. System objects have built-in object-level functions. You can also define your own. A user-defined object-level function is part of the object's definition and can always be used in scripts for the object itself. You can also choose to make a user-defined function accessible to other scripts.
object-level structure	A structure associated with a particular type of window, menu, or user object—or with the Application object. An object-level structure can always be used in scripts for the object itself. You can also make an object-level structure accessible from other scripts.
OCX	An OLE custom control. An OCX encapsulates functionality typically provided by an application and makes it available to a client application via properties, functions, and events.
ODBC	Open Database Connectivity. A standard API (application programming interface) developed by Microsoft. It allows a single application to access a variety of data sources for which ODBC-compliant drivers exist. The application uses SQL as the standard data access language. PowerBuilder and InfoMaker can use ODBC to connect to most DBMSs.

OLE	Object Linking and Embedding. Allows you to use other programs (server applications) and their data in your Windows application. Their data remains in the server application's own format. When your application includes an OLE object, it can invoke the server application to manipulate the object. See also OCX .
overloading	A function name that has more than one definition. Each version of the function has a different argument list, which is how they are distinguished. See also overriding .
overriding	A function in a descendent object that replaces a function of the same name in its ancestor object. The two functions have the same argument list. See also overloading .
painter	<p>In PowerBuilder, any of multiple special windows with tools for building and using your objects (forms, queries, and reports) and managing your data and working environment.</p> <p>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 textboxes.</p>
PainterBar	The toolbar containing the icons associated with the current painter. The icons in this toolbar depend on the painter, and you use them to initiate actions and activities associated with the painter.
parent object	The object that contains the current object. For example, a window is the parent object of the controls it contains.
PBD	<p>PowerBuilder dynamic library. A library that is loaded when the application executes. At that time, PowerBuilder looks for any PBDs you specified in the executable.</p> <p>Any PBDs containing objects that will be dynamically loaded during execution must be included in your application's path.</p>
PB.INI file	The PowerBuilder initialization file on Windows. The file containing the preferences you have defined to customize PowerBuilder. The equivalent on UNIX is .pb.ini, and the equivalent on the Macintosh is PowerBuilder Preferences. See also INI file .
PBL file	PowerBuilder library file. Pronounced <i>pibble</i> . See also library .

PBR file	PowerBuilder resource file. An ASCII file with the extension PBR. A PBR file contains the name of each file in which a resource that is assigned dynamically in scripts in the application is stored. These resources can include bitmap, icon, and cursor files.
PBU	PowerBuilder unit. A PBU is defined in logical inches. The size of a logical inch is defined by your operating system as a specific number of pixels; the number is dependent on the display device. PBUs allow you to build applications that look the same on all terminal screens.
pcode	See pseudocode .
pipeline	<p>An object you create to reproduce data within a database or across databases. You create pipelines in the Data Pipeline painter.</p> <p>Some of the ways you can use pipelines is to pipe data from one or more tables to a table in the same or a different DBMS; pipe corporate data from a database server to a SQL Anywhere database; pipe data to a new table when a change is disallowed in the Table painter.</p>
plug-in	See DataWindow plug-in and PowerBuilder window plug-in .
pointer	The image on the screen that indicates where the mouse is pointing. You can assign different images to the pointer.
polymorphism	Two or more functions in the same object with the same name but different argument lists (different signatures). PowerBuilder determines which function to call by comparing the data types of the arguments in the call to those in the prototype. Polymorphism should be used with care to avoid ambiguities in choosing the function to be called.
popup menu	<p>A menu that displays only after a specified event occurs. To display a popup menu, you must create a script to display the menu and use the PopMenu function to specify where the menu will display on the screen.</p> <p>A menu bar item does not require a script to display the dropdown menu under it.</p>
popup window	A window that displays only in response to an event within a window but can exist outside the window and, in some cases, after the window that opened it is closed.

The popup window may or may not have a parent window. When it has a parent window, it is dependent. The popup is hidden when the parent is minimized and visible when the parent is maximized (it is never overlapped by the parent). When the popup window does not have a parent, it is independent and can be overlapped by the window that opened it.

When you minimize a popup window, the icon for the popup displays outside the window (regardless of whether it has a parent).

On the Macintosh, the equivalent of a popup window is a modeless dialog.

position pointer

The starting point of the next write in a disk file. The position pointer is unique for each user of a file and is advanced automatically after every read or write.

PowerBuilder initialization file

See **PB.INI file**.

PowerObject

The base class from which all PowerBuilder system objects are descended.

PowerPanel

A response window that 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.

PowerBuilder window plug-in

A PowerBuilder feature that lets you display a PowerBuilder child window on Web pages viewed in a browser that supports Netscape plug-ins. This window can contain all window controls and can open other (popup or response) windows.

PowerScript

The PowerBuilder language. You use PowerScript in scripts and user-defined functions to build PowerBuilder applications.

PowerScript is a high-level, object-oriented language designed for the PowerBuilder environment. It contains commands (functions) for manipulating the graphic objects in an application, performing calculations, processing character strings, converting data from one data type to another, and other processing.

PowerTip

Text displayed whenever the pointer pauses over a toolbar button that tells you what the button is for.

primary key

A column or set of columns that uniquely identifies each row in a table.

For example, an Employee table could contain two employees with the same first name and last name. To prevent confusion, an Employee table usually has a column called Emp_id containing a unique ID number for each employee. You define the Emp_id column as the primary key column. Each row in the Employee table has one employee ID and contains data for the one employee with that ID.

profile

A report containing timing and other information about the execution of an application. Several different profiles can be created using PowerScript functions or the Application Profiler.

project object

An object that lets you specify how the executable version of an application is to be generated. It includes options for compiling code, creating dynamic libraries, and bundling resources.

property

A value that defines the characteristics of a PowerBuilder object or control, such as its size, position, color, or tab sequence.

proxy object

A local representation of a remote user object in a distributed application. Proxy objects are retained for backward compatibility with PowerBuilder 5.0.

In PowerBuilder 5.0, each remote object contained in a server application had a corresponding proxy object in the client application. When you saved a remote object in a server application, PowerBuilder saved a proxy name that served as an alias for the user object.

pseudocode

An interpreted language that is supported on all PowerBuilder platforms. Pseudocode can be generated quickly and produces a compact executable. Compare to **machine code**.

query

In PowerBuilder, a query is a named SQL SELECT statement. You can create and save queries with the Query painter and then use them repeatedly to specify data requirements.

real

A standard data type that is a signed floating-point number with 6 digits of precision in the range 1.17E-38 to 3.4E+38.

reference

A handle used to access an object instance. Can be assigned to a variable of the appropriate type. Similar to a point but unable to do pointer arithmetic.

remote object

In a distributed application, a custom class (nonvisual) user object contained in an application located on a remote server. A client can invoke functions that are associated with a remote object just as if they were defined on a local object.

remote procedure call	See RPC .
RPC	A method for accessing a non-result-set database stored procedure by declaring the stored procedure as an external function or subroutine.
repository	A set of five system tables maintained by PowerBuilder where extended attribute definition information related to columns (such as display formats, validation rules, and font information) is stored. When you create a table, you can define extended attributes for the table columns (such as column heads or an edit style for the column).
reserved word	A word that has a special meaning to the compiler. You cannot use a reserved word as an identifier.
response window	<p>The window that displays to request information from the user and to which the user must respond. A response window is always opened from within another window (its parent). Typically, a response window is opened after a specific event occurs in the parent window. On the Macintosh, the equivalent of a response window is a modal dialog.</p> <p>A response window is application modal: when it displays, it is the active window (it has focus) and no other window in the application is accessible until the user responds to it. The user can go to other Windows applications; but when the user returns to the application, the response window is still active.</p> <p>A response window acts like a modal popup window.</p>
result set	The rows of data retrieved from the database when a SQL SELECT statement is executed.
return code	A return value, usually a positive or negative integer, that is returned by an event to indicate to the system what action should be taken next.
scope	Where a variable or function name is recognized. Names can be recognized at the local, global, or object level. Object variables or functions can be recognized outside the object, within descendent objects, or only within the object itself.
script	<p>A collection of statements that reflect the processing that will occur in the window or application when a particular event is triggered.</p> <p>You write scripts using PowerScript, the PowerBuilder language. Scripts consist of PowerScript statements, functions, and declarations that perform processing in response to an event.</p>

SDI application	<p>Single Document Interface application. An application in which users work with one main window at a time to perform an activity (although that window may display various popup or child windows to do supporting chores as a user works). When users want to perform a different kind of activity, they go to a different main window to do it.</p> <p>SDI may be appropriate for building an application that is very simple, especially if it deals with only one kind of database and the user needs to perform only one operation at a time.</p> <p>Compare to MDI application.</p>
secure mode	<p>A feature that helps to ensure that PowerBuilder applications downloaded over the Internet will not damage a client system or access information on the client system. Secure mode severely restricts the PowerScript functions that can be called by an application. Secure mode is implemented through separate versions of the PowerBuilder window plug-in and window ActiveX.</p>
server push	<p>In a distributed application, a technique that allows the server to send messages back to the client. This is particularly useful when the client needs to be notified of the completion of an asynchronous request.</p>
shared library	<p>A library of code that makes functions available to an application. On different platforms, shared libraries go by different names. In Windows, shared libraries are called DLLs.</p>
shared object	<p>In a distributed application, a nonvisual user object that is shared by multiple client connections. Shared objects provide convenient access to common data that would otherwise need to be retrieved separately by each client connection.</p>
shared variable	<p>A variable that belongs to an object definition and exists across all instances of the object. Shared variables retain their value when an object is closed and opened again.</p> <p>Shared variables are always private. They are accessible only in scripts for the object and for controls associated with the object.</p> <p>Shared variables can belong to the Application object, a window, a user object, or a menu.</p>
shortcut key	<p>A key or combination of keys that provides a quick way to perform certain operations (such as CTRL+INS).</p>

On Windows 3.1 and Windows 95, these keys are called *shortcut keys* or *accelerator keys*. On Motif, they are called *accelerator keys*. On the Macintosh, they are called *keyboard equivalents*.

See also **accelerator key**.

SQL

Structured Query Language. Pronounced *sequel*. A language used to retrieve and manipulate data in a relational database. You can use SQL commands interactively in PowerBuilder and InfoMaker to work with a database. You can also embed SQL commands in PowerScript to communicate with a database.

SQLCA

SQL Communications Area. The default Transaction object in PowerBuilder.

standard class user object

A user object that 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.

Once you have defined the 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.

standard visual user object

An object you build starting from a standard PowerBuilder control. Such an object inherits its properties and events from the standard PowerBuilder control; you modify it to perform processing specific to your application.

static linking

The type of linking that occurs when the functions from a static library are referenced by an application become part of the executable module.

static lookup

Instructs PowerBuilder to find the specified function or event at compile time.

string

A standard data type that is characters enclosed in single (') or double (") quotation marks, including a string of 0 length (the empty string ""). The maximum number of characters in a string is 60,000.

structure

A collection of one or more related variables of the same or different data types grouped under one name. With a structure, you can refer to related objects as a unit rather than individually. For example, you could define the user's ID, address, access level, and a picture (bitmap) of the user as a structure called `user_struct`.

In some languages (including Pascal and COBOL), a structure is called a *record*.

style	The appearance of a PowerBuilder window or control (such as its size, position, color, or tab sequence). A window or control's style is determined by its properties.
StyleBar	A toolbar for changing text fonts, font size, style (bold, italic, and underline), and alignment (left, centered, and right).
Sync ActiveX	A component of the Synchronizer that you use to provide synchronization in a window of a 32-bit application or on a Web page. The Sync ActiveX executes instructions in a Sync data file.
Sync data file	A text file that contains synchronization instructions. Each component of the Synchronizer (Sync Builder, the Sync runtime executable, and the Sync ActiveX) executes synchronization instructions found in a Sync data file. You can create and run Sync data files using Sync Builder.
Sync runtime executable	A component of the Synchronizer that causes synchronization to occur. When the Sync runtime executable runs, it can execute one synchronization instruction that you provide on the command line or many synchronization instructions in a Sync data file. See also synchronization .
synchronization	In general, arranging items in a master set and a target set so that the items in each set exist in parallel, occur simultaneously, or operate in unison. For PowerBuilder developers, using the Sync runtime executable to keep deployed application files (the target set) refreshed with the latest files (the master set) or using the Sync ActiveX in the application window to refresh files.
synchronous request	In a distributed application, a function call made by a client that instructs the server to perform processing immediately. Synchronous requests cause the client to wait until processing is completed.
system class	A class defined by PowerBuilder. An object you define in a painter is a descendant of a system class, even when you don't explicitly choose to use inheritance for the object you define.
table	The organizing mechanism in a relational database. A table contains rows and columns of data. Each row represents a single entry. Each column represents a category of data.
tag value	A text string you associate with a control or a DataWindow column. You can use the tag for your own purposes in the application.
time	A standard data type that is time in 24-hour format: <i>hh:mm:ss.ffffff</i> where:

hh is the hour (a number 00 to 23)
mm is the minutes (a number 00 to 59)
ss is the seconds (a number 00 to 59)
ffffff is the microseconds (a number 000000 to 999999)

The colons (:) and period (.) are required. Blanks are not allowed.

For example, 23:59:59.999999 is one microsecond before midnight.

PowerBuilder and InfoMaker use the 24-hour time format:

hh:mm:ss:ffffff (hours:minutes:seconds:microseconds)

trace file

A binary file containing monitoring information that is generated when you run an application with profiling enabled. You use PowerScript functions or the Application Profiler to create an execution profile of the application from the data in the trace file.

Transaction object

The communications area for a database connection that facilitates communication between a script and the database. It stores status information for recent database activity. The default Transaction object in PowerBuilder is SQLCA (SQL Communications Area).

Transport object

The object you create to receive a connection on a server in a distributed application. The Transport object is instantiated on the server and contains several properties, including the name of the connection driver being used and the application name.

type promotion

The automatic conversion of values to compatible, higher-ranking data types. Type promotion occurs when expressions are evaluated.

UnsignedInteger

A standard data type that is a 16-bit unsigned integer in the range 0 to 65535. The abbreviated form for UnsignedInteger is *unsignedint* or *uint*.

UnsignedLong

A standard data type that is a 32-bit unsigned integer in the range 0 to 4,294,967,295. The abbreviated form for UnsignedLong is *ulong*.

user event

An event that you define and trigger from other scripts.

user object

An object you build in PowerBuilder to perform processing that you use frequently in your applications. User objects are reusable, and you can use them in windows and in other user objects the same way you use the PowerBuilder controls.

Some types of user objects are:

- Class user object
- Custom class user object
- Custom visual user object

External visual user object
Standard class user object
Standard visual user object

validation rule	<p>An expression used to ensure that data entered into a column is valid for the column.</p> <p>The database ensures that all data is the correct data type and size for a column. You use a validation rule to ensure that the data meets additional criteria. For example, you could use a validation rule to ensure that data in the numeric column is equal to or greater than a specified amount.</p>
VBX user object	<p>An object that is compatible with Microsoft Visual Basic and can be used in PowerBuilder applications on Windows. You can build or purchase the DLLs (dynamic link libraries) that define the VBX controls.</p>
visual user object	<p>A reusable control or set of controls with a certain behavior. For example, a visual user object could consist of several buttons that function as a unit. The buttons could have scripts associated with them that perform standard processing. Once the object has been defined, you can use it as often as you need.</p>
window ActiveX	<p>A PowerBuilder feature that lets you display a PowerBuilder child window on Web pages viewed in a browser that supports ActiveX. This window can contain all window controls and can open other (popup or response) windows. The window ActiveX provides an interface to the PowerBuilder application through a set of events and functions that you access via JavaScript or VBScript.</p>
window-level variable	<p>An instance or shared variable, SQL cursor, or SQL procedure belonging to a window.</p>
.WindU	<p>An initialization file required by PowerBuilder for UNIX to contain settings for Wind/U, a product from Bristol Technology used to implement PowerBuilder in the UNIX Motif environment.</p>

Index

A

- accelerator keys 95
- aligning columns in DataWindow objects 162
- ampersand (&) accelerator key 95
- ancestor
 - objects 34
 - scripts 111
 - windows 105
- application framework 34
- Application object
 - creating 31
 - current 34
 - definition 8
 - icon 46
 - library search path 48
 - Open event for 83
- applications
 - building 191
 - debugging 177
 - definition 8
 - frameworks for 34
 - MDI 24
 - running 50
 - templates for 34
- attributes, extended 78

B

- background color, default to 3D window option 56
- bitmap, for tutorial 28
- buttons, showing hidden 50

C

- class user objects 14
- Clicked event, CommandButton control 86, 89
- Close events in Application object 10

- ColorBar, placement and display 35
- columns on DataWindow objects
 - aligning 162
 - rearranging 161
- comments
 - DataWindow 142, 159
 - in scripts 72
 - menu 103, 123
 - window 68, 129, 130
- COMMIT statement 117
- CONNECT statement 88
- context-sensitive online Help 42, 73
- controls
 - CommandButton 65
 - deleting 58
 - duplicating 60
 - Picture 58
 - specifying properties for 61
 - StaticText 60
- current application 34

D

- data source
 - Quick Select 138
 - SQL Select 151
- data types, naming conventions 87
- database
 - connecting to 75
 - Powersoft Demo, setting up 28
- database connectivity
 - about 79
 - Transaction object 82
- database management system (DBMS) 3
- Database painter, using 77
- database profiles 79
- databases
 - connecting to 7
 - connecting to at execution time 82

- extended attributes 78
- native interfaces 7
- retrieving, presenting, and manipulating data 11
- table definitions in 77
- DataWindow controls 107
- DataWindow data expressions 121
- DataWindow objects
 - about 137
 - aligning columns 162
 - attaching to DataWindow controls 146, 166, 172
 - creating 138, 151
 - data source 138
 - display order 139
 - enhancing 160
 - overview 11
 - presentation style 138
 - previewing 141, 157, 163
 - rearranging columns 161
 - retrieval argument 154
 - saving 142, 159
 - selecting columns with Quick Select 139
 - WHERE clause 155
- DataWindow painter
 - bands 140
 - buffering retrieved rows 141
 - preview 141, 157, 163
 - retrieval argument 154
 - WHERE clause 155
- DBError event 111
- DBMS 3
- debugging
 - about 177
 - adding breakpoints 180
 - adding stops 178
 - running in debug mode 182
 - stepping through code 184
 - stopping at breakpoints 186
- DeleteRow function 118
- deleting controls 58
- detail band in DataWindow objects 140
- dropdown menus
 - adding items 95
 - description 13

E

- ellipsis points in menus 13
- events
 - about 6
 - Close 10
 - DBError 111
 - Open 10
 - RowFocusChanged event 120
 - triggering from menu scripts 124
 - see also* Open event
- executable file
 - application icon 46
 - creating 195
 - generating machine code 195
 - regenerating objects 195
- extended attributes 78

F

- files 28
- floating toolbar 36
- footer band in DataWindow objects 140
- Freeform presentation style
 - columns 160
 - DataWindow definition 151
- functions
 - about 6
 - context-sensitive online Help 73
 - DeleteRow 118
 - GetActiveSheet 124
 - global 13
 - InsertRow 117
 - object-level 13
 - Open 73
 - OpenSheet 132, 133
 - ParentWindow 132, 133
 - ProfileString 83
 - Reset 117
 - Retrieve 116
 - SetFocus 116, 117
 - SetPointer 89
 - SetRowFocusIndicator 116
 - SetTransObject 119
 - Trim 88
 - Update 117

G

GetActiveSheet function 124
global structures 14

H

HALT statement 90
header band in DataWindow objects 140
hyphen (-) separator character 99

I

icons, application 46
inheritance
 and application framework 34
 and object-oriented programming 6
 and user objects 107
 and windows 105, 106, 127
 in Menu painter 98
InsertRow function 117

L

libraries
 about 15
 creating 32
 search path for application 48
Library painter, comments 142
local database 3

M

MDI applications 24
MDI MicroHelp 100
MDI sheets 98, 135
menu bars, description 13
menu items
 adding scripts for 124
 defining 95
 description 13

Menu painter
 accelerator keys 95
 menu items 95
 opening 94, 124
menus
 about 6, 12
 adding scripts for 124
 and toolbars 101
 attaching to windows 122
 creating 98
 inheriting 98
 menu items 95
 previewing 97, 104
 saving 103
 separator lines in 99

N

naming conventions 87

O

object-level structures 14
objects
 about 6
 Application 8
 inheritance 98
online documentation *see* Powersoft Online Books
online Help
 context-sensitive for functions 73
 using 39
Open event
 and Application object 10
 script generated for 50
Open function 73
OpenSheet function 132, 133

P

PainterBar
 placement and display 35
 popup menu 36
 PowerScript painter actions 71

- using 17
- painters 16
- Parent (pronoun) 90
- ParentWindow example 132, 133
- PBL *see* PowerBuilder Library (PBL)
- popup menus
 - about 36
 - PainterBar 36
- PowerBar
 - placement and display 35
 - using 16
- PowerBuilder
 - initialization file 77, 83
 - preferences file 83
- PowerBuilder Library (PBL) 15
- PowerPanel 17
- PowerScript
 - about 6
 - HALT statement 90
- PowerScript painter
 - compiling a script 74
 - error window 74
 - list of actions 71
- Powersoft Demo Database
 - connecting to 75
 - setting up 28
- Powersoft Online Books 44
- PowerTips
 - about 35
 - using 17
- presentation styles
 - DataWindow object 138
 - Freeform 151
 - Tabular 138
- previewing DataWindow object 141, 157, 163
- ProfileString function 83
- projects
 - about 15
 - building 195
- pronoun
 - Parent 90
 - This 119

Q

- queries 13
- Quick Select
 - sort criteria 140
 - using 138

R

- Reset function 117
- retrieval argument
 - creating 154
 - WHERE clause 155
- Retrieve function
 - about 116
 - specifying an argument for 121
- right mouse button, and popup menus 36
- ROLLBACK statement 117
- RowFocusChanged event 120, 147

S

- scripts
 - about 6
 - comments in 72, 84
 - compiling 74
 - displaying ancestor scripts 111
 - error window 74
 - for user events 116
- scrollbars, vertical 109
- Select painter
 - about 152
 - tab area 153
- SELECT statement 153
- SetFocus function 116, 117
- SetPointer function 89
- SetRowFocusIndicator function 116
- SetTransObject function 119
- setup for tutorial 28
- sheets, in an MDI application 98
- shortcut keys 95
- SQL Anywhere 3
- SQL painter 151
- SQL Select 151
- SQL statements

COMMIT 117
CONNECT 88
ROLLBACK 117
SELECT 153
SQL syntax, in Select painter 153
SQLCA (SQL Communications Area) 82, 119
structures 14
StyleBar
 placement and display 35
 using 17
summary band in DataWindow objects 140
SystemError event 10

T

tab area 153
tab order 67
Tabular presentation style 138
This (pronoun) 119
toolbars
 buttons on 101
 controlling 35
 moving 36
 showing text 35
Transaction object 82, 119
Trim function 88
tutorial setup 28

U

Update function 117
user events
 about 113
 adding scripts for 116
 defining 113
 triggering from menu scripts 124
user objects
 about 14
 using 107

V

variables, naming conventions 87
vertical scrollbars 109
visual user objects 14

W

WHERE clause 155
Window painter, deleting a control 58
windows
 about 6, 10
 ancestor 105
 attaching menus to 122
 CommandButton controls on 65
 controls on 6
 creating 54
 DataWindow controls on 107
 deleting a control 58
 descendent 106, 127
 inheriting 106, 127
 opening in a script 70
 Picture controls on 58
 previewing 69
 response 54
 saving 68, 123, 129, 130
 StaticText controls on 60
 tab order in 67