Getting Started

Appeon PowerBuilder® 2019 R3

Contents

1 Welcome to PowerBuilder

This part is an overview of the PowerBuilder development environment.

1.1 Introduction to PowerBuilder

About this chapter

This chapter introduces the PowerBuilder development environment, which you use in the tutorials in Parts 2-4. It also describes the building blocks of a PowerBuilder application.

For more information

For a more detailed description of the PowerBuilder development environment, see Chapter 1, *The PowerBuilder Environment* in *Users Guide*.

1.1.1 What PowerBuilder is

PowerBuilder is an enterprise development tool that allows you to build many types of applications and components. It is one of a group of Appeon products that together provide the tools to develop client/server, multitier, and Internet applications.

What's in a PowerBuilder application?

A PowerBuilder client application can contain:

- 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 client 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

PowerScript provides a rich assortment of built-in functions that can 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.

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

Object-oriented programming with PowerBuilder

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 encapsulation, inheritance, and polymorphism, you can get the most out of each object you create, making your work more reusable, extensible, and powerful.

Multitier applications

PowerBuilder lets you build applications that run in a distributed computing environment. A multitier application lets you:

- Centralize business logic on servers, such as JBoss, WebLogic, WebSphere, or COM+

- Partition application functions between the client and the server, thereby reducing the client workload

- Build scalable applications that are easy to maintain

For information about multitier applications, see Chapter 6, *Developing Distributed Applications* in *Application Techniques*.

Database connectivity

PowerBuilder provides easy access to corporate information stored in a wide variety of databases. Data can be accessed through the PowerBuilder ODBC or JDBC interfaces, through a middle-tier data access server like the SAP DirectCONNECT server, or through a native or direct connection to a database.

For information on database connectivity, see Chapter 4, *Working with Database Connections* in *Connecting to Your Database*.

Online Help and documentation

PowerBuilder Help can be accessed using Help buttons and menu items, or by selecting the F1 key from anywhere in PowerBuilder. There are jumps in several places from the Help to books in HTML format. Manuals are also available on the Appeon website.

1.1.2 The PowerBuilder environment

Workspaces and targets

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

A PowerBuilder target can be one of several types:

- Application target

  A client/server or multitier executable application.

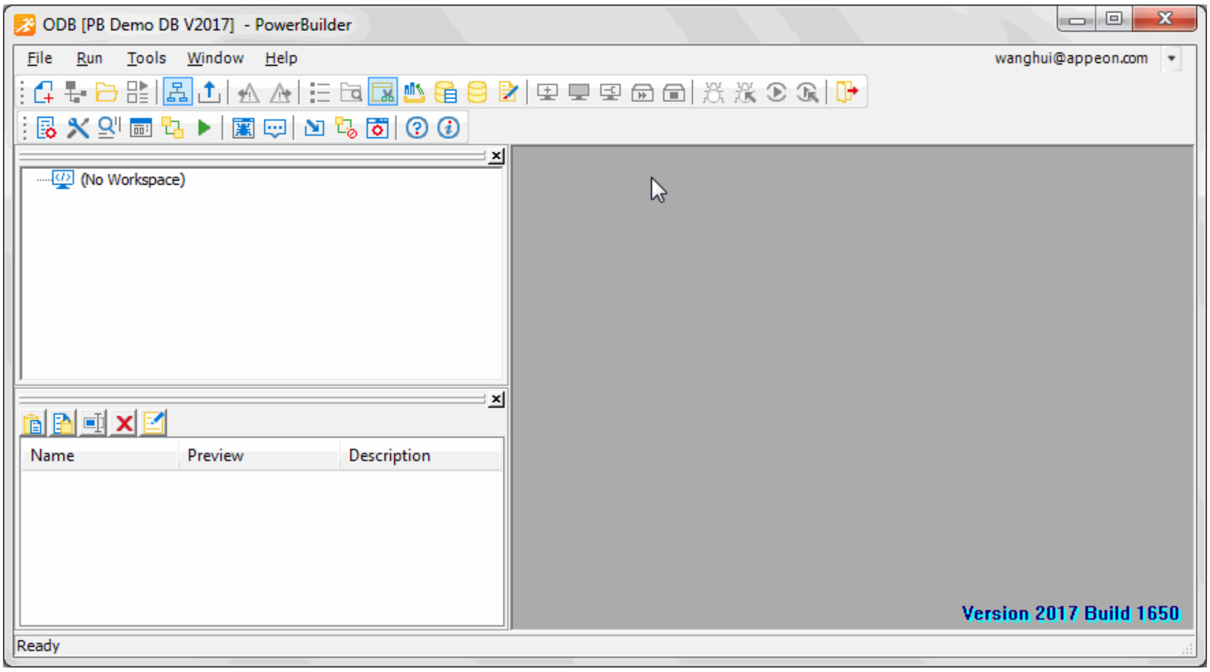
- .NET target (obsolete)

  A .NET target that you can use to deploy nonvisual custom class components as .NET assemblies or Web services.

The first lesson in the tutorial shows you how to create a workspace and an Application target. Later you learn how to create .NET targets.

The development environment

When you start PowerBuilder, it opens in a window that contains a menu bar and the PowerBar at the top, and the System Tree and Clip windows on the left.



System Tree

The System Tree window can serve as the hub of your development activities. You use it to open, run, debug, and build your targets, and for drag-and-drop programming.

Clip window

The Clip window lets you store code fragments that you use frequently.

Output window

The output of a variety of operations (migration, builds, deployment, project execution, object saves, and searches) displays in an Output window at the bottom of the main window. The Output window opens automatically when output information is generated, but you can open the Output window at any time by clicking the Output window toolbar button.

Painters

Once you have created a workspace and a PowerScript target, you build the components of the target using painters. Painters provide an assortment of tools for enhancing and fine tuning the objects in a target.

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.

Wizards

Wizards simplify the creation of applications, objects, components, websites, and Web pages.

Design-time controls

Design-time controls (DTCs) create basic HTML and scripts from information you provide in property sheets. The property sheets display when you drop a DTC on a Web page in the HTML editor.

To-Do List

The To-Do List displays a list of development tasks you need to do for the current target. Entries on the To-Do list can be created automatically by most PowerBuilder wizards. You can also type in entries or import them from a text file and then link them to a task that you want to complete.

Browser

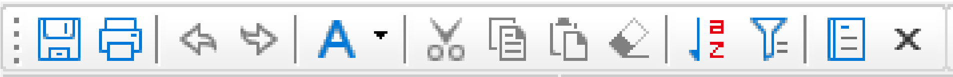
The Browser lets you see all the objects, methods, variables, and structures that are defined for or available to your PowerScript target. Objects in the Browser can be displayed in alphabetic or hierarchical order or filtered by the object name. The Browser displays methods with their complete prototypes (signatures), which include the datatypes of all arguments and return values.

PowerBar

The PowerBar displays when you begin a PowerBuilder session. The PowerBar is the main control point for building PowerBuilder applications. You can use the New, Inherit, or Open buttons on the PowerBar to open all of the PowerBuilder painters. From the PowerBar, you can also open the Browser, debug or run the current application, and build and deploy the workspace.

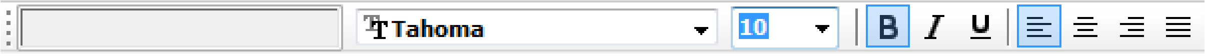
PainterBar

When you open a painter or editor, PowerBuilder displays a new window that has a workspace in which you design the object you are building. PowerBuilder also displays one or more PainterBars with buttons that provide easy access to the tools available in the painter or editor. For example, here is the PainterBar for the DataWindow painter.



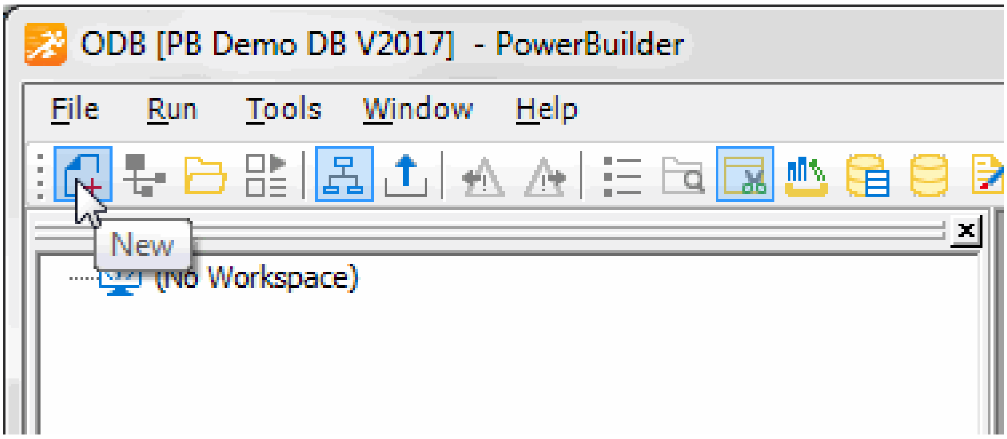
StyleBar

The StyleBar displays when you open any painter that can 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 can display a brief description of the button (a PowerTip). The ability to display PowerTips is toggled on and off by selecting the Show PowerTips menu item in any toolbar pop-up menu.



You can also include brief descriptive texts on all toolbar buttons by selecting ShowText from any toolbar pop-up menu.

Customizing the environment

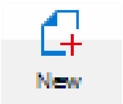
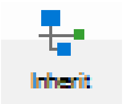
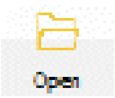
In addition to displaying text in toolbar buttons, you can move the toolbars around, add new toolbars, and customize existing ones. You can add buttons for opening painters and performing other activities.

You can also rearrange the System Tree, Clip, and Output views, set up custom layouts for each painter, choose whether PowerBuilder opens your last workspace at start-up with or without painters and editors open, customize shortcut keys, and change the colors and fonts used in scripts.

PowerBar buttons

The buttons in the PowerBar give you quick access to the most common PowerBuilder tasks:

Table 1.1: Buttons in the PowerBar

Button	Use to
 New	Create new workspace, target, component, or other object, or open a tool.
 Inherit	Inherit from menu, user object, or window.
 Open	Open an existing application, DataWindow, function, menu, pipeline, project, query, structure, user object, window, HTML page, HTML frame, style sheet, or script file.

Button	Use to
Preview	Preview a window or DataWindow object.
SysTree	Show or hide the System Tree window.
Output	Show or hide the Output window.
Next	Move to the next line in the Output window.
Previous	Move to the previous line in the Output window.
To-Do List	Display a list of development tasks you need to do. These can be self entered or entered automatically by PowerBuilder wizards.
Browser	View object information (such as object properties or global variables) and copy, export, or print it.
Clip	Show or hide the Clip window.
Library	Create and maintain libraries of PowerBuilder objects.
DB Prof	Specify how to connect to a database.
Database	Maintain databases, control user access to databases, and manipulate data in databases.
Edit	Edit a file.
I. Build	Start an incremental build of the workspace.
F. Build	Start a full build of the workspace.

Button	Use to
Deploy	Deploy the workspace.
Skip	When a series of operations is in progress, such as a full deploy of the workspace, skip to the next operation.
Stop	Stop a build or deploy operation or series of operations.
Debug	Debug the current target.
Sel Dbg	Select a target and debug it.
Run	Run the current target.
Sel Run	Select a target and run it.
Exit	Exit from PowerBuilder.

1.1.3 PowerBuilder objects

The basic building blocks of a PowerScript target are objects:

Table 1.2: Basic building blocks of a PowerScript target

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

Object	Use
User object	Reusable processing module or set of controls, either visual or nonvisual
Pipeline	Reproduces data within a database or across databases
Project	Packages application for distribution to users

These objects are described in more detail in the sections that follow.

Application object

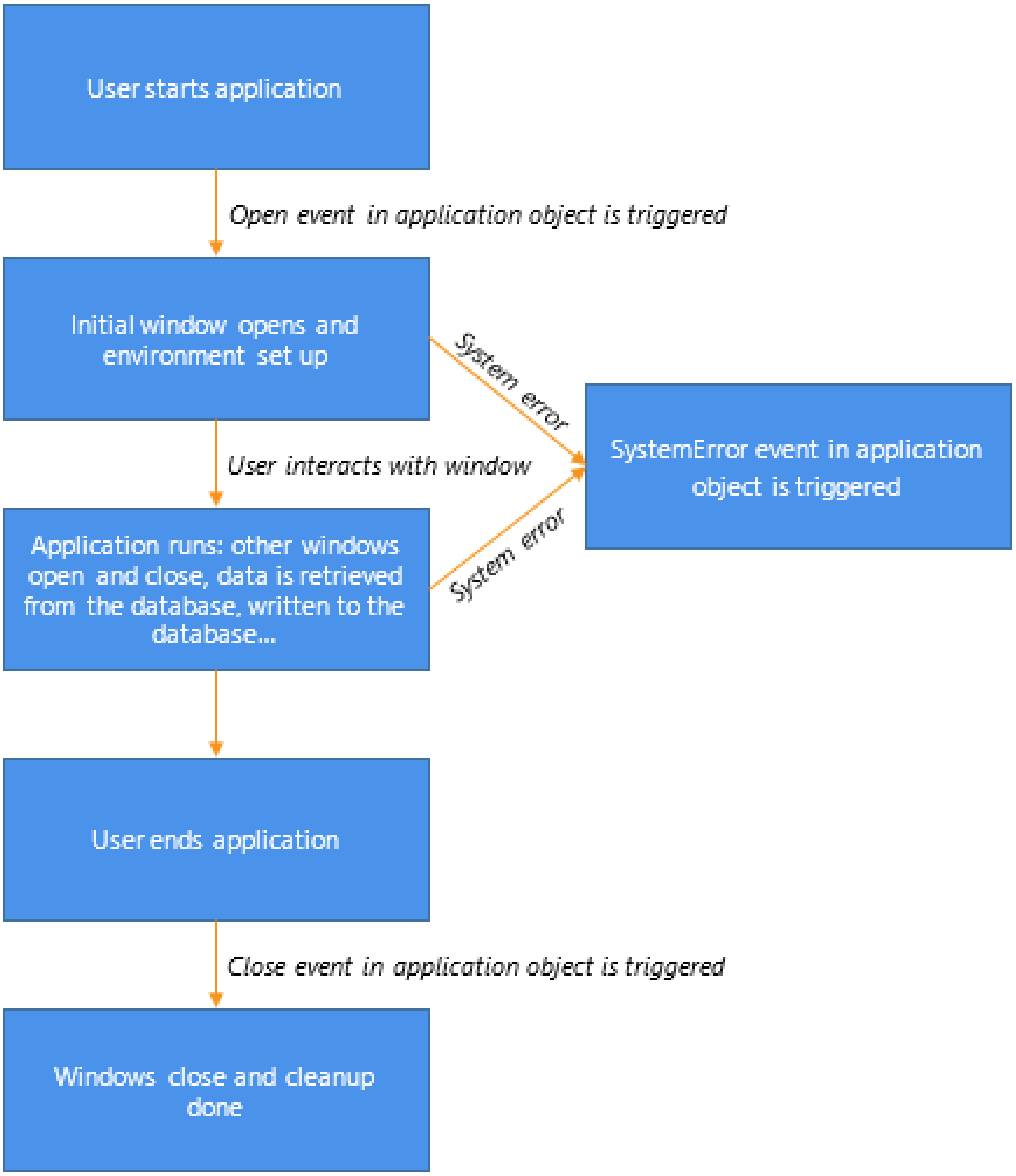
The Application object is the entry point into an application. It is a discrete object that is saved in a PowerBuilder library (PBL file), just like a window, menu, function, or DataWindow object.

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

When a user runs the application, an Open event is triggered in the Application object. The script you write for the Open event initiates the activity in the application. When the user ends the application, the Close event in the Application object is triggered.

The script you write for the Close event typically does all the cleanup required, such as closing a database or writing to a preferences file. If there are serious errors during execution that are not caught using PowerBuilder's exception handling mechanism, the Application object's SystemError event is triggered.

Figure: Application life cycle


User starts application
Open event in application object is triggered
Initial window opens and environment set up
System error
SystemError event in application object is triggered
User interacts with window
Application runs: other windows open and close, data is retrieved from the database, written to the database...
System error
User ends application
Close event in application object is triggered
Windows close and cleanup done

Windows

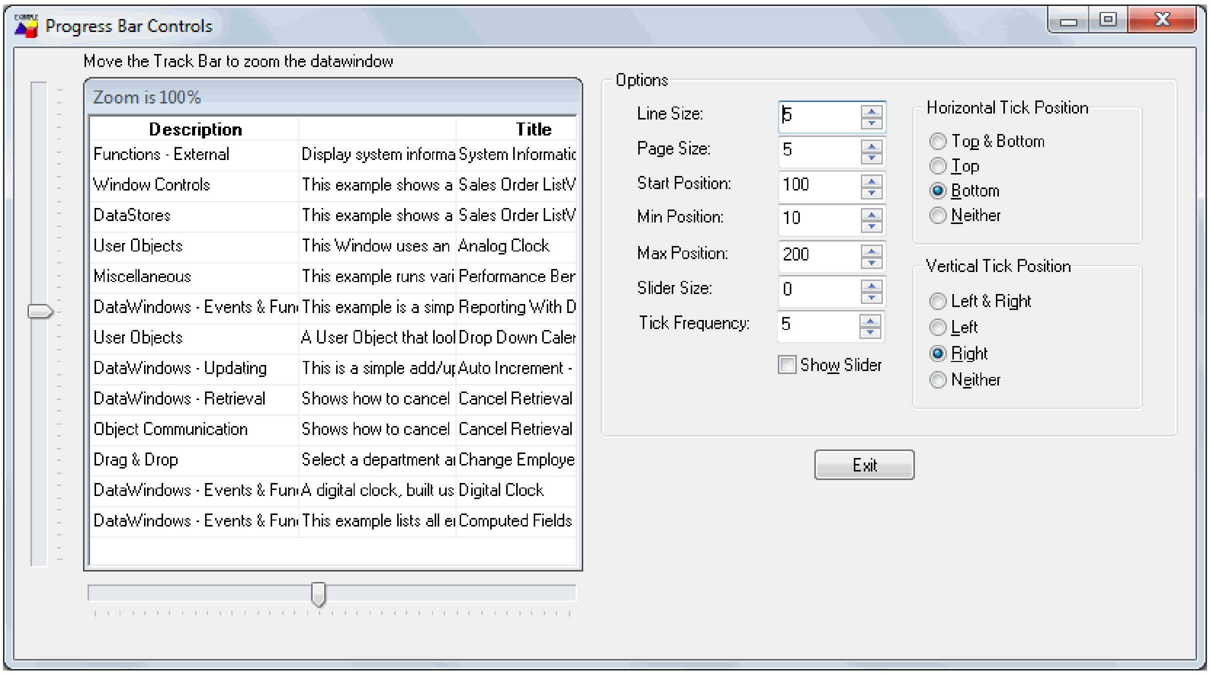
Windows are the primary interface between the user and a PowerBuilder application. Windows can display information, request information from a user, and respond to the user's mouse or keyboard actions.

A window consists of:

- Properties that define the window's appearance and behavior (for example, a window might have a title bar and a Minimize box)
- Events triggered by user actions

- Controls placed in the window

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



Progress Bar Controls

Move the Track Bar to zoom the datawindow

Zoom is 100%

Description		Title
Functions - External	Display system informa	System Informatic
Window Controls	This example shows a	Sales Order ListV
DataStores	This example shows a	Sales Order ListV
User Objects	This Window uses an	Analog Clock
Miscellaneous	This example runs vari	Performance Ber
DataWindows - Events & Fun	This example is a simp	Reporting With D
User Objects	A User Object that lool	Drop Down Caler
DataWindows - Updating	This is a simple add/u	Auto Increment -
DataWindows - Retrieval	Shows how to cancel	Cancel Retrieval
Object Communication	Shows how to cancel	Cancel Retrieval
Drag & Drop	Select a department a	Change Employe
DataWindows - Events & Fun	A digital clock, built us	Digital Clock
DataWindows - Events & Fun	This example lists all e	Computed Fields

Options

Line Size: 5
Page Size: 5
Start Position: 100
Min Position: 10
Max Position: 200
Slider Size: 0
Tick Frequency: 5

□ Show Slider

Horizontal Tick Position
○ Top & Bottom
○ Top
● Bottom
○ Neither

Vertical Tick Position
○ Left & Right
○ Left
● Right
○ Neither

On the left of the window is a DataWindow control with horizontal and vertical trackbars. On the right is a group box that contains static text controls (containing descriptive labels), edit mask controls (as they appear when the SpinControl property is on), a check box, and two smaller group boxes with radio buttons. Under the main group box is a command button.

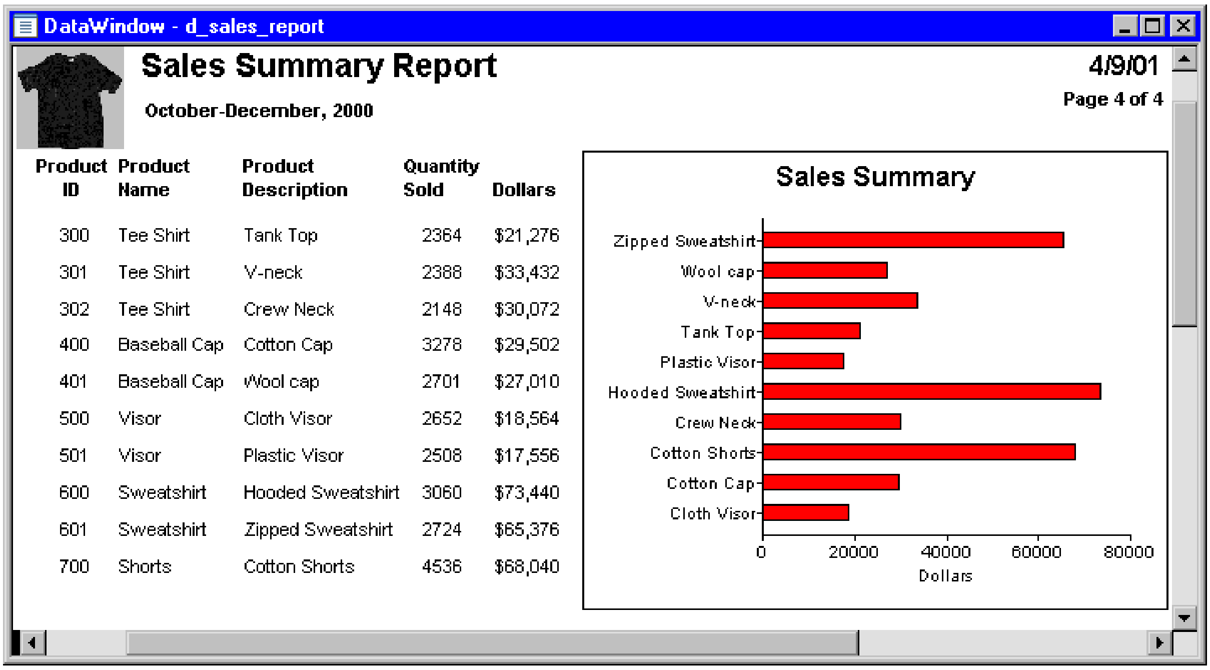
DataWindow objects

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 Tabular or 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.



Sales Summary Report

October-December, 2000

4/9/01

Page 4 of 4

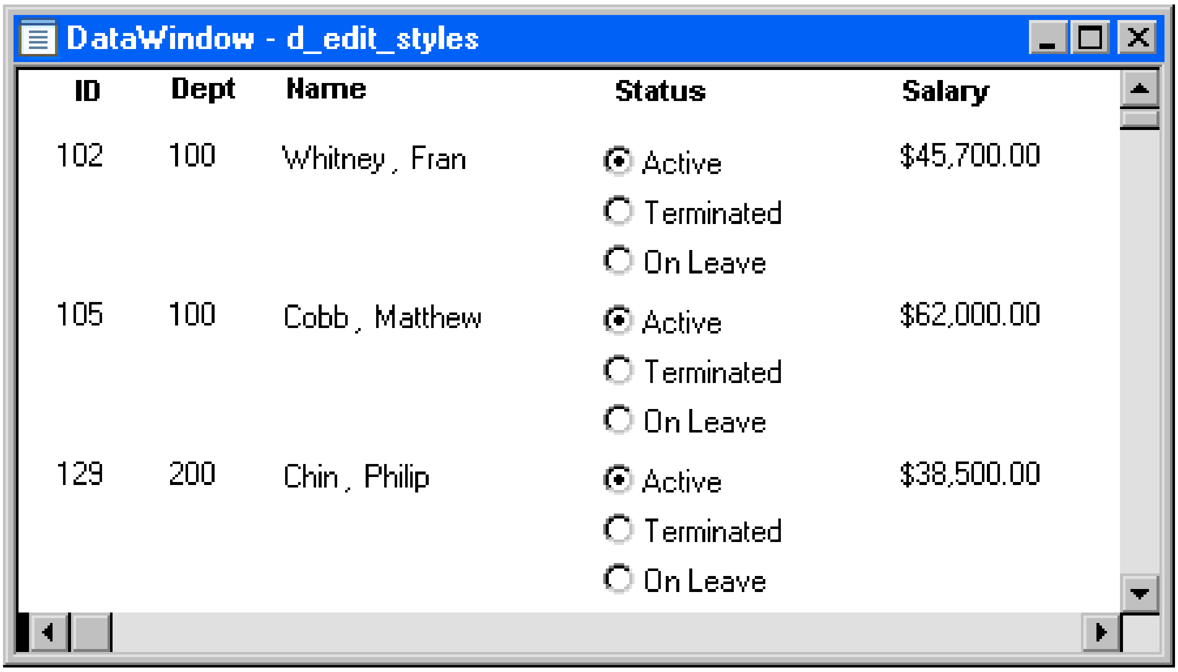
Product ID	Product Name	Product Description	Quantity Sold	Dollars
300	Tee Shirt	Tank Top	2364	$21,276
301	Tee Shirt	V-neck	2388	$33,432
302	Tee Shirt	Crew Neck	2148	$30,072
400	Baseball Cap	Cotton Cap	3278	$29,502
401	Baseball Cap	Wool cap	2701	$27,010
500	Visor	Cloth Visor	2652	$18,564
501	Visor	Plastic Visor	2508	$17,556
600	Sweatshirt	Hooded Sweatshirt	3060	$73,440
601	Sweatshirt	Zipped Sweatshirt	2724	$65,376
700	Shorts	Cotton Shorts	4536	$68,040

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.

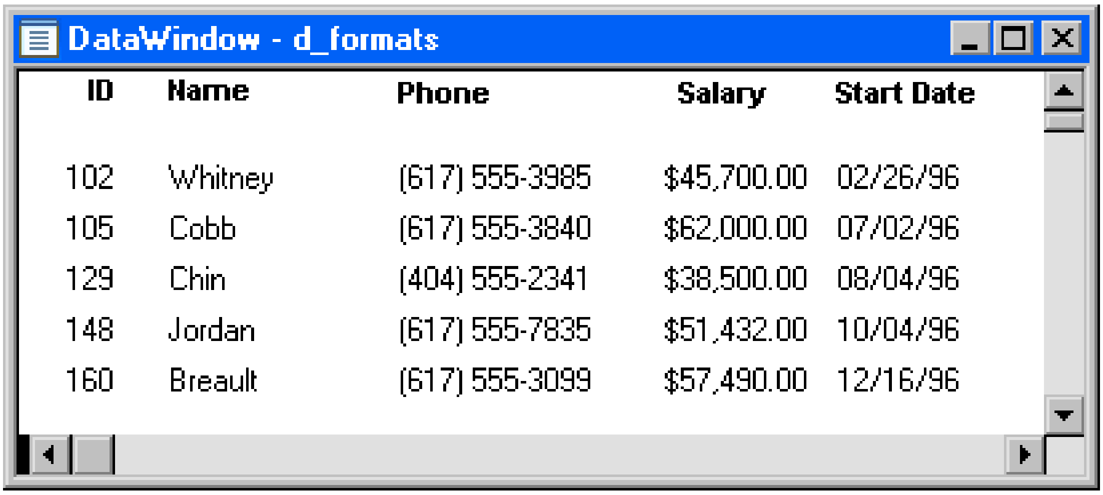
For example:

- If a column can take only a small number of mutually exclusive values, you can have the data appear as radio buttons in a DataWindow so users know what their choices are.



ID	Dept	Name	Status	Salary
102	100	Whitney , Fran	● Active ○ Terminated ○ On Leave	$45,700.00
105	100	Cobb , Matthew	● Active ○ Terminated ○ On Leave	$62,000.00
129	200	Chin , Philip	● Active ○ Terminated ○ On Leave	$38,500.00

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



ID	Name	Phone	Salary	Start Date
102	Whitney	(617) 555-3985	$45,700.00	02/26/96
105	Cobb	(617) 555-3840	$62,000.00	07/02/96
129	Chin	(404) 555-2341	$38,500.00	08/04/96
148	Jordan	(617) 555-7835	$51,432.00	10/04/96
160	Breault	(617) 555-3099	$57,490.00	12/16/96

- If a column can take numbers only in a specific range, you can specify a simple validation rule for the data. This can spare you from writing code to make sure users enter valid data.

Web DataWindow

This technique is not recommended and is considered to be obsolete. An obsolete feature is no longer eligible for technical support and will no longer be enhanced, although it is still available.

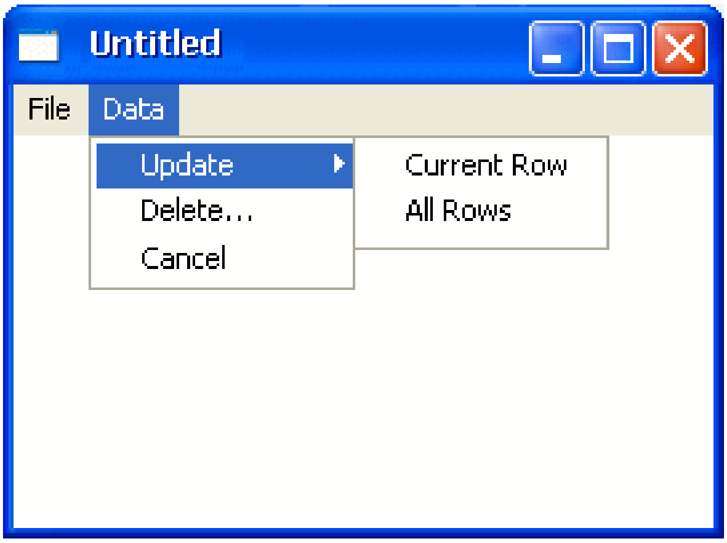
The ability to use this technique has been retained for backward compatibility.

Menus

Menus are lists of items that a user can select from a menu bar for the active window. The items on a menu are usually related. They provide the user with commands (such as Open and Save As on the PowerBuilder File menu) or alternate ways of performing a task (for example, the items on the Edit menu in the Window painter correspond to buttons in the PainterBar).

You can select menu items with the mouse or with the keyboard, or use accelerator (mnemonic access) keys defined for the items. You can define your own keyboard shortcuts for any PowerBuilder menu item from a dialog box that you open with the Tools>Keyboard Shortcuts menu item.

A drop-down 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 drop-down 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 drop-down Data menu (Update, Delete, and Cancel), and two Menu objects on the cascading menu beside Update (Current Row and All Rows).

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. These are further divided into system functions (functions that are always available for objects of a certain object class) and user-defined functions.

- 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.

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.

There are two kinds of structures:

- Object-level structures 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 are not associated with any object or type of 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. Several applications might all require a standard file viewer.

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.

User objects can be visual or nonvisual. They can be further divided into standard or custom user objects. Standard user objects, whether visual or nonvisual, are system objects that are always available with PowerBuilder. You can also use controls for external visual objects that were created outside PowerBuilder. The main types of user objects are:

- Visual user objects

  These 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 is defined, you can use it as often as you need.

- Nonvisual user objects

  These are reusable processing modules that have no visual component. Standard class user objects inherit events and properties from built-in system objects. You typically use nonvisual 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 custom class user object. To use a custom 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 multitier applications. This is because you typically use nonvisual components for applications that are run on a server.

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

You can create Project objects that build executable applications.

1.2 About the PowerBuilder Tutorial

About this chapter

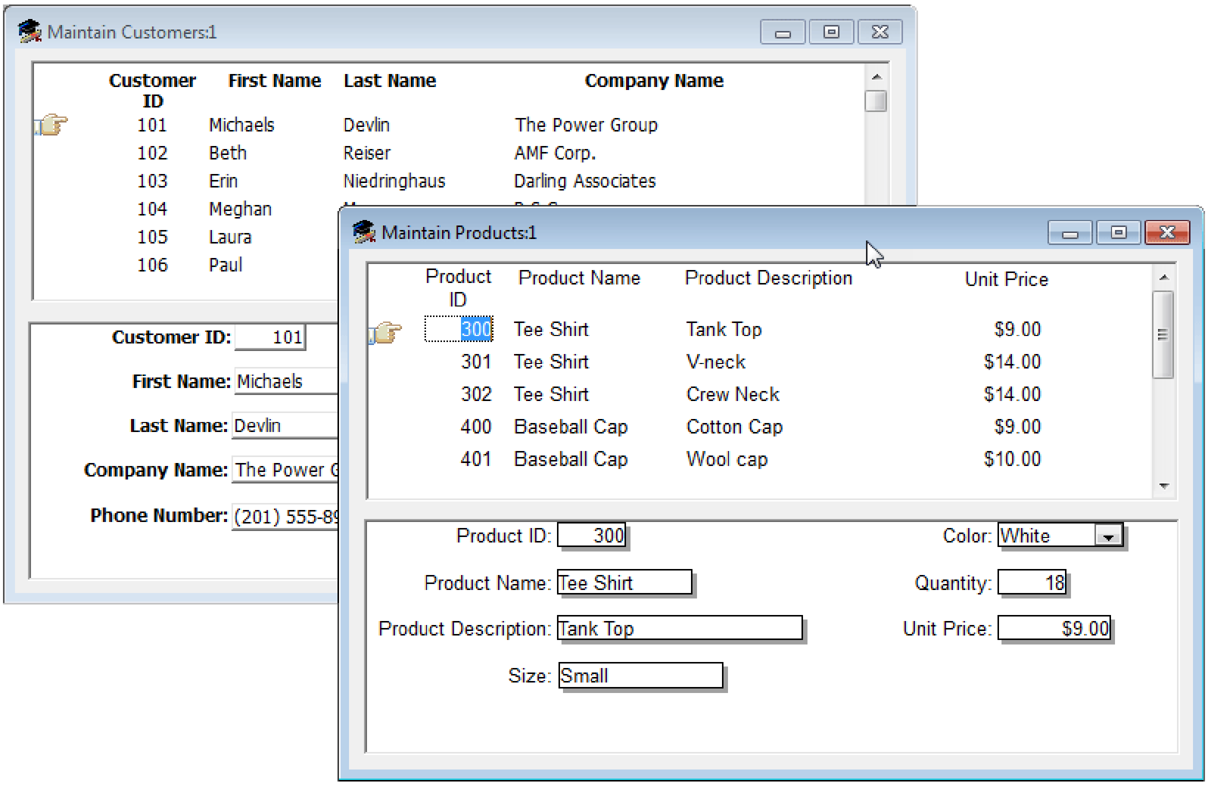
This chapter describes what you will do in the PowerBuilder tutorial and how to get set up for it.

1.2.1 Learning to build a client/server application

The PowerBuilder tutorial is divided into parts. The first part of the tutorial is a set of eleven 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 MDI application includes two windows that provide access to the Customer and Product tables in the Demo database.



Maintain Customers:1

Customer ID	First Name	Last Name	Company Name
101	Michaels	Devlin	The Power Group
102	Beth	Reiser	AMF Corp.
103	Erin	Niedringhaus	Darling Associates
104	Meghan		
105	Laura		
106	Paul		

Customer ID: 101

First Name: Michaels

Last Name: Devlin

Company Name: The Power G

Phone Number: (201) 555-8

Maintain Products:1

Product ID	Product Name	Product Description	Unit Price
300	Tee Shirt	Tank Top	$9.00
301	Tee Shirt	V-neck	$14.00
302	Tee Shirt	Crew Neck	$14.00
400	Baseball Cap	Cotton Cap	$9.00
401	Baseball Cap	Wool cap	$10.00

Product ID: 300

Product Name: Tee Shirt

Product Description: Tank Top

Size: Small

Color: White

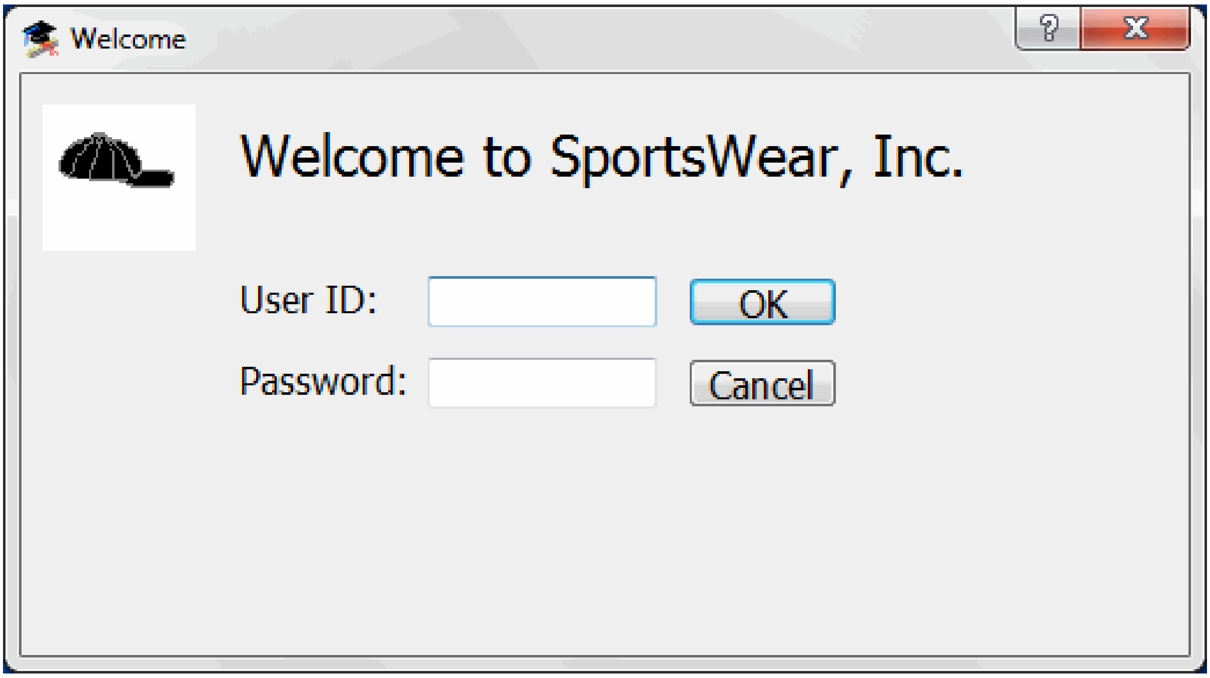
Quantity: 18

Unit Price: $9.00

These 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.

Login window

The MDI application also includes a login window that allows you to connect to the database at start-up time.



1.2.2 How you will proceed

The following table describes what you will do in each of the tutorial lessons.

Table 1.3: Tutorial lessons and what you will accomplish

Lesson	What you will do
1	Start PowerBuilder; begin familiarizing yourself with the development environment; use the Workspace wizard and the Template Application wizard to create an Application object, windows, and menus in a PowerBuilder workspace and target.
2	Explore the PowerBuilder environment and customize the workspace.
3	Create a login window to allow the user to enter database connection parameters (user ID and password).
4	Connect to the database using the Transaction object and user-entry parameters; see how database profiles are defined in the PowerBuilder environment.
5	Change the base sheet window by adding master and detail DataWindow controls; add scripts to allow users to retrieve data and perform insert, update, and delete operations against the database.
6	Modify the frame menu and create a new sheet menu for the application.
7-8	Build the DataWindow objects that retrieve customer and product information, then add them to the Customer and Product windows.
9	Run the tutorial application in debug mode; see how to set breakpoints in scripts, step through the code, and display the contents of variables.
10	Create a new window to test exception handling in PowerBuilder.
11	Create an executable file that you can use to run the application outside the PowerBuilder development environment.

1.2.2.1 How long it will take

You can do all the tutorials in about six hours, or you can stop after any lesson and continue at another time.

If you are interrupted

You can save your work and exit PowerBuilder at any time. When you are ready to continue, you can open the tutorial workspace and continue where you left off.

1.2.2.2 What you will learn

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

Client/server applications

You will learn basic PowerBuilder techniques and concepts, including those listed in the following table:

Table 1.4: Features demonstrated in the PowerScript tutorial

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
Menu painter	Define menus, menu items, accelerators, and shortcut keys
Layout view	Design how the windows, menus, and DataWindows will look when you run the application
Script view	Define scripts for applications, windows, window controls, and menus
Debugger	Identify logic errors that may cause problems when you run the application
Project painter	Create an executable version of an application

1.2.3 Setting up for the tutorial

Before you start the tutorial, you need to make sure that you can connect to a database and that you have the tutorial files.

Connecting to a database

The tutorial uses the PB Demo DB V2019R3 database that installs with PowerBuilder. This is an SQL Anywhere database and requires the SAP SQL Anywhere engine.

If you do not already have SQL Anywhere on your local machine or server, you must install it now. You can download and install it from the SAP website. If you installed PowerBuilder in a nondefault location, you must make sure that the odbc.ini registry entry defining the Demo Database as a data source points to the correct location of the SQL Anywhere engine.

The Tutorial directory

The PowerBuilder installation directory includes a Tutorial folder that contains all the files you need to work with the tutorial. The installation also creates a copy of that folder under your user profile directory. You will work with the copy of the Tutorial folder, not the original source. The Tutorial folder locates in %SystemDrive%\Users\[username]\Documents \Appeon\PowerBuilder [version]\Tutorial on Windows:

The Tutorial folder includes the following files:

Table 1.5: Files required by the PowerScript tutorial

File	Contents
tutor_pb.pbl	PowerBuilder library that contains several objects that you use in the tutorial
tutsport.bmp	A bitmap
tshirtw.jpg	A graphic
tutorial.ico	An icon

When you have finished the tutorial, you can delete the files.

The Tutorial\Solutions directory

The Tutorial\Solutions directory contains a PowerBuilder library called pbtutor.pbl that contains all the objects and scripts that you create in the first part of the tutorial, as well as workspace and target files. You can use this solutions library as a reference while you complete the first part of the tutorial.

2 Building a Client/Server Application

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.

2.1 LESSON 1 Starting PowerBuilder

This lesson provides the information you need to start PowerBuilder and create an application.

In this lesson you:

- Create a new workspace
- Create a target
- Specify an icon for the application
- Change the size of the main window
- Run the application

How long does it take?

About 20 minutes.

2.1.1 Create a new workspace

Where you are

> Create a new workspace

Create a target

Specify an icon for the application

Change the size of the main window

Run the application

The workspace is where you build, edit, debug, and run PowerBuilder targets. You can build several targets within a single workspace.

Now you start PowerBuilder and create a new workspace.

First read the Release Bulletin for this release

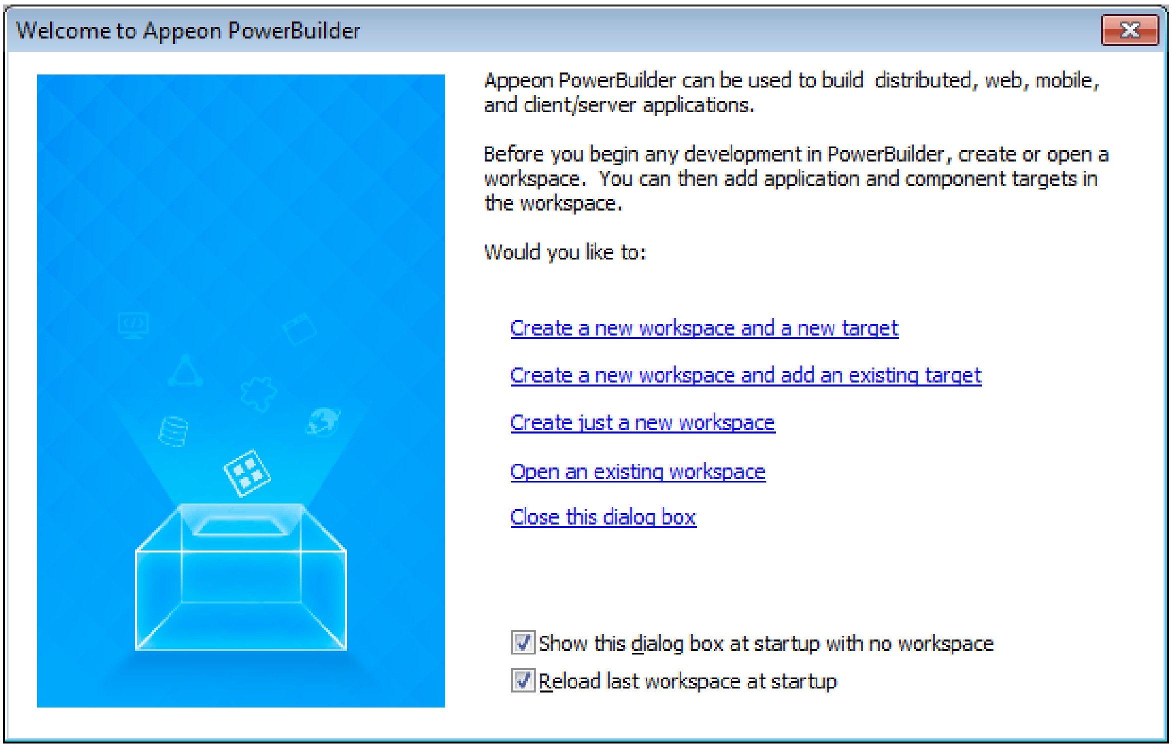
Any last-minute items are documented in the Release Bulletin. To make sure you have all the files necessary to complete the tutorial, see Setting up for the tutorial.

1. Double-click the PowerBuilder icon (representing PB190.EXE) in %AppeonInstallPath%\PowerBuilder [version]\

   or

   Select All Programs>Appeon>PowerBuilder 2019 R3>PowerBuilder 2019 R3 from the Windows Start menu.

The Welcome to PowerBuilder dialog box displays.



Welcome to Appeon PowerBuilder

Appeon PowerBuilder can be used to build distributed, web, mobile, and client/server applications.

Before you begin any development in PowerBuilder, create or open a workspace. You can then add application and component targets in the workspace.

Would you like to:

Create a new workspace and a new target

Create a new workspace and add an existing target

Create just a new workspace

Open an existing workspace

Close this dialog box

☑ Show this dialog box at startup with no workspace

☑ Reload last workspace at startup

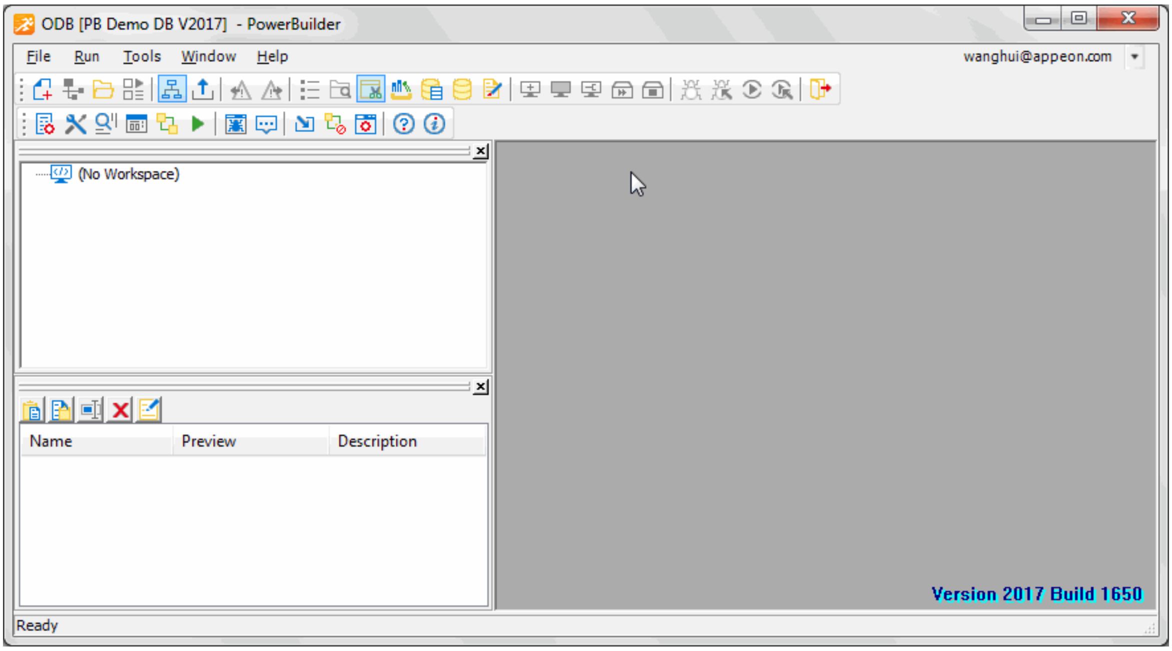
The Welcome to PowerBuilder dialog box allows you create a new workspace and add a new target or an existing target to the workspace.

If you do not want PowerBuilder to display the dialog box again

You can leave the "Show this dialog box at startup with no workspace" check box unchecked to keep PowerBuilder from displaying the welcome dialog box every time you start PowerBuilder. Select the "Reload last workspace at startup" check box to load the most recently used workspace each time you start a PowerBuilder session. Then click "Close this dialog box".

The PowerBuilder development environment displays.

If this is the first time you are opening PowerBuilder on your machine, you see only a top-level entry in the System Tree to indicate that no workspace is currently open. Otherwise, the System Tree might show a workspace with targets and objects in it.



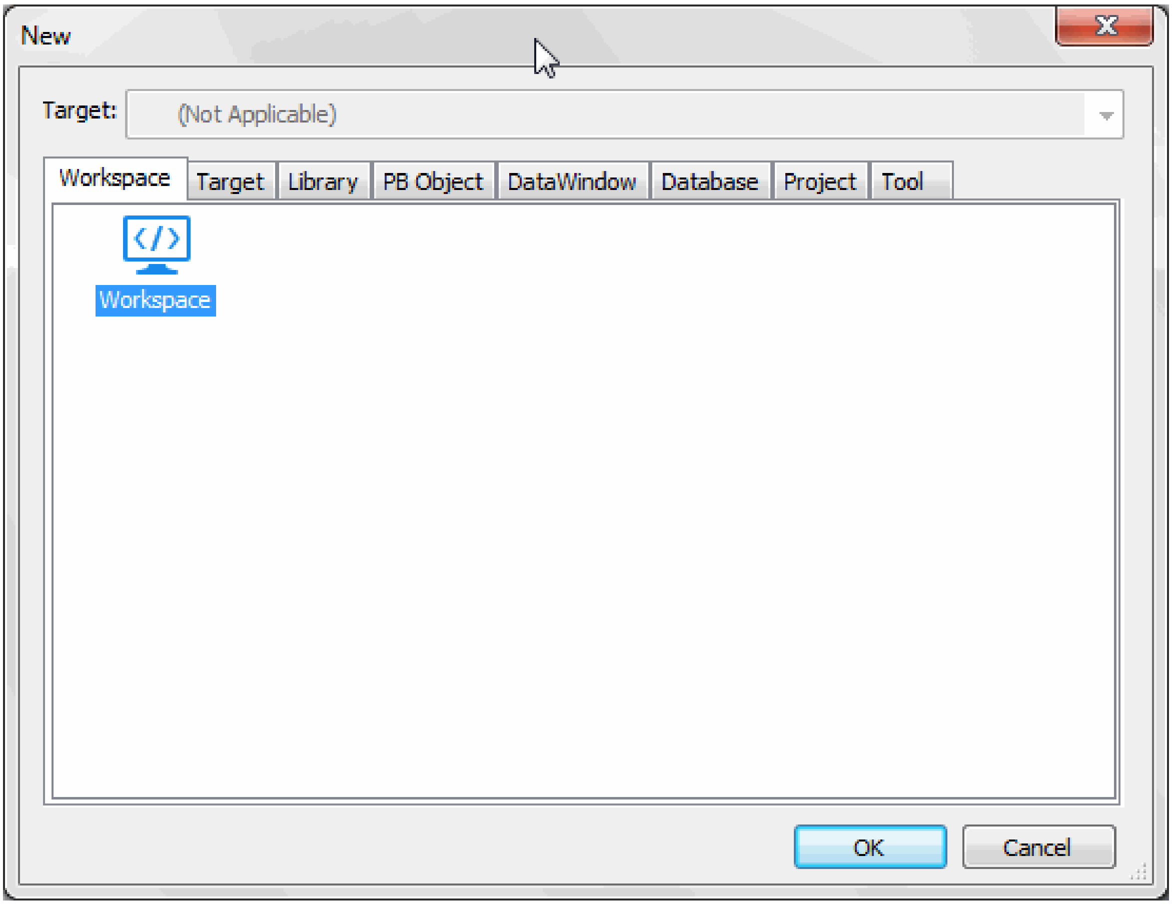
2. Select New from the File menu

   or

   Click the New button in the PowerBar.

   The Workspace page of the New dialog box displays.

   PowerBuilder displays the page of the New dialog box that was used before the dialog box was last closed. In this exercise, make sure that the Workspace page of the New dialog box displays.



3. Select Workspace from the Workspace page of the New dialog box.

   Click OK.

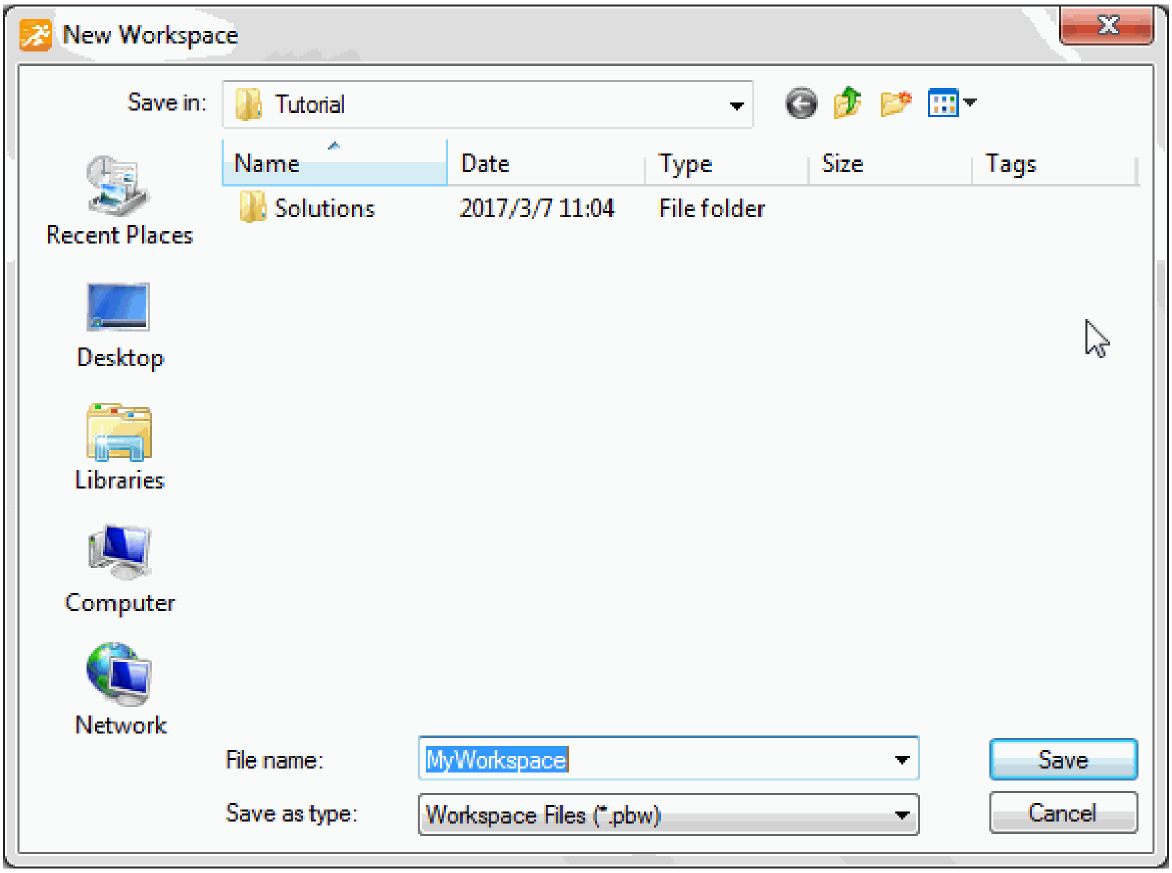
   The New Workspace dialog box displays.

4. Choose the tutorial folder.

   If you have created a workspace before, the dialog opens to the location of the most recently-used workspace. For this new workspace, change the location to the path described in the next paragraph.

   If this is your first workspace, the New Workspace dialog box opens to %SystemDrive %\Users\[username]\Documents. Navigate from this point to Appeon\PowerBuilder [version]\Tutorial. The solutions for the tutorial are in the Solutions subfolder, but you will create your own solutions as you work your way through the tutorial.

5. Type MyWorkspace in the File name text box.



Save in: Tutorial

Name	Date	Type	Size	Tags
Solutions	2017/3/7 11:04	File folder		

File name: MyWorkspace

Save as type: Workspace Files (*.pbw)

6. Click Save.

   The New Workspace dialog box closes and the workspace you created appears as the first item in the System Tree.

2.1.2 Create a target

Where you are

Create a new workspace

> Create a target

Specify an icon for the application

Change the size of the main window

Run the application

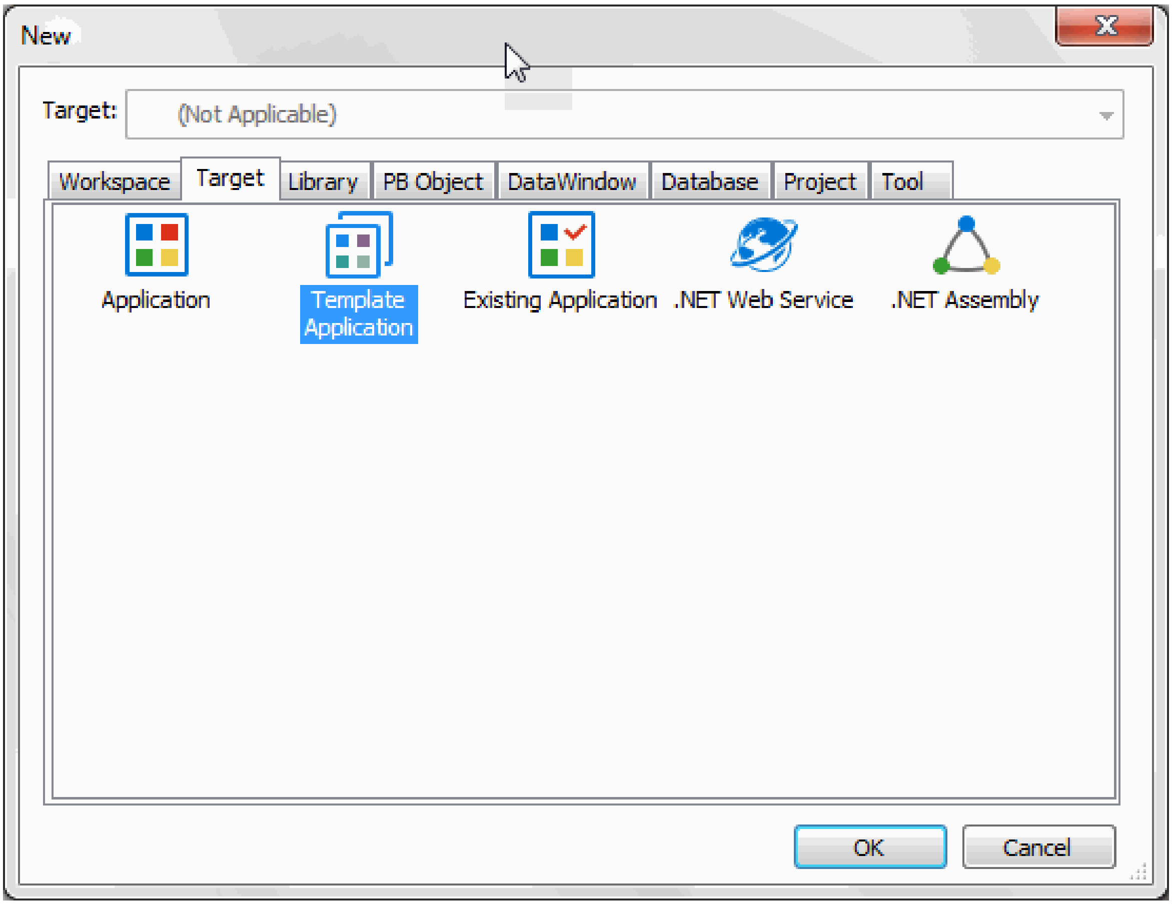
Now you create a new target using the Template Application wizard. Based on the choices you make, the Template Application wizard creates precoded events, menus, windows, and user objects in addition to the application object.

1. Select New from the File menu and click the Target tab

   or

   Right-click MyWorkspace in the System Tree, select New from the pop-up menu, and click the Target tab.

   The Target page of the New dialog box displays.



Target: (Not Applicable)

2. Select the Template Application icon and click OK.

   The Template Application wizard displays. The first page of most wizards explains what the wizard is used for. As you step through the wizard, you can press F1 to get Help on most fields.

3. Click Next until the Specify New Application and Library page displays.

4. Type pbtutor in the Application Name text box.

   The wizard automatically assigns file names to a library and target that use this application name. It assigns the library a PBL extension and the target a PBT extension.

5. Click Next.

   The Specify Template Type page displays. The MDI Application with Microhelp radio button is selected. You will create an MDI template application, so you do not need to change this selection.

About MDI

MDI stands for multiple document interface. In an MDI application, the main window for the application is called the MDI frame. Using the MDI frame menu bar, you can open additional windows known as sheet windows that display inside the frame window.

6. Click Next 4 times until the Name Individual Sheets page displays.

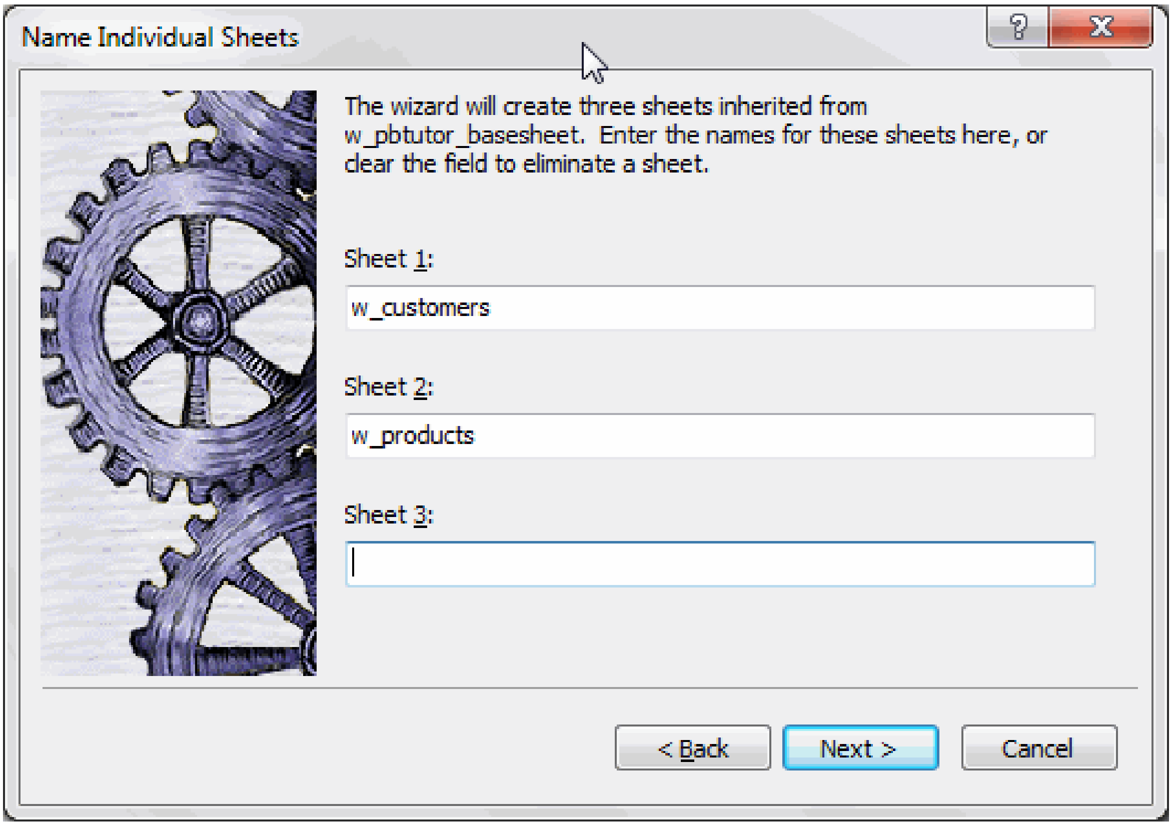
   In this tutorial you accept the default application type, library search path, frame and frame menu names, sheet menu and manager service, and MDI base sheet.

If you have clicked Next too many times

You can use the wizard's Back button to navigate back to the correct wizard page.

7. On the Name Individual Sheets page, type w_customers for Sheet 1, w_products for Sheet 2, and clear the Sheet 3 text box.

   PowerBuilder will generate two windows based on the default basesheet (w_pbtutor_basesheet), one for customers and one for products. You will add a third sheet window later in the lesson on exception handling.


Name Individual Sheets

The wizard will create three sheets inherited from w_pbtutor_basesheet. Enter the names for these sheets here, or clear the field to eliminate a sheet.

Sheet 1: w_customers

Sheet 2: w_products

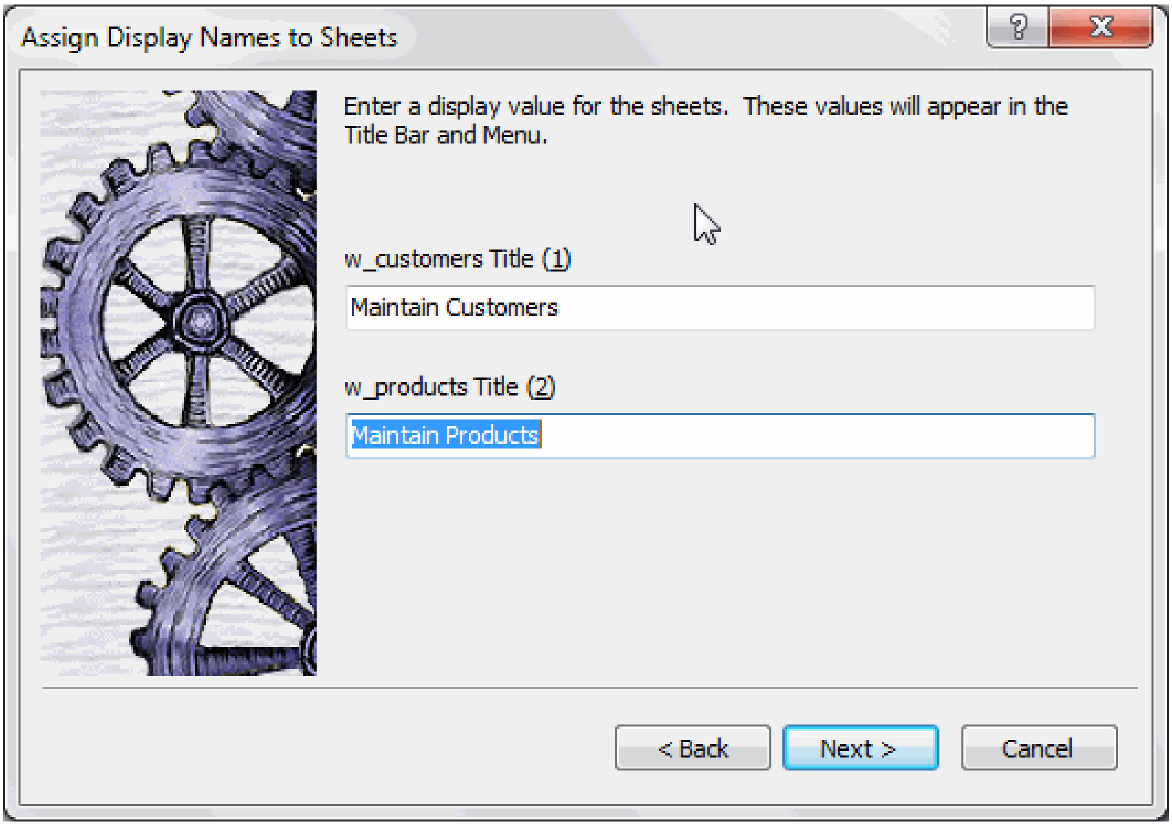
Sheet 3:

8. Click Next.

   Type Maintain Customers as the display name for Sheet 1.

   Type Maintain Products as the display name for Sheet 2.

   The names you type will display in the title bars of these sheet windows.


Assign Display Names to Sheets

Enter a display value for the sheets. These values will appear in the Title Bar and Menu.

w_customers Title (1)

Maintain Customers

w_products Title (2)

Maintain Products

9. Click Next twice.

   You do not need to change the names of the About and Toolbar windows.

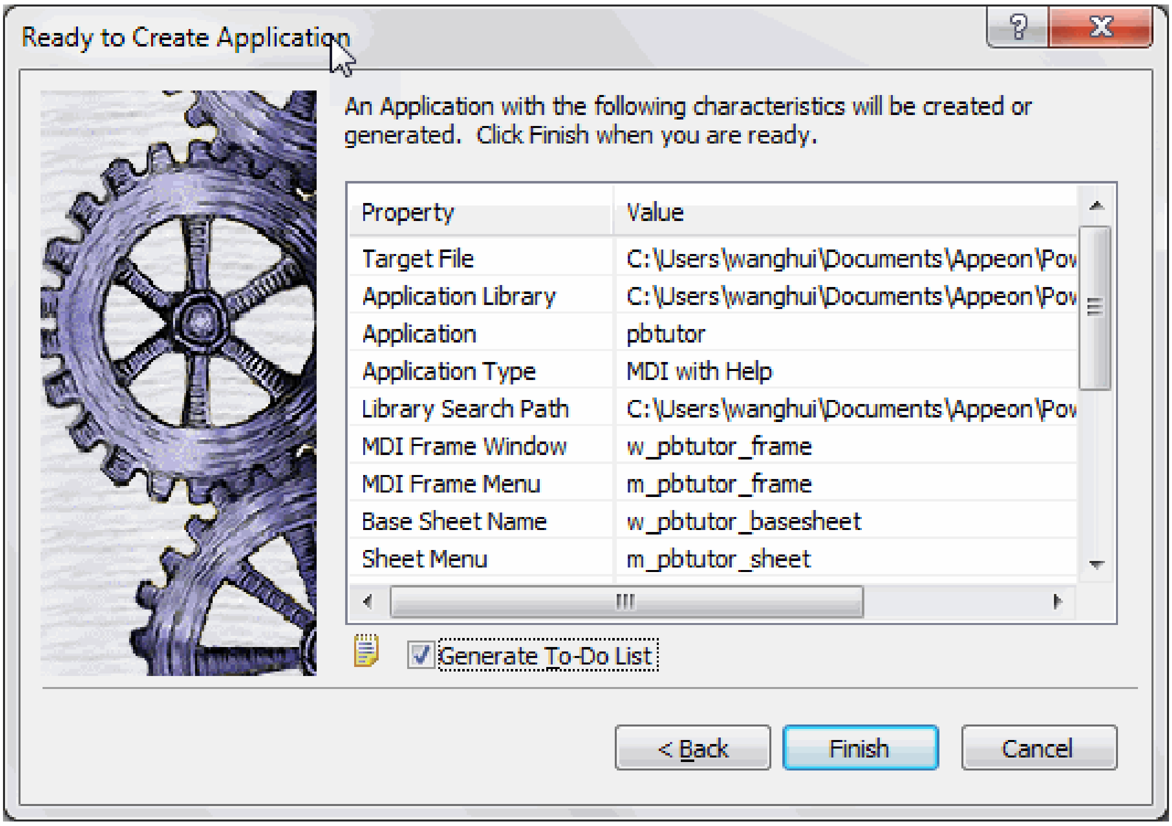
10. On the Specify Connectivity page, select None.

    You will add a Connection object later.

11. Click Next twice to display the Ready To Create Application page.

    You will create a project later.

    This is the last wizard page. It lists your current selections so that you can review them and use the Back button to go back and change them if necessary.



Ready to Create Application

An Application with the following characteristics will be created or generated. Click Finish when you are ready.

Property	Value
Target File	C:\Users\wanghui\Documents\Appeon\Pow
Application Library	C:\Users\wanghui\Documents\Appeon\Pow
Application	pbtutor
Application Type	MDI with Help
Library Search Path	C:\Users\wanghui\Documents\Appeon\Pow
MDI Frame Window	w_pbtutor_frame
MDI Frame Menu	m_pbtutor_frame
Base Sheet Name	w_pbtutor_basesheet
Sheet Menu	m_pbtutor_sheet

☑ Generate To-Do List

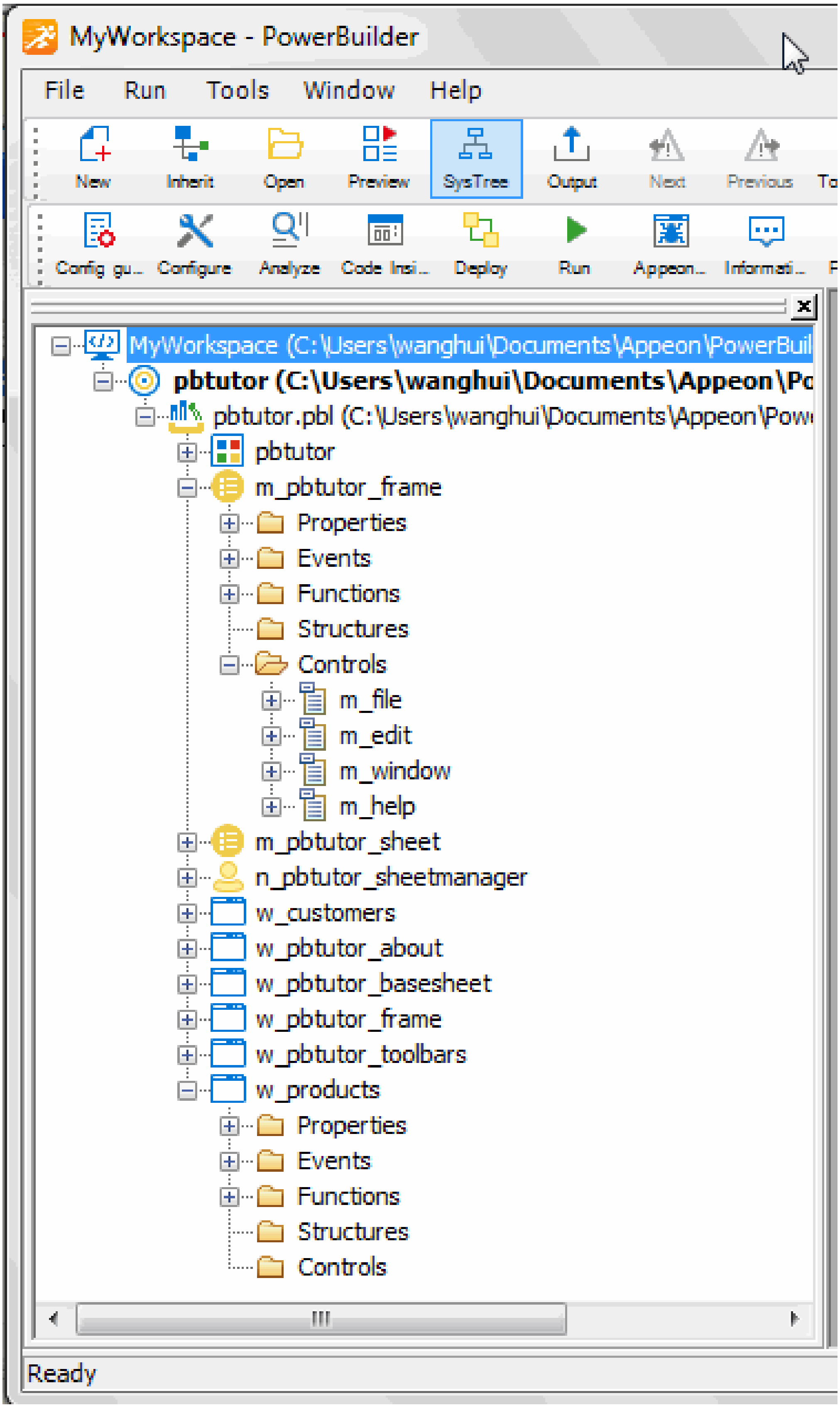
12.Make sure the Generate To-Do List check box is selected.

13.Click Finish.

The Template Application wizard creates the pbtutor.pbt target and the pbtutor.pbl library, and sets the new pbtutor application as the default application.

You can expand the System Tree to view all the objects that have been created by the Template Application wizard. The System Tree does not display the file extension of the pbtutor target, but it does display the directory where the target file is saved.

The pbtutor.pbl library displays under the pbtutor target in the System Tree. It contains the target Application object, which has the same name as the target object but displays under the library file. Other objects generated by the wizard also display under the library file.



2.1.3 Specify an icon for the application

Where you are

Create a new workspace

Create a target

> Specify an icon for the application

Change the size of the main window

Run the application

Now you specify an icon for the application. The icon appears in the workspace when you minimize the application during execution. PowerBuilder includes the icon automatically when you create an executable file. You specify an icon from the Properties view in the Application painter.

1. Double-click the pbtutor Application object in the System Tree

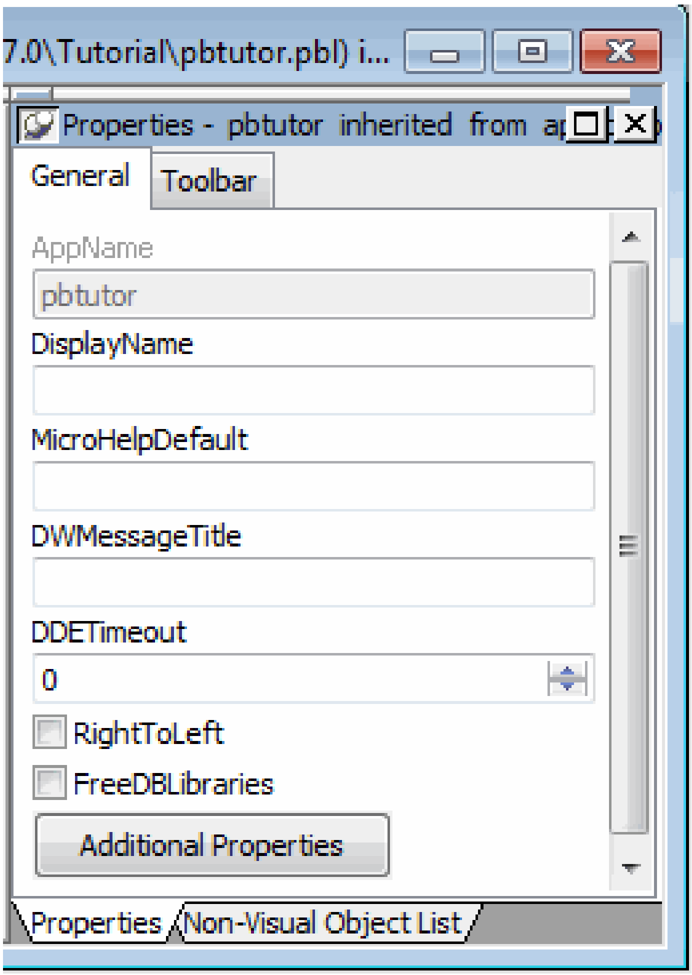
   or

   Right-click the pbtutor Application object in the System Tree and select Edit from the pop-up menu.

   The pbtutor Application object is located under the pbtutor library, which is under the pbtutor target object that you created with the Template Application wizard. Different views of the Application object display in the Application painter.

2. Make sure the Properties view displays in the Application painter.

   If the Properties view is not open, you can open it by selecting View>Properties from the menu bar. The menu item is grayed out if the Properties view is already open.



7.0\Tutorial\pbtutor.pbl) i...

Properties - pbtutor inherited from ap

AppName

pbtutor

DisplayName

MicroHelpDefault

DWMessageTitle

DDETimeout

0

□ RightToLeft

□ FreeDBLibraries

3. Click the Additional Properties button in the Properties view.

   A tabbed Application property sheet displays.

4. Select the Icon tab.

5. Click Browse.

   Navigate to the Tutorial directory.

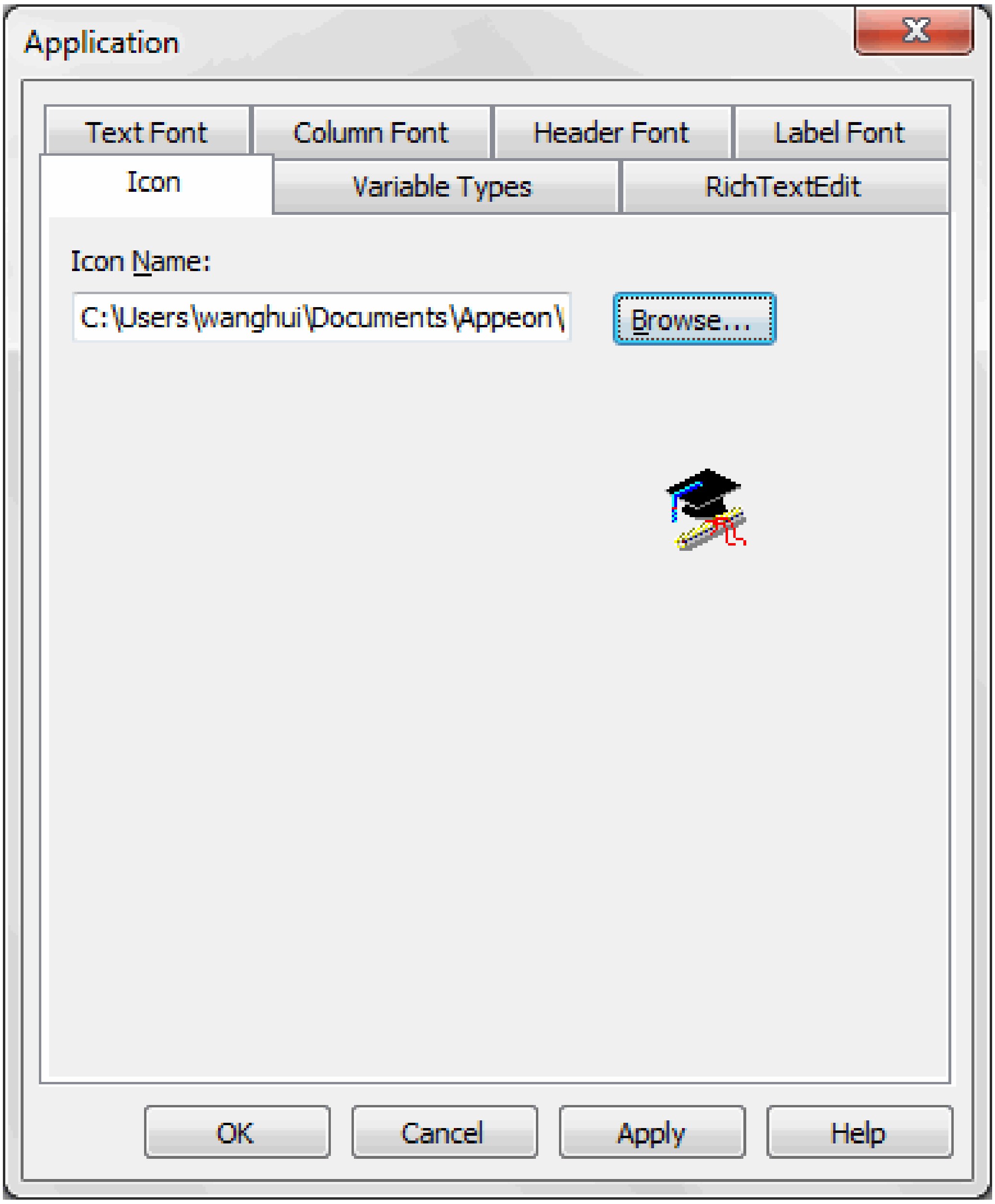
6. Select the tutorial.ico file.

   Click Open.

If you do not see the ICO file extension

You do not see ICO file extensions if the Hide File Extensions for Known File Types check box is selected in the Options dialog box of your Windows Explorer.

The tutorial icon displays on the Icon page of the Application property sheet.



Icon Name: C:\Users\wanghui\Documents\Appeon\

7. Click OK.

   Click the Save button in PainterBar1 or select File>Save.

   Click the Close button in PainterBar1 or select File>Close.

2.1.4 Change the size of the main window

Where you are

Create a new workspace

Create a target

Specify an icon for the application

> Change the size of the main window

Run the application

Now you change the size of the application's main window. When you run the application, the main window displays in the position and size that you specify.

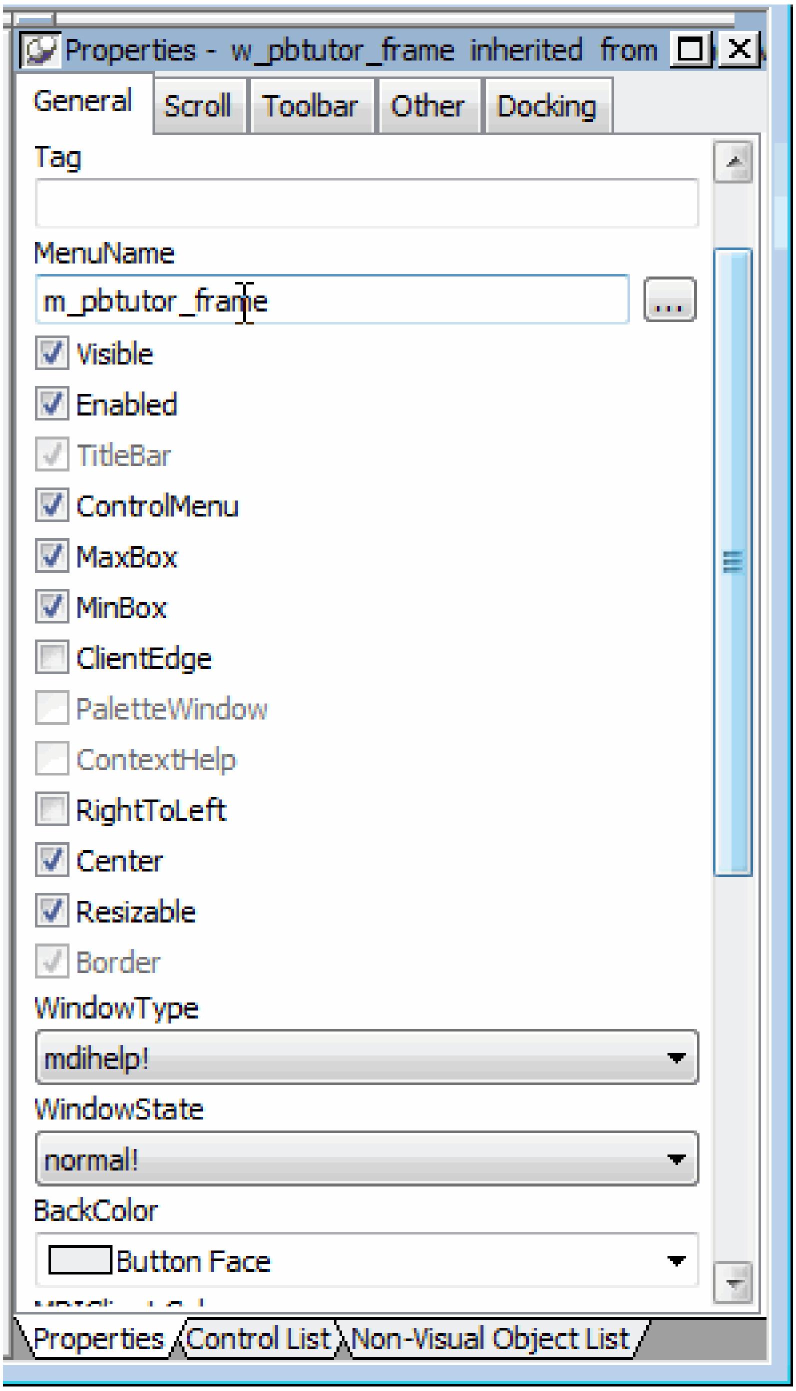
1. Double-click w_pbtutor_frame in the System Tree.

   The Window painter opens the application's frame window.

2. Check the Center check box on the General page in the Properties view.

   Now when you run the application, the frame window will be centered.

3. Scroll down and select normal! in the WindowState drop-down list box.



Tag

MenuName m_pbtutor_frame

☑ Visible
☑ Enabled
☑ TitleBar
☑ ControlMenu
☑ MaxBox
☑ MinBox
□ ClientEdge
□ PaletteWindow
□ ContextHelp
□ RightToLeft
☑ Center
☑ Resizable
☑ Border

WindowType mdihelp!

WindowState normal!

BackColor Button Face

If your Properties view looks different

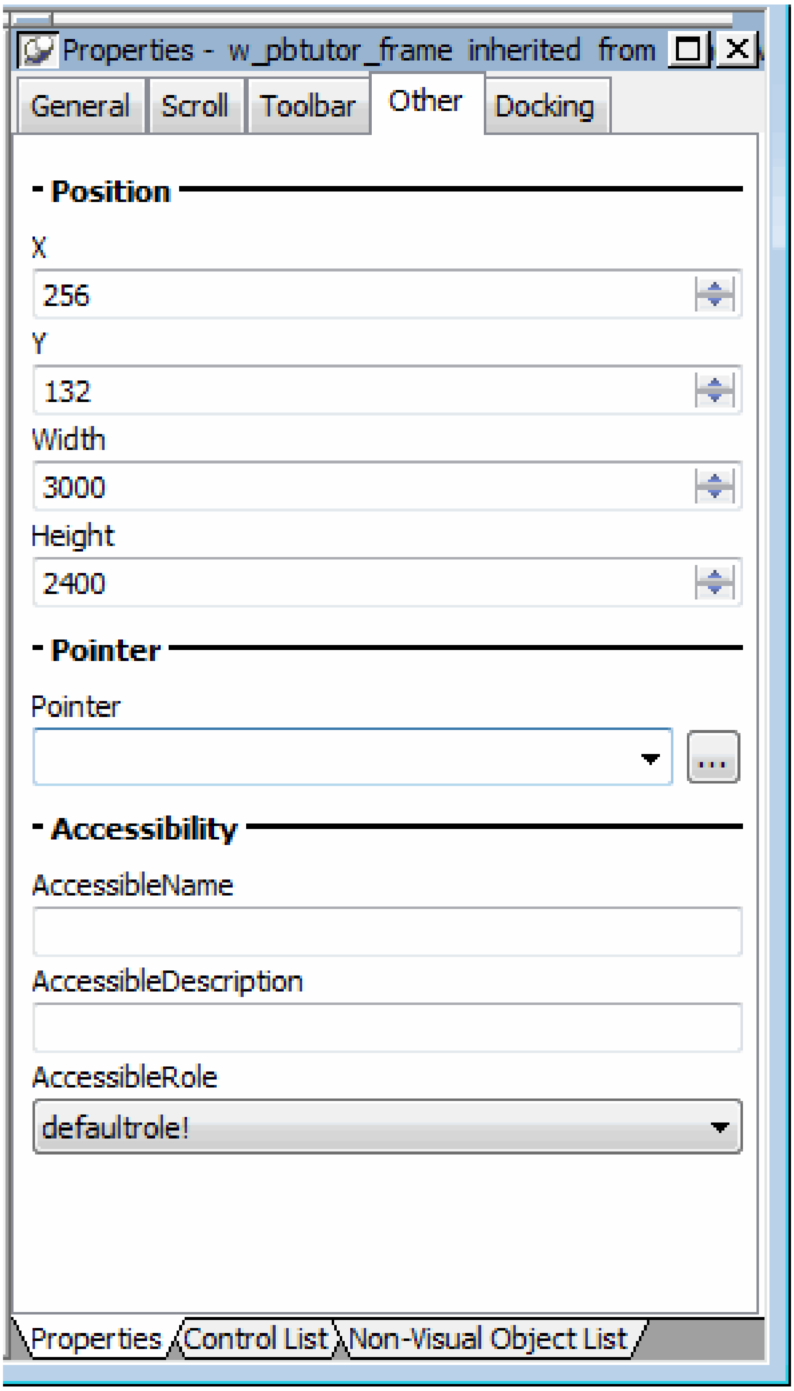
You can change the position of Properties view labels by right-clicking the Properties view and selecting a preference from the pop-up menu. You can position the labels either to the left of all fields, or on top of the text fields and to the right of the check boxes.

4. Click the Other tab in the Properties view.

5. Type 3000 in the Width text box and 2400 in the Height text box.

   The size of the window rectangle in the Layout view changes. The values you type are in PowerBuilder Units (PBUs).

6. Press the Tab key.



Properties - w_pbtutor_frame inherited from

Position

X
256

Y
132

Width
3000

Height
2400

Pointer

Pointer

Accessibility

AccessibleName

AccessibleDescription

AccessibleRole
defaultrole!

7. Select File>Close from the PowerBuilder menu.

   Click Yes when you are prompted to save your changes.

   The Window painter closes.

Next you run the application. When you run the application, the frame window will be centered and sized as you specified.

2.1.5 Run the application

Where you are

Create a new workspace

Create a target

Specify an icon for the application

Change the size of the main window

> Run the application

Now you run the application to see how it works. At this point the application does not do very much. By running the application, you can see the windows and menus that were created for you when PowerBuilder generated the application based on your choices. You will modify them later.

1. Click the Run button (Run) on the PowerBar.

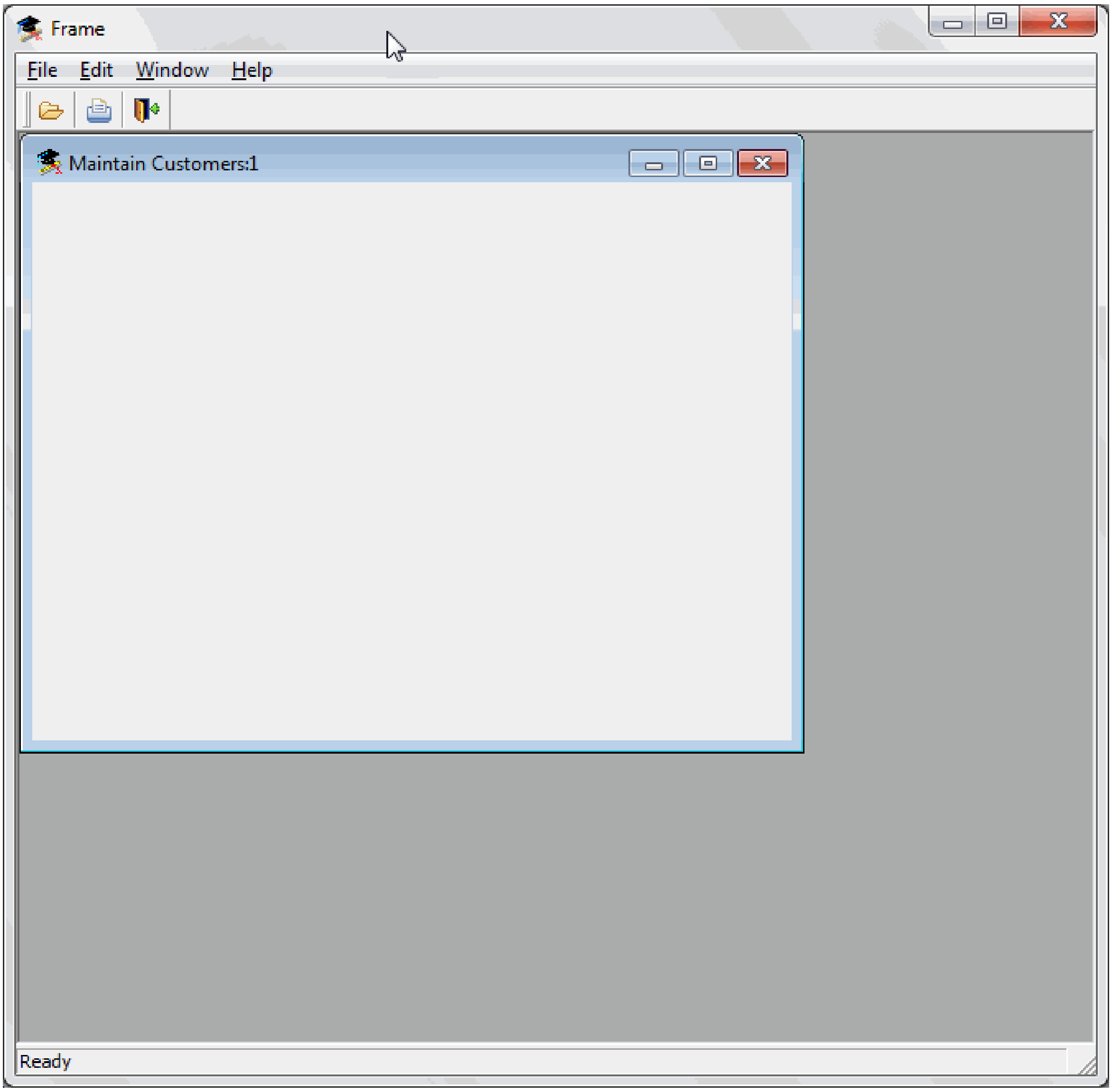
   The MDI frame window displays and is maximized. All MDI applications created using the Wizard have a menu bar and a toolbar with some items already coded for you.

2. Select File>New>Maintain Customers.

   The application opens a sheet window. The display name that you typed in the Template Application wizard for Sheet 1 appears in the title bar. The sheet window title has a number after it to indicate the instance of the window that displays.

About the number in the window title bar

The number 1 appears following the window title because this is the first instance of the w_customers sheet window that is open. The code that adds the instance number to the title bar is in the ue_postopen event of the w_master_detail_ancestor base sheet window.



3. Select File>New>Maintain Products.

   A second application sheet window displays.

4. Select Window>Tile Horizontal.

   The sheet windows are arranged horizontally inside the MDI frame, with the active sheet window at the top.

5. Select File>Toolbars from the menu bar.

   The application displays the Toolbars dialog box.

6. Select Floating in the Toolbars dialog box.

   The toolbar floats within the MDI frame. You might need to move the Toolbars dialog box to see the floating toolbar.

7. Select Top.

   The toolbar is repositioned at the top of the frame.

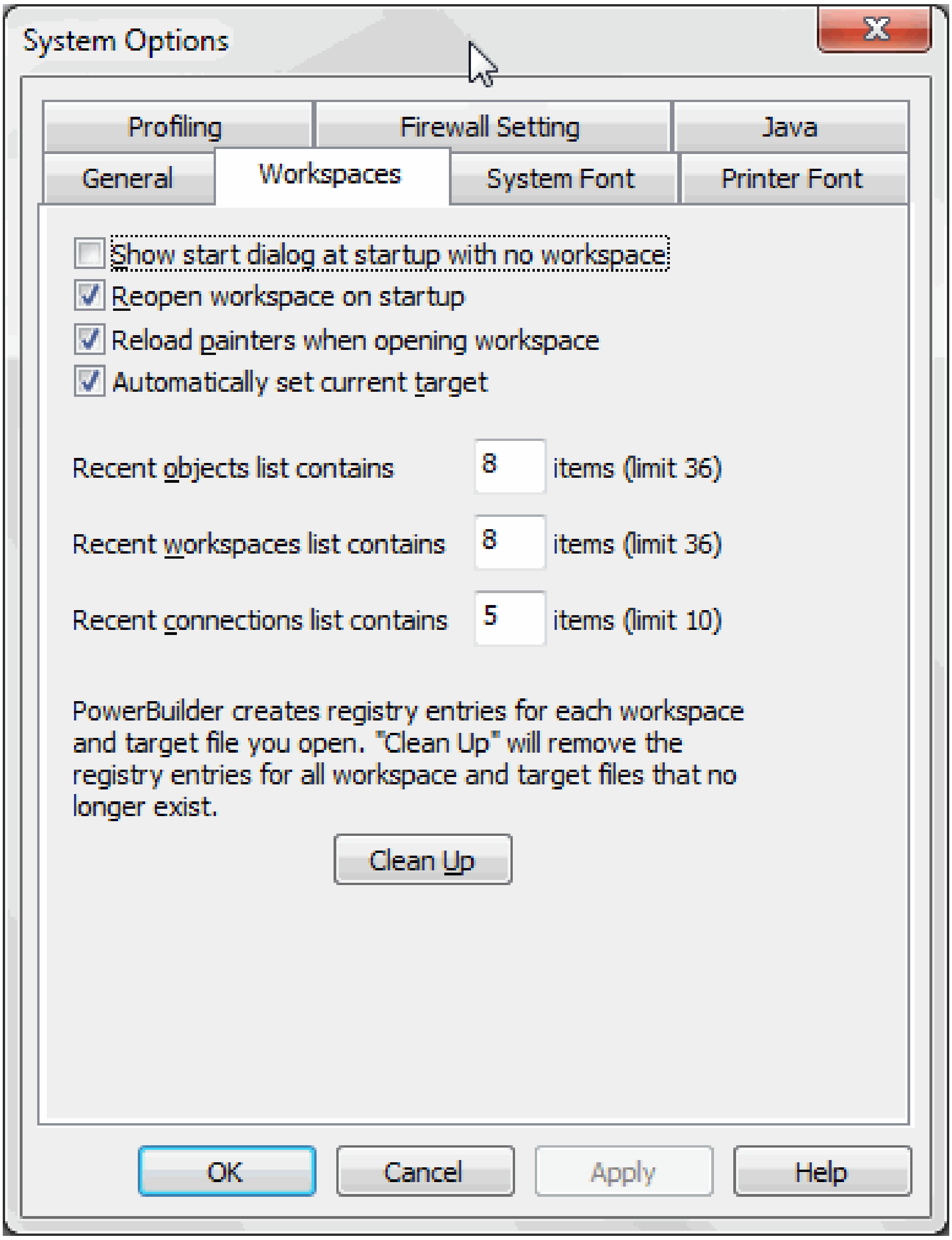
8. Click Done to close the Toolbars dialog box.

9. Select File>Exit.

   The application closes and you return to the PowerBuilder development environment.

   When you exit and restart PowerBuilder, you might want to have PowerBuilder in the state it was in when you exited, with the workspace and painters you were working in open.

10.Select Tools>System Options from the menu bar and then click the Workspaces tab.



☐ Show start dialog at startup with no workspace
☑ Reopen workspace on startup
☑ Reload painters when opening workspace
☑ Automatically set current target

Recent objects list contains 8 items (limit 36)

Recent workspaces list contains 8 items (limit 36)

Recent connections list contains 5 items (limit 10)

PowerBuilder creates registry entries for each workspace and target file you open. "Clean Up" will remove the registry entries for all workspace and target files that no longer exist.

11.Make sure the Reopen Workspace On Startup and the Reload Painters When Opening Workspace check boxes are selected.

Click OK.

Now when PowerBuilder starts up, it opens the workspace and the painters that were open when you exited. If you were coding in PowerBuilder when you exited, the last script you were working on opens at the last line you edited.

2.2 LESSON 2 Customizing the PowerBuilder Environment

This lesson provides the information you need in order to become familiar with the PowerBuilder environment and to customize the workspace. This lesson is optional -- you can skip to Lesson 3 if you want to.

In this lesson you:

- Manipulate the System Tree window
- Open an object
- Manipulate views
- Set up the toolbars

How long does it take?

About 25 minutes.

2.2.1 Manipulate the System Tree window

Where you are

> Manipulate the System Tree window

Open an object

Manipulate views

Set up the toolbars

The Workspace page in the System Tree provides you with an overview of your work. By expanding the workspace and the objects it contains, you can see the content and structure of your target.

You can work directly with all the objects in the workspace. For example, you can edit, run, search, or regenerate a window using its pop-up menu in the System Tree. In this exercise you reposition, close, and open the System Tree. You can reposition the System Tree in relation to the main window using its drag bar. You can also change the way the System Tree, Clip, and Output windows are arranged.

1. Click the Output button (Output) in the PowerBar to display the Output window.

2. Select Tools>System Options from the menu bar.

   Clear the Horizontal Dock Windows Dominate check box on the General page and click OK.

   The System Tree and Clip windows now occupy the full height of the main window.

3. Click and hold the drag bar at the top of the System Tree.

   Drag the System Tree to position it above, below, or to the right of the painter workspace.

   The painter workspace is the gray (blank) area, initially to the right of the System Tree, where painters display when you open an object.

   When you start dragging the System Tree, a gray rectangular outline displays. It indicates the area that the System Tree would occupy if you released the mouse button.

4. When the gray rectangular outline is positioned where you want the System Tree to display, release the mouse button.

   The System Tree displays in the new location.

5. Close the System Tree by clicking the SysTree button (SysTree) in the PowerBar.

   The current workspace remains open, but the System Tree closes. Closing the System Tree leaves more space for the painter workspace views.

6. Reopen the System Tree by clicking the SysTree button in the PowerBar again.

7. Select Tools>System Options from the menu bar.

   Select the Horizontal Dock Windows Dominate check box on the General page and click OK.

   You change back to the default selection for this design-time property.

8. Close the Clip and Output windows by clicking their buttons on the PowerBar (Output Clip) or by clicking the small x in the corner of each window.

9. Right-click MyWorkspace and select Close from the pop-up menu.

   The workspace closes. No workspaces display in the System Tree.

2.2.2 Open an object

Where you are

Manipulate the System Tree window

> Open an object

Manipulate views

Set up the toolbars

Now you open an object created by the Template Application wizard.

1. Select File>Recent Workspaces from the menu bar, then MyWorkspace from the cascading menu.

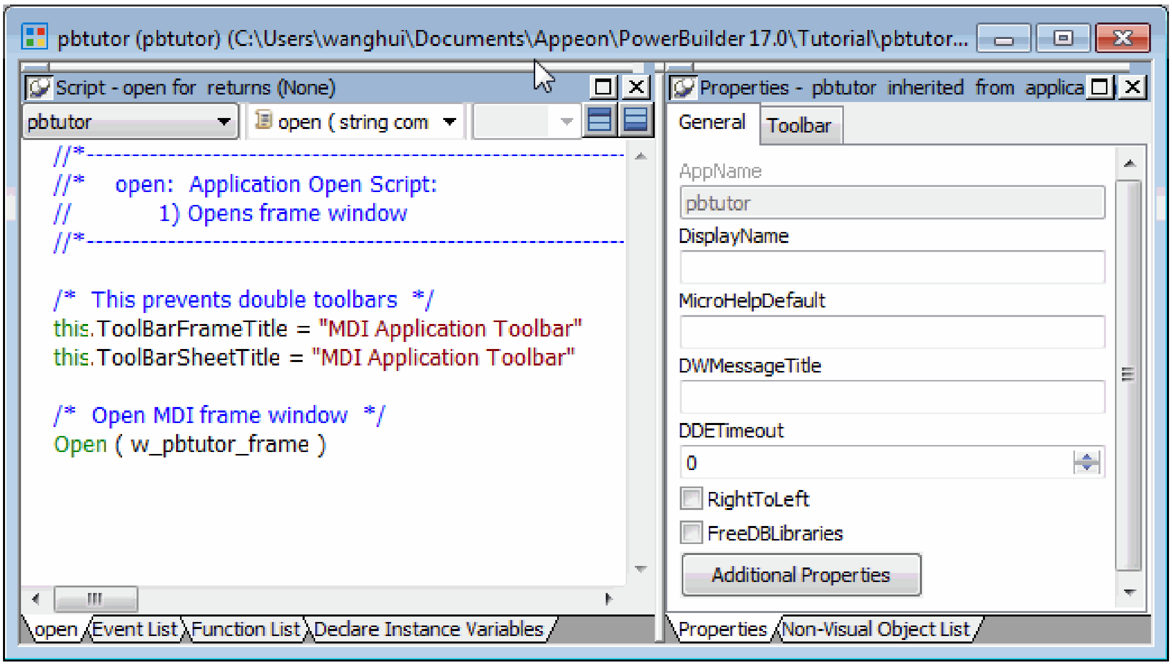
2. In the System Tree, expand MyWorkspace, the pbtutor target, and pbtutor.pbl.

3. Double-click the pbtutor Application object

or

Right-click the pbtutor Application object and select Edit from the pop-up menu.

The Application painter opens. It displays different views of the pbtutor Application object. Your view layout scheme may look different. To display the default layout, select View>Layouts>Default.


pbtutor (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial\pbtutor...
Script - open for returns (None)
pbtutor
open (string com
Properties - pbtutor inherited from applica
General
Toolbar
Additional Properties
open
Event List
Function List
Declare Instance Variables
Properties
Non-Visual Object List

```
//*-----------------------------------------------------------------
//*    open:  Application Open Script:
//           1) Opens frame window
//*-----------------------------------------------------------------

/*  This prevents double toolbars  */
this.ToolBarFrameTitle = "MDI Application Toolbar"
this.ToolBarSheetTitle = "MDI Application Toolbar"

/*  Open MDI frame window  */
Open ( w_pbtutor_frame )
```

AppName: pbtutor

DisplayName:

MicroHelpDefault:

DWMessageTitle:

DDETimeout: 0

□ RightToLeft

□ FreeDBLibraries

The default Application painter layout displays two stacks of tabbed panes. The left stack contains tabs for a Script view (Open tab -- it is set to the Open event on the Application object), an Event List view, a Function List view, and the Declare Instance Variables view. The right stack contains tabs for the Properties view and a Non-Visual Object List view.

4. Look at the code in the Open event in the Script view.

   The PowerScript code that was generated by the wizard in the Application Object Open event calls a PowerScript function to open the main window in the application. You will modify this code later in the tutorial.

2.2.3 Manipulate views

Where you are

Manipulate the System Tree window

Open an object

> Manipulate views

Set up the toolbars

Now you learn to control the location and appearance of PowerBuilder painter views. You can add views to a painter workspace by selecting them from the View menu in the workspace menu bar.

You can add multiple views of the same type and you can combine views into a stack of panes with selection tabs at the bottom. You can resize a view by grabbing and dragging the separator bars that surround it or that surround neighboring views in the painter workspace.

These exercises demonstrate how you can change the appearance of Application painter views, but you can manipulate views in all painters in the same way.

Now you:

- Add an extra Script view
- Display view title bars
- Float and dock views
- Manipulate tabbed views
- Save a view layout scheme
- Reset the default view layout scheme

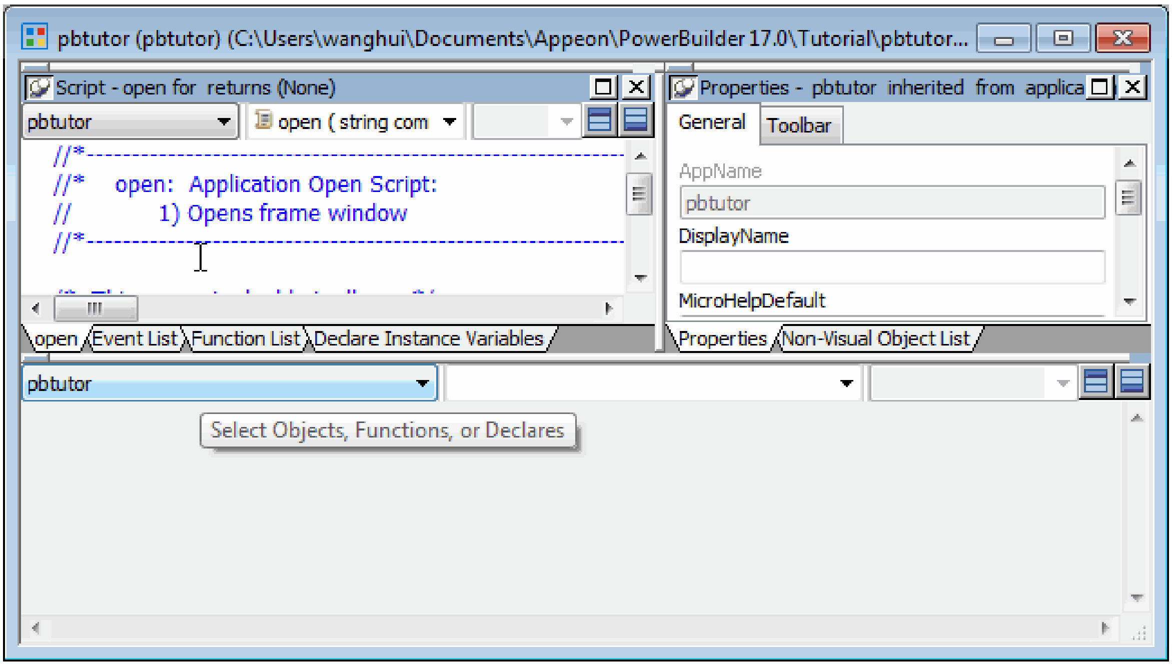
2.2.3.1 Add an extra Script view

The default Application painter layout actually has two Script views. One of the Script views displays the script for an Application object event, and the other Script view displays the declared variables for the object instance or the entire application. Both of these Script views are in the same stack of tabbed views (panes).

Now you add a third Script view that is not part of a stack of tabbed panes. You can add multiple Script views to your painter layout, but no two Script views can display the same script at the same time.

1. Select View>Script from the menu bar.

   A new Script view displays. It is not attached to a stack of tabbed panes. It lists the Application object in the left drop-down list box. The other two drop-down lists are empty and the right drop-down list is grayed out.



pbtutor (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial\pbtutor...
Script - open for returns (None)
pbtutor
open (string com
Properties - pbtutor inherited from applica
General
Toolbar
open
Event List
Function List
Declare Instance Variables
Properties
Non-Visual Object List
pbtutor
Select Objects, Functions, or Declares

```
//*---------------------------------------------------------
//*    open:  Application Open Script:
//         1) Opens frame window
//*---------------------------------------------------------
```

AppName: pbtutor

DisplayName:

MicroHelpDefault

If an existing Script view shows the Open event, the new Script view is empty. Otherwise it displays the Open event.

2. Select the Close event from the second drop-down list box.

   If another Script view is already open to the Close event, an error message displays in the PowerBuilder status bar.

2.2.3.2 Display view title bars

Now you display a view title bar by pinning it to the painter workspace background. If a title bar is unpinned, you see it only when your cursor pauses near the top edge of a view.

1. Move the cursor to the top of the extra Script view you just added.

   The view title bar rolls down. It contains a pushpin button on the left and a maximize/minimize button and a close button on the right. The name of the view displays on the left side of the title bar, next to the pushpin button.

2. Click the pushpin (📌) in the title bar

   or

   Right-click the view title bar and click Pinned from the pop-up menu.

   The pushpin button and the Pinned menu item are toggle switches. You can click the pushpin button or the pop-up menu item to pin and unpin the view title bars.

2.2.3.3 Float and dock views

Now you float and dock a view in the painter workspace. Floating a view enables you to move it around outside the painter frame.

1. Right-click the title bar of an unstacked view you want to float

or

Right-click the tab of a view in a stack of tabbed panes.

If the title bar is not pinned, move the cursor over the title bar area and wait until it displays before you right-click it.

2. Click Float in the pop-up menu.

   When a view is floated, the Float menu item is not enabled. When a view is docked, the Dock menu item is not enabled.

3. Drag the view around the screen.

   Notice that the floating property allows you to move the view outside the painter workspace.

4. Right-click the title bar of the floating view.

   Click Dock in the pop-up menu.

   The view returns to its original location.

2.2.3.4 Manipulate tabbed views

Now you separate a view from a stack of tabbed panes and place it above the stack. You then return it to the stack and change its position in the stack.

1. Press and hold the mouse button on the Function List tab.

   Drag the tab onto the separator bar that separates the two default stacks in the Application painter.

   Release the mouse button.

   When you release the mouse button, the Function List view is no longer part of a stack. If you drag the tab too far and release it over the right stack with the Properties view and Non-Visual Object List, the Function List becomes part of that stack.

Alternate way to float a view from a stack

If you hold the Ctrl or Shift key down as you drag a tabbed pane from a stack, the pane becomes a floating view.

2. Press and hold the mouse button on the Function List title bar.

   Drag it over the stack from which you separated it.

   Release the mouse button when the gray rectangular outline of the Function List view overlaps the stack.

   The Function List view returns to its original stack, but it is added as the last pane in the stack.

3. Press and hold the mouse button on the Function List tab.

   Drag it sideways over the other tabs in the same stack.

Release the mouse button when the small gray rectangular outline overlaps another tab in the stack of tabbed panes.

The Function List view moves to the position in the stack where you release the mouse button.

2.2.3.5 Save a view layout scheme

You can save view layout schemes for a PowerBuilder painter and use them every time you open the painter.

1. Arrange the views in the painter as you like.

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

3. Click the New Layout button in the Layout dialog box.

4. Enter a name for your layout in the text field, click the background of the dialog box, and then click the x button in the upper right corner of the dialog box to close it.

   Your layout scheme is saved. Now, when you select View>Layouts, you see your layout listed on the cascading menu.

Saving the toolbars and System Tree layouts

PowerBuilder saves the customizations you make to the toolbars and System Tree separately from the view layout. It retains those settings and reapplies them to every workspace you access and every view layout you select.

2.2.3.6 Reset the default view layout scheme

Each PowerBuilder painter has a default view layout scheme. You can always reset the layout scheme to this default layout.

1. Select View>Layouts from the menu bar.

2. Choose Default from the cascading menu.

   The default view layout scheme displays in the painter workspace.

2.2.4 Set up the toolbars

Where you are

Manipulate the System Tree window

Open an object

Manipulate views

> Set up the toolbars

A painter workspace always includes the PowerBar and other PainterBar toolbars that you can use as you work. The buttons in the toolbars change depending on the type of target or object you are working with. You can also customize the toolbars to include additional functionality.

Now you change the appearance of the toolbars to:

- Show labels on toolbar buttons
- Float the toolbars
- Reposition the toolbars

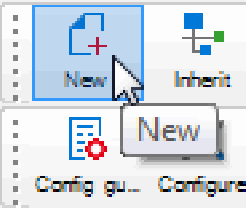
2.2.4.1 Show labels on toolbar buttons

You can learn a toolbar button's function by placing the cursor over it to view its PowerTip. A PowerTip is pop-up text that indicates a button's function.

You can also display a label on each toolbar button.

1. Move the pointer to any button on the PowerBar, but do not click.

   The button's PowerTip displays.



2. Select Tools>Toolbars from the menu bar.

   The Toolbars dialog box displays.

3. Select the Show Text check box, then click the Close button.

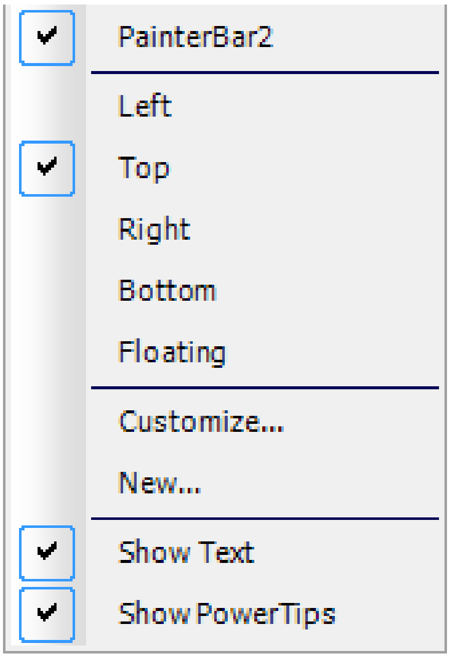
   PowerBuilder displays a label on each of the buttons in the PowerBar and the PainterBars.

2.2.4.2 Float the toolbars

You can float the toolbars so that you can move them around the painter workspace as you work.

1. Right-click anywhere in the PowerBar.

   The pop-up menu for the toolbars displays. From the pop-up menu you can set the toolbar's location to the left, top, right, or bottom of the workspace. You can also set it to floating.

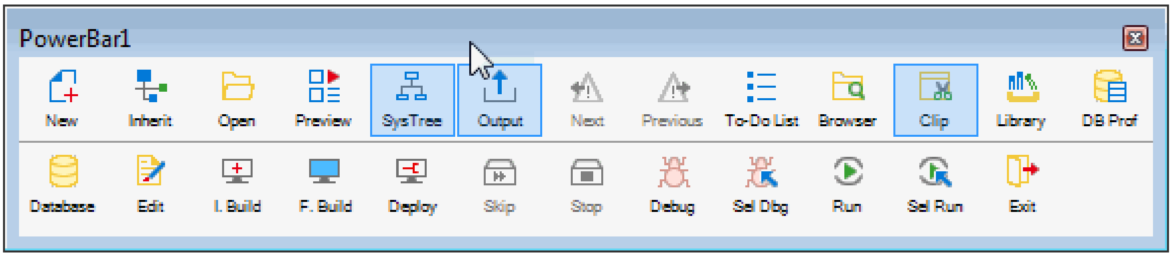


About pop-up menus

Throughout PowerBuilder, pop-up menus provide a fast way to do things. The menu items available in the pop-up depend on the painter you are using and where you are in the workspace when you click the right mouse button.

2. Select Floating from the pop-up menu.

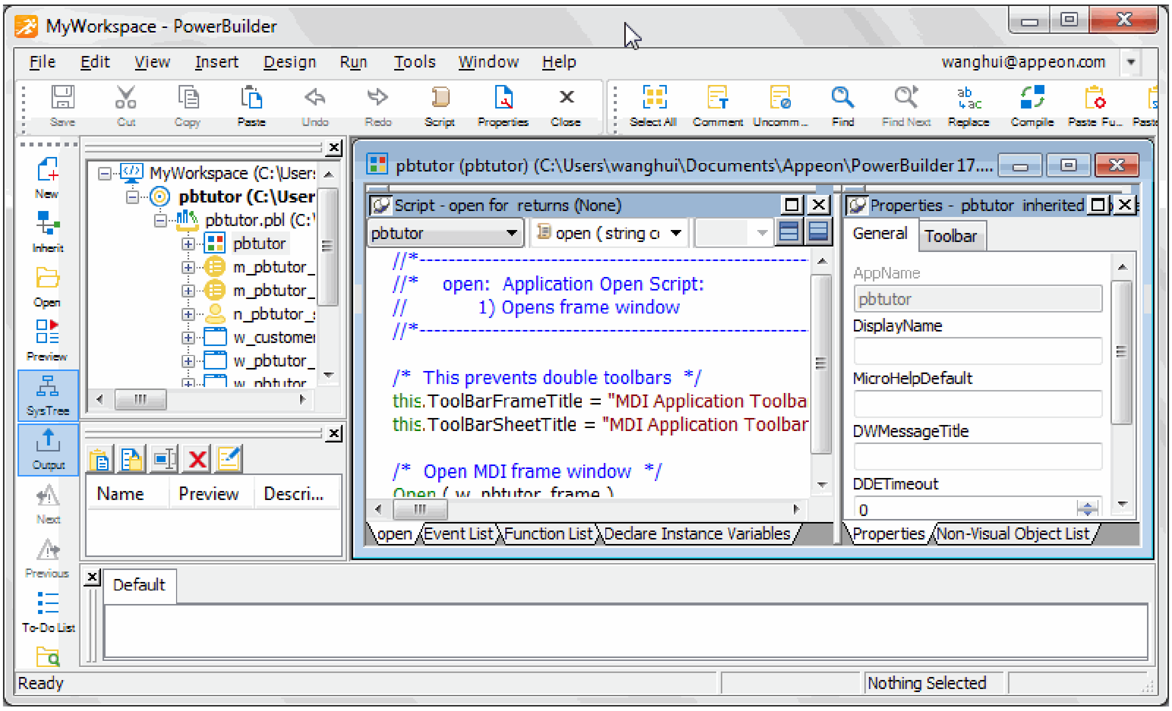
   The PowerBar changes to a floating toolbar. You can adjust its shape.



3. Move the pointer to an edge or border area in the PowerBar.

4. Press and drag the PowerBar toward the left side of the workspace.

   Release the mouse button when the PowerBar becomes a vertical bar.


MyWorkspace - PowerBuilder
File Edit View Insert Design Run Tools Window Help
wanghui@appeon.com
Save Cut Copy Paste Undo Redo Script Properties Close Select All Comment Uncomm... Find Find Next Replace Compile Paste Fu...
New Inherit Open Preview SysTree Output Next Previous To-Do List
MyWorkspace (C:\User
pbtutor (C:\User
pbtutor.pbl (C:\
pbtutor
m_pbtutor_
m_pbtutor_
n_pbtutor_
w_customer
w_pbtutor_
w_pbtutor
Name Preview Descri...
pbtutor (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17....
Script - open for returns (None)
pbtutor
open (string c
open Event List Function List Declare Instance Variables
Properties - pbtutor inherited
General Toolbar
Properties Non-Visual Object List
Default
Ready
Nothing Selected

```
//*-----------------------------------------------
//*    open:  Application Open Script:
//          1) Opens frame window
//*-----------------------------------------------

/*  This prevents double toolbars  */
this.ToolBarFrameTitle = "MDI Application Toolba
this.ToolBarSheetTitle = "MDI Application Toolbar

/*  Open MDI frame window  */
Open ( w_pbtutor_frame )
```

AppName: pbtutor

DisplayName:

MicroHelpDefault:

DWMessageTitle:

DDETimeout: 0

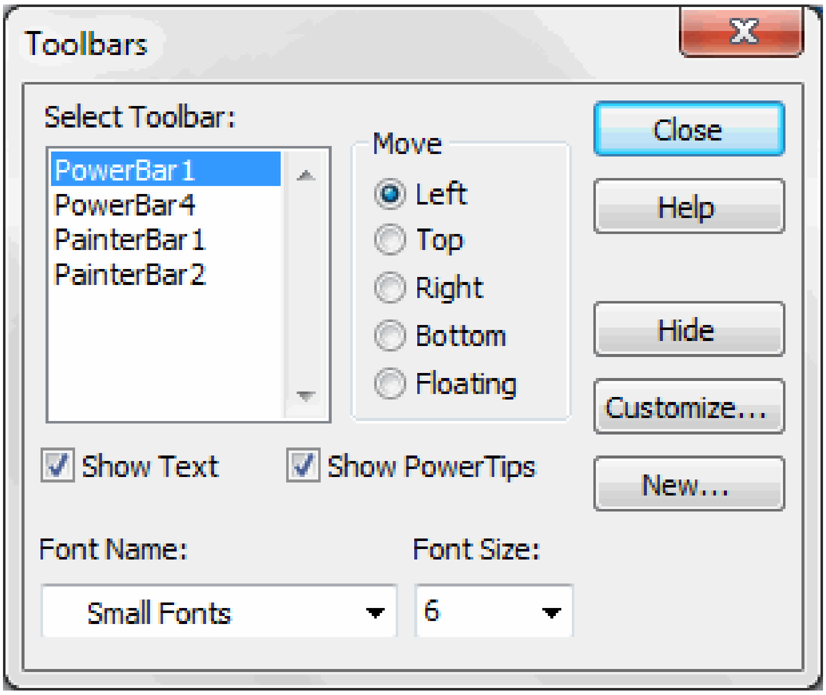
The PowerBar is docked at the left side of the frame.

2.2.4.3 Reposition the toolbars

You can customize the position of the toolbars to suit your work style.

1. Select Tools>Toolbars from the menu bar.

   The Toolbars dialog box displays. The selected Move radio button indicates the position of the currently selected toolbar.



Select Toolbar: PowerBar1, PowerBar4, PainterBar1, PainterBar2

Move: ● Left ○ Top ○ Right ○ Bottom ○ Floating

☑ Show Text ☑ Show PowerTips

Font Name: Small Fonts

Font Size: 6

2. Click Top.

   This repositions the PowerBar at the top of the workspace.

   Radio buttons are grayed if a selected toolbar is hidden

   If a selected toolbar is hidden (not visible) in the painter, you cannot select where it appears in the workspace. In this case, the radio buttons are grayed and you must first click the Show button before you can select a radio button. The Show button replaces the Hide button when a toolbar is hidden.

3. Click PainterBar1 in the Select Toolbar list box and select Right.

   Click Close in the Toolbars dialog box.

4. Right-click PainterBar2 and select Left from the pop-up menu.

   You have swapped the locations of the two painter bars.

5. Arrange the toolbars to suit your preferences.

   You can also drag the toolbars to the top, bottom, left, or right of the painter workspace. When a toolbar is in a fixed location, it has a drag bar at the left or top of its buttons. You can click the drag bar and drag the mouse to move the toolbar around the painter workspace.

   PowerBuilder applies toolbar configuration properties to all painters and saves them for the next PowerBuilder session.

6. Close the Application painter.

   If you are not continuing immediately with the tutorial

   You can close PowerBuilder or the tutorial workspace if you want. In that case, you must open the tutorial workspace before you continue with the next lesson.

2.3 LESSON 3 Building a Login Window

Windows are the main interface between users 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 anytime during the application development process.

In this lesson you:

- Create a new window
- Add controls to the window
- Change the tab order on the window
- Code some Help events and preview the window

- Write the script to open the window

How long does it take?

About 25 minutes.

2.3.1 Create a new window

Where you are

> Create a new window

Add controls to the window

Change the tab order on the window

Code some Help events and preview the window

Write the script to open the window

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

About response windows

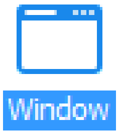
Response windows are dialog boxes that require information from the user. Response windows are application modal. When a response window displays, it is the active window (it has focus) and no other window in the application is accessible until the user responds. The user can go to other applications, but when the user returns to the application, the response window is still active.

1. Click the New button (New) in the PowerBar.

   The New dialog box displays.

2. Click the PB Object tab.

3. Select the Window icon and click OK.

   The Window painter opens. Notice that you have two new toolbars, the StyleBar (with character style and text alignment buttons) and PainterBar3 (with color and border buttons, as well as grayed out control alignment buttons).

4. Make sure that the Layout view and the Properties view display in the Window painter.

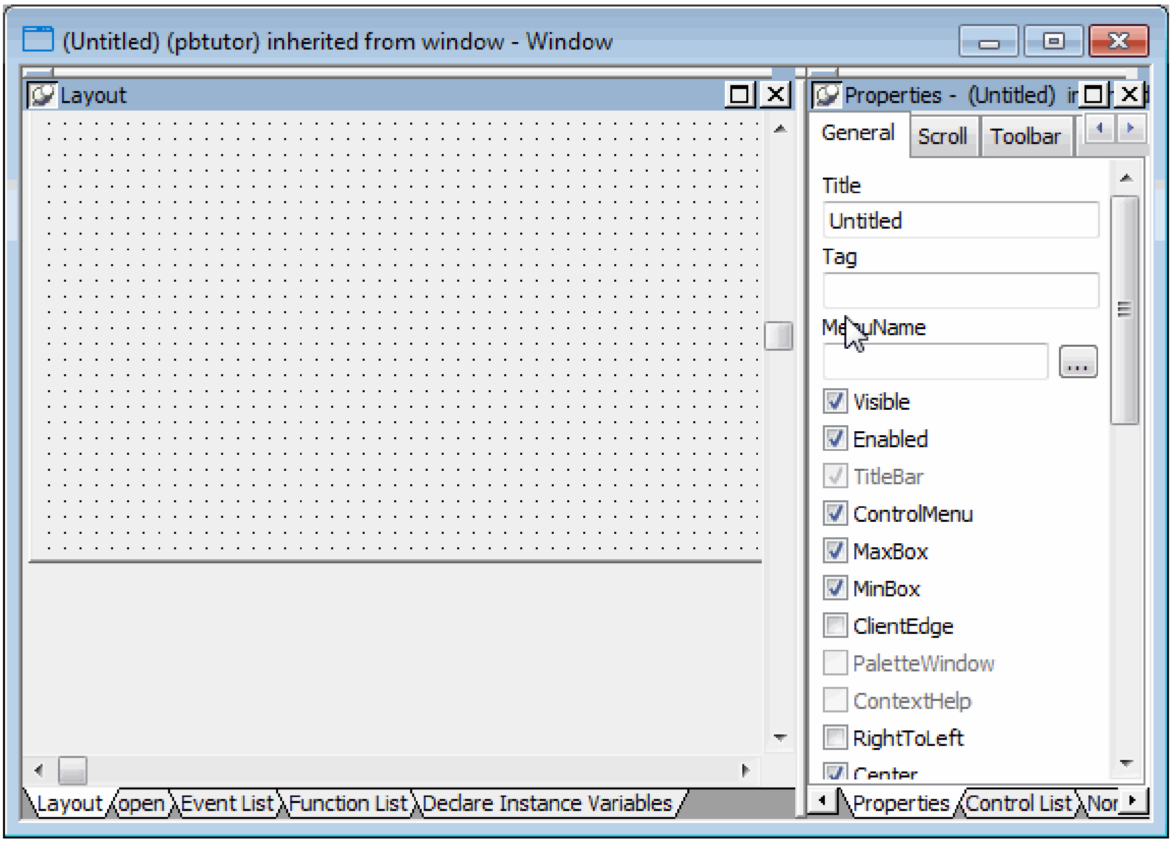
   You can display these views by selecting them from the View menu. If they are grayed out in the menu, the views are already displayed in the painter.

   The default view layout scheme contains both views.

To retrieve the default painter layout

Select View>Layouts>Default from the workspace menu bar.

The rectangle in the Layout view represents the window you are building. The default properties in the Properties view indicate that the window is visible and enabled, and has the Main window type. You might need to scroll down in the Properties view to see the window type.


(Untitled) (pbtutor) inherited from window - Window
Layout
Properties - (Untitled) ir
General
Scroll
Toolbar
Layout
open
Event List
Function List
Declare Instance Variables
Properties
Control List
Nor

Title
Untitled
Tag
MenuName
☑ Visible
☑ Enabled
☑ TitleBar
☑ ControlMenu
☑ MaxBox
☑ MinBox
☐ ClientEdge
☐ PaletteWindow
☐ ContextHelp
☐ RightToLeft
☑ Center

If your window does not have a pegboard look

If the window in your Layout view displays as a solid color, the Show Grid option has been disabled. To enable it, select Design>Options from the menu bar. Then select the Show Grid check box on the General page of the Options dialog box. Click Apply, then OK to save the change and close the dialog box.

5. Type Welcome in the Title text box on the General page of the Properties view.

6. Select response! in the Window Type drop-down list box.

   Make sure the TitleBar and ControlMenu check boxes are selected.

   Select the ContextHelp check box.

   The ContextHelp property adds a question-mark button next to the (ControlMenu) close button in the login window's title bar. Users of your application can click the question-mark button to trigger Help events for the window controls. You can add a question-mark button to a response window, but not to a main window.

7. Click the Other tab in the Properties view.

   Type 2300 in the Width text box and 1000 in the Height text box.

   Press the Tab key.

   The size of the window rectangle in the Layout view changes. The values you type are in PowerBuilder Units (PBUs). You might need to modify these values later, while you are adding controls to the window.

   Other methods for entering position properties

   You can use the spin controls (↕) to enter values instead of typing them.

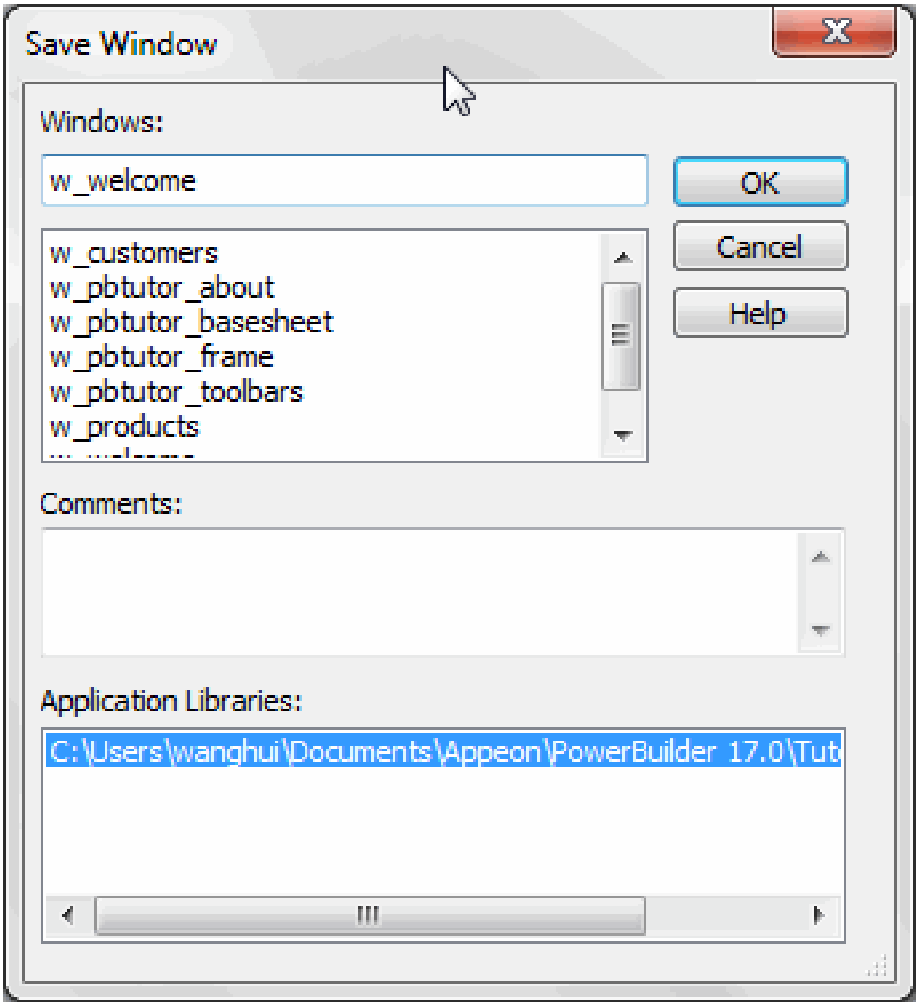
   Alternatively, you can change the size of the login window in the Layout view by moving the pointer to the bottom or right edge of the window. When it turns into a double-headed arrow, you can drag the arrow to change the window size.

8. Select File>Save from the menu bar.

   The Save Window dialog box displays. The only library in the Application Libraries text box is pbtutor.pbl, and it is selected.

9. Type w_welcome for the window name.

   The prefix w_ is standard for windows.



10.(Optional) Type the following lines in the Comments text box:

```
This is the login window. It requires the application user to enter an ID and a
 password before continuing.
```

These comments are visible in the List view of the Library painter.

11.Click OK.

PowerBuilder saves the new login window. If you expand MyWorkspace, pbtutor, and pbtutor.pbl in the system tree, you can see w_welcome listed under it.

2.3.2 Add controls to the window

Where you are

Create a new window

> Add controls to the window

Change the tab order on the window

Code some Help events and preview the window

Write the script to open the window

Controls allow users to interact with PowerBuilder objects, such as windows and DataWindows. You set properties of these controls and later add script for control events and functions.

Selecting a control button from the PainterBar

You can add controls from the Insert menu or by selecting a control button in PainterBar1. Unless you customize your toolbars, there is only one control button that appears in the PainterBar. When you first open a painter, PainterBar1 includes the CommandButton control button, which has a down arrow to its right. Clicking the down arrow displays a drop-down list of control buttons.

Some of the controls you can select from the drop-down list are described in the table below:

Table 2.1:

Button appearance	Control type	Use in tutorial
Button	CommandButton	Default icon for the control button in PainterBar1. You add command buttons later in this lesson.
PctHLink	Picture	To add a picture to the login window.
Picture	PictureHyperlink	Not used in tutorial. Its purpose is to provide a link to a website.
PctBtn	PictureButton	Not used in tutorial. It is like a command button, but it displays a picture as well as text.
Text	StaticText	To add text labels to the login window.
HyperLink	StaticHyperLink	To provide a link to a website.
SngleEdit	SingleLineEdit	To add user entry text boxes to the login window.
MultiEdit	MultiLineEdit	Not used in tutorial. Its purpose is to add a multiline edit text box.

After you select a control, it appears in place of the CommandButton button on PainterBar1.

Adding controls with a 3D appearance

To make your controls look three dimensional, select Design>Options from the menu bar and make sure that the Default To 3D check box is selected on the General page of the Options dialog box. You can change the window background color from the default color of ButtonFace gray using the Properties view in the Window painter.

Now you modify the login window you just created by adding controls and changing some of their properties. You:

- Add a Picture control
- Add StaticText controls
- Specify properties of the StaticText controls

- Add SingleLineEdit controls
- Specify properties of the SingleLineEdit controls
- Add CommandButton controls
- Specify properties of the CommandButton controls

2.3.2.1 Add a Picture control

Now you add a Picture control to the login window.

1. Select the Picture button (Picture) from the drop-down list of controls

   or

   Select Insert>Control>Picture from the menu bar.

2. Click inside the rectangular window in the Layout view.

   A Picture control displays at the selected location. At the same time you add the control, the Properties view switches from displaying the window properties to displaying the control properties.

   If you do not see the Properties view, select View>Properties from the menu bar. If the Properties view does not display the control properties, click the picture control in the Layout view.

   How to delete controls

   If you add a control to the window and later decide you do not want it, select the control and press the Delete key. This deletes the control and its scripts.

3. Select the text p_1 in the Name text box on the General tab of the Properties view.

   Type p_sports in the Name text box.

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

4. Click the ellipsis button next to the PictureName text box.

   The Select Image dialog box displays.

5. Select the "Use a custom image" option, click the ellipsis button and navigate to the Tutorial directory if it is not already selected.

   Select the tutsport.bmp file.

   The bitmap you selected appears in the control you added to the Layout view. The Visible, Enabled, and OriginalSize check boxes are selected by default in the Properties view.

6. Make sure the picture control is selected in the Layout view.

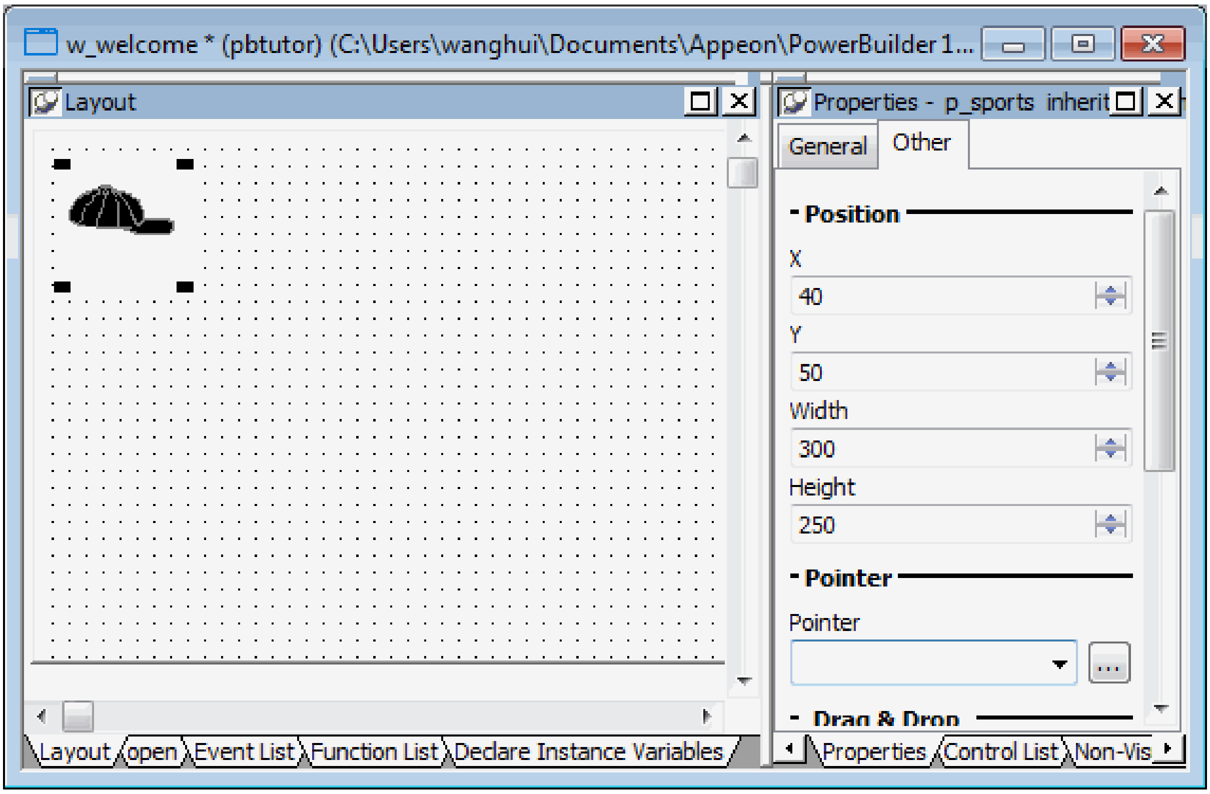
   Click the Other tab in the Properties view.

   Type 40 in the X text box and 50 in the Y text box.

You can use the spin controls in the X and Y text boxes to enter these values. You might want to adjust the position of the picture control again after you preview the window at the end of this lesson.

7. Type 300 in the Width text box and 250 in the Height text box.

You change the size of the picture control. You might want to adjust the picture size again after you preview the window.



w_welcome * (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 1...

Properties - p_sports inherit

Position

X: 40

Y: 50

Width: 300

Height: 250

Pointer

Pointer:

2.3.2.2 Add StaticText controls

Now you add StaticText controls to the login window. You will use these controls to add descriptive labels to the login window.

1. Select the Text button (A) from the drop-down list of controls

   or

   Select Insert>Control>StaticText from the menu bar.

2. Click to the right of the Picture control you added in the Layout view.

   A StaticText control displays at the selected location.

3. Right-click the StaticText control and select Duplicate from the pop-up menu.

   PowerBuilder creates a duplicate of the selected control.

4. Select Duplicate from the StaticText control's pop-up menu again.

   PowerBuilder creates another duplicate.

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

5. Adjust the location of the Static Text controls so that there are at least two grid lines between them.

2.3.2.3 Specify properties of the StaticText controls

Now you specify the properties of the StaticText controls (label text boxes) to define how they should appear on the login window.

1. Select the first StaticText control you added by clicking on it.

   The Properties view displays properties of the StaticText control. If you do not see the Properties view, select View>Properties from the menu bar.

2. Select the text st_1 in the Name text box on the General page of the Properties view.

   Type st_welcome in the Name text box.

   Now the control has a more descriptive name. The prefix st_ is standard for StaticText controls.

3. Select the text none in the Text text box.

   Type Welcome to SportsWear, Inc.

   If you press Enter, click to another field or tab to another page in the Properties view, or click in a different view, the text you typed replaces none in the Layout view.

4. Click the Font tab in the Properties view.

   Change the TextSize property for this control to 18 points.

   The size of the text in the control changes.

   The default typeface is Arial TrueType. You can select a different typeface and font size if this one is not available on your system.

   Using the StyleBar

   You can also use the StyleBar to change fonts. If you do not see the StyleBar, select Tools>Toolbars from the menu bar, click StyleBar in the Select Toolbar list box, and then select one of the Move positions such as bottom or floating.

5. Adjust the size of the StaticText control to fit the text you entered.

   Keep adjusting the size until you see all of the text you entered.

   To adjust the size, drag the upper-right corner of the control toward the upper-right corner of the window in the Layout view. You can also adjust the size by entering appropriate values on the Other page of the Properties view for this control.

6. Select the second StaticText control you added in the Layout view.

   Type st_userid in the Name text box on the Properties view General page.

   Type User ID: in the Text text box and press the Tab key.

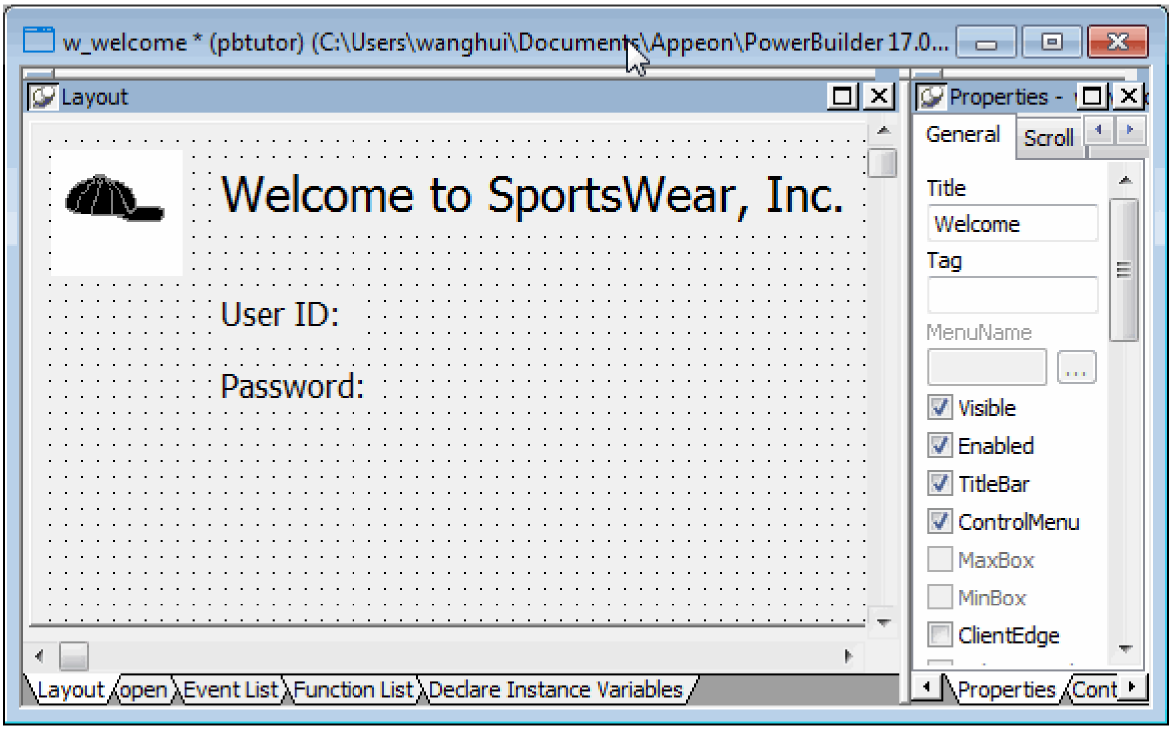
The text displayed in the control changes.

7. Select the third StaticText control you added in the Layout view.

   Type st_password in the Name text box on the Properties view General page.

   Type Password: in the Text text box and press the Tab key.

   Your changes display in the Layout view.



w_welcome * (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0...

Welcome to SportsWear, Inc.

User ID:

Password:

Title: Welcome

Tag:

MenuName:

☑ Visible

☑ Enabled

☑ TitleBar

☑ ControlMenu

□ MaxBox

□ MinBox

□ ClientEdge

2.3.2.4 Add SingleLineEdit controls

Now you 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 text box in which the user can enter a single line of text. SingleLineEdit controls are typically used for the input and output of data.

1. Select the SingleLineEdit button ( ) from the drop-down list of controls

   or

   Select Insert>Control>SingleLineEdit from the menu bar.

2. Click to the right of the st_userid StaticText control in the Layout view.

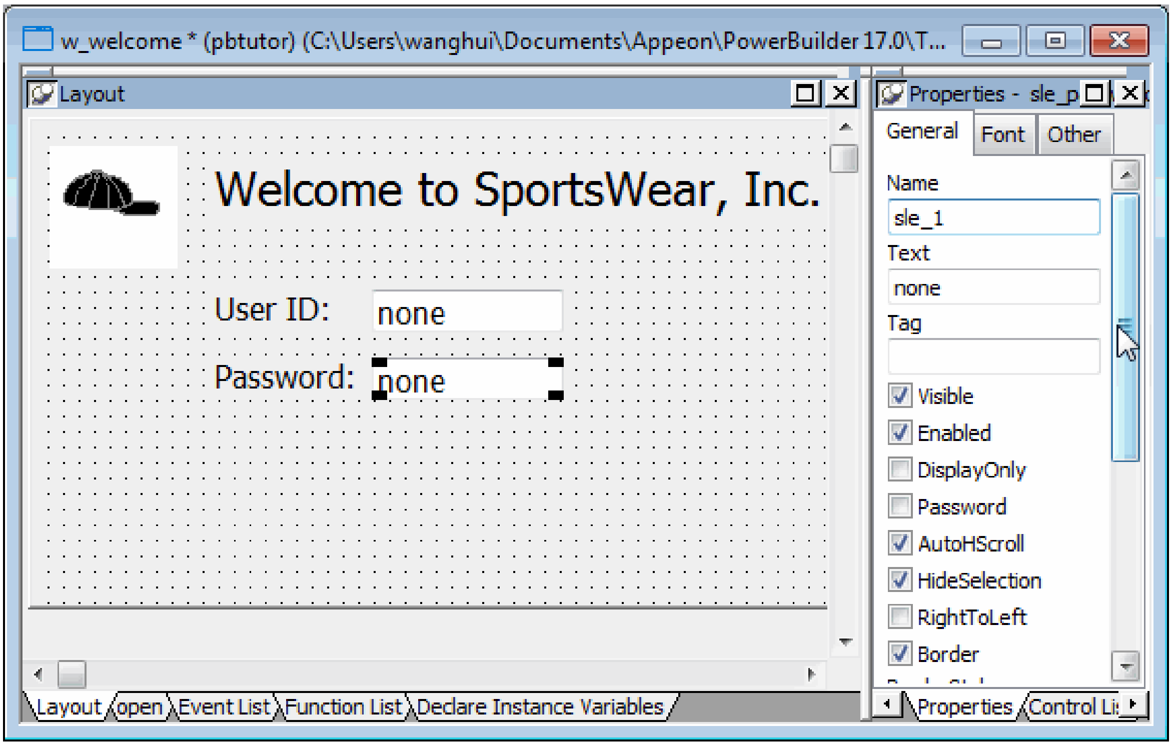
   A SingleLineEdit control displays where you clicked.

3. Increase the width of the SingleLineEdit control.

4. Right-click the SingleLineEdit control and select Duplicate from the pop-up menu.

   Adjust the position of this SingleLineEdit control so that it is just to the right of the st_password StaticText control.

You should now have two SingleLineEdit controls arranged vertically to the right of the StaticText controls.


w_welcome * (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\T...
Layout
Welcome to SportsWear, Inc.
User ID:
none
Password:
none
Properties - sle_p
General
Font
Other
Layout
open
Event List
Function List
Declare Instance Variables
Properties
Control Li

Name: sle_1

Text: none

Tag:

☑ Visible

☑ Enabled

☐ DisplayOnly

☐ Password

☑ AutoHScroll

☑ HideSelection

☐ RightToLeft

☑ Border

2.3.2.5 Specify properties of the SingleLineEdit controls

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

1. Select the first SingleLineEdit control you added.

   The General page of the Properties view displays properties of the SingleLineEdit control. If you do not see the Properties view, select View>Properties from the menu bar.

2. Select the text sle_1 in the Name text box.

   Type sle_userid in the Name text box.

   Clear the default text none in the Text text box.

   The prefix sle_ is standard for SingleLineEdit controls.

3. Select the second SingleLineEdit control you added.

   Type sle_password in the Name text box.

   Clear the default text none in the Text text box.

   Select the Password check box.

   Because you checked the Password check box, the password the user types at runtime will display as a string of asterisks.

2.3.2.6 Add CommandButton controls

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

1. Select the CommandButton button (cmd Button) from the drop-down list of controls

   or

   Select Insert>Control>CommandButton from the menu bar.

2. Click to the right of the sle_userid SingleLineEdit control.

   A CommandButton control displays where you clicked.

3. Right-click the CommandButton control and select Duplicate from the pop-up 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.

2.3.2.7 Specify properties of the CommandButton controls

Now you define the properties of the CommandButton controls.

1. Select the top CommandButton control.

   The General page of the Properties view displays properties of the CommandButton control.

2. Type cb_ok in the Name text box on the Properties view General page.

   Type OK in the Text text box.

   Select the Default check box.

   This step changes the default name of the control to something more descriptive and adds a text label (OK) to the button. Because you selected the Default check box, when an application user presses the Enter key, the Clicked event for this button will be triggered. (The user does not have to click the button itself for the event to be triggered.)

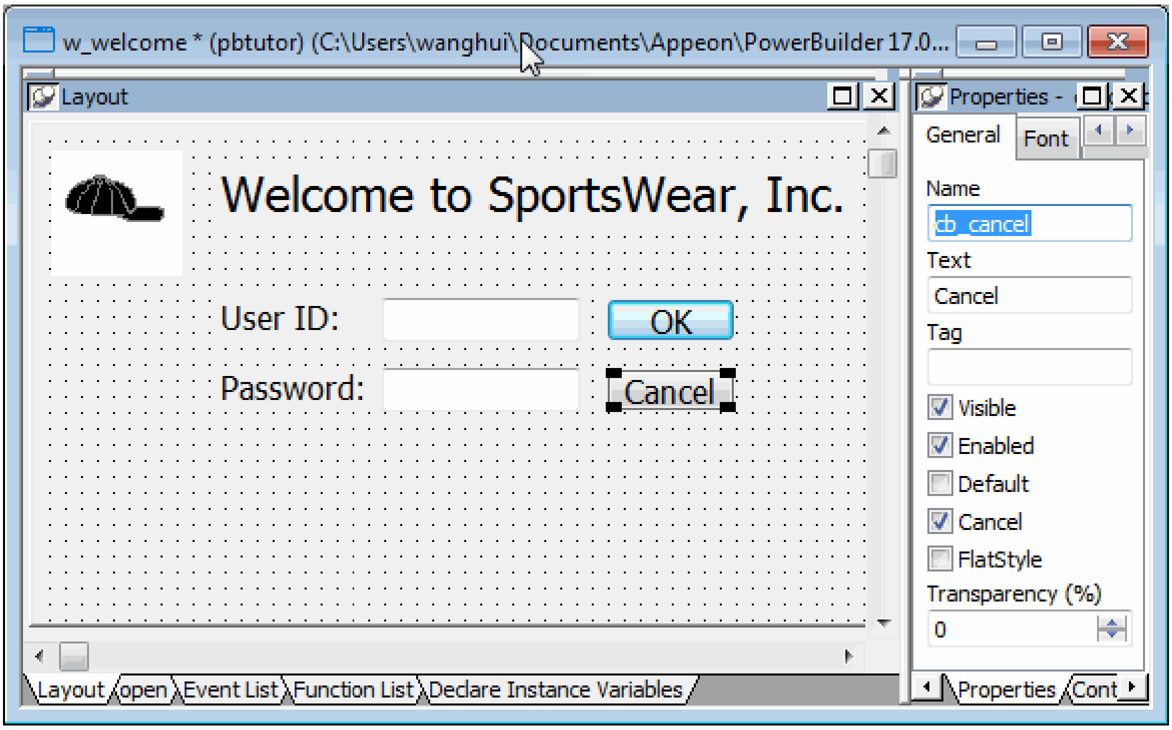
3. Select the bottom CommandButton control.

   Type cb_cancel in the Name text box.

   Type Cancel in the Text text box.

   Select the Cancel check box.

   Because you selected the Cancel check box, when an application user presses the Esc key, the Clicked event for this button will be triggered.



w_welcome * (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0...

Name: cb_cancel

Text: Cancel

Tag:

☑ Visible

☑ Enabled

□ Default

☑ Cancel

□ FlatStyle

Transparency (%): 0

2.3.3 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

Code some Help events and 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 change the tab order for the window you created.

1. Select Format>Tab Order from the menu bar.

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

2. Click the tab order number for the sle_userid control.

   Type 10 if the tab order value for this control is not already 10.

   You can type a new tab order number only when the old number turns blue against a red background.

3. Make changes as needed so that the other controls have these values:

Table 2.2:

Click this control	Control name	Has this value
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

4. Select Format>Tab Order from the menu bar.

   This is a toggle switch. Selecting this menu item a second time saves your changes and clears the tab order numbers from the login window. You can also use the Tab Order button on the PainterBar.

2.3.4 Code some Help events and preview the window

Where you are

Create a new window

Add controls to the window

Change the tab order on the window

> Code some Help events and preview the window

Write the script to open the window

Now you use the Script view to add context-sensitive Help messages to the SingleLineEdit controls that you placed on the login window, and you preview the window.

Using the Script view

The Script view has three drop-down list boxes.

The first drop-down list box displays the list of available controls for the current object plus two special entries, Functions and Declare. The contents of the second drop-down list box depend on the selection in the first drop-down list box. The third drop-down list box contains all ancestor objects of the current object, if any.

Table 2.3:

Selection in first drop-down list box	Contents of second drop-down list box	Contents of third drop-down list box
An object or control name	List of events for the selected object or control.	All of the ancestor objects of the current object, if any
Functions	List of editable functions. Displays (New Function) if no editable functions exist.	All ancestor objects with functions having the same signature as selected function

Selection in first drop-down list box	Contents of second drop-down list box	Contents of third drop-down list box
Declare	List of declaration types: global, shared, and instance variables, and global and local external functions.	Empty

If the Script view is not currently displayed in your Window painter, you can open it by double-clicking an object in the Layout view.

1. Double-click the top SingleLineEdit control in the Layout view.

   The object name, sle_userid, appears in the first drop-down list box in the Script view.

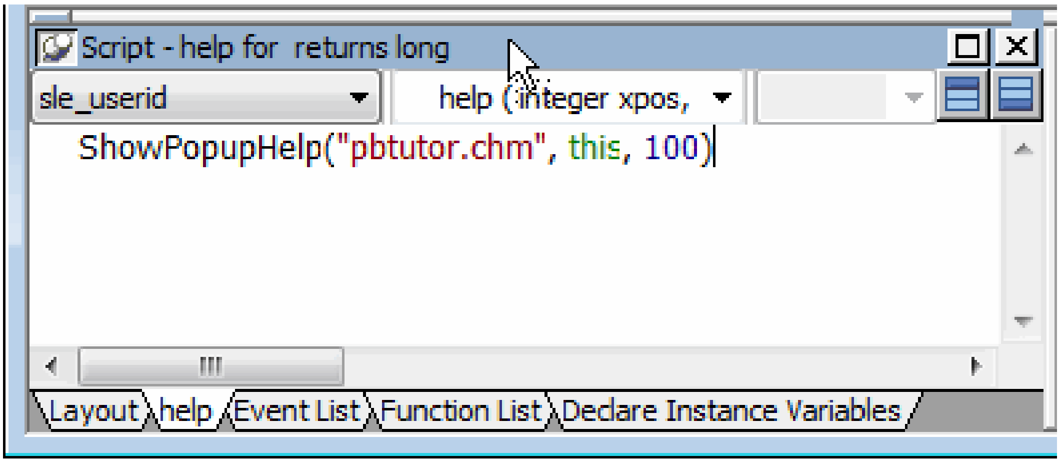
2. Select the Help event in the second drop-down list box in the Script view.

   The Help event has the prototype: help (integer xpos, integer ypos) returns long [pbm_help]

3. Type the following line of PowerScript code in the script area:

   ```
   ShowPopupHelp("pbtutor.chm", this, 100)
   ```

   You can find the pbtutor.chm file in the Tutorial directory. The second argument refers to the current SingleLineEdit control, and the last argument refers to a context ID in the pbtutor.chm file.



```
ShowPopupHelp("pbtutor.chm", this, 100)
```

   As you type text in the Script view, notice that PowerBuilder changes the text colors to show what kind of syntax element you have entered (such as keywords, variables, and comments).

4. Select sle_password from the first drop-down list box in the Script view.

   Select the Help event in the second drop-down list box.

   PowerBuilder compiles the code you typed for the Help event of the sle_userid SingleLineEdit text box. You now add a Help event for the sle_password SingleLineEdit text box.

5. Type the following lines in the script area:

```
ShowPopupHelp("pbtutor.chm", this, 200)
```

6. Select File>Run/Preview from the menu bar

   or

   Click the Run/Preview Object in the PowerBar.

   The Run dialog box displays. Be sure the Objects of Type list box displays Windows and the w_welcome object is selected.

Previewing the window

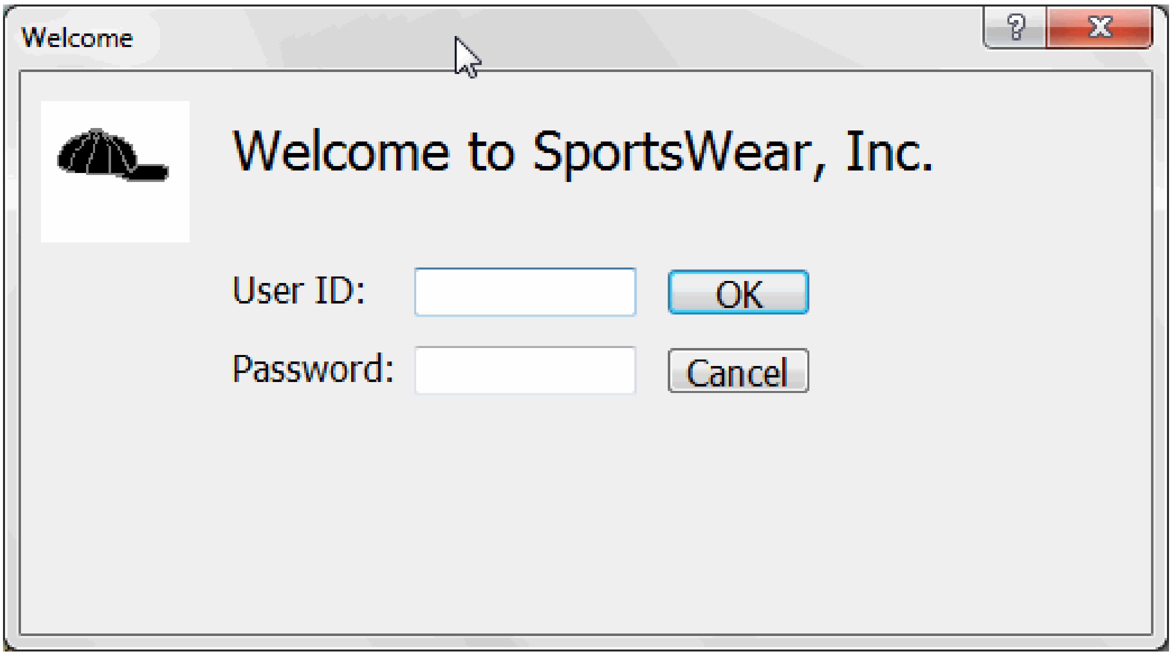
You can preview the window without running scripts by selecting Design>Preview on the menu bar or the Preview button in PainterBar1 (which uses the same icon as the Run/Preview Object button in the PowerBar). However, you must run scripts to view the results of the Help event scripts you just entered.

7. Click OK.

   A message box prompts you to save your changes.

8. Click Yes.

   The login window appears as it would at runtime. If you do not like the window layout, you can change the size, location, and fonts of the window controls when you go back to the Window painter workspace.



9. Click the question-mark button in the login window title bar.

   Click inside the sle_userid SingleLineEdit text box.

   A message displays: Type your user ID here. This text is associated with context ID 100 in the pbtutor.chm file. You entered the context ID as an argument of the ShowPopupHelp call for the sle_userid Help event.

10.Click anywhere in the window to close the message.

Click inside the sle_password SingleLineEdit text box.

Press F1.

A message displays: Type your password here. This text is associated with context ID 200 in the pbtutor.chm file.

11.Click anywhere in the window to close the message.

Click the close button in the login window title bar.

You return to the Window painter workspace.

Later you add code to the Clicked event for the Cancel button that closes the application.

12.Close the Window painter by clicking the close button in PainterBar 1.

The close button is labeled with an x.

2.3.5 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

Code some Help events and preview the window

> Write the script to open the window

Now you add a one-line script to open the login window as soon as the application starts executing. Although you could change the script in the Application object Open event to open the login window, in this tutorial you make the call to open the login window from the Open event of the MDI frame window. This window (w_pbtutor_frame) was created by the Template Application wizard.

In wizard-generated script, the frame window is called by the Application object Open event. When you add a call to open the login window from the MDI frame window Open event, the login window still displays before the frame window. Because the login window is a response window, it temporarily blocks execution of the remainder of the frame window Open event script.

In this exercise you:

- Modify the frame window Open event
- Compile the script

2.3.5.1 Modify the frame window Open event

Now you modify the frame window Open event script to open the login window.

Wizard-generated script

The frame window Open event already has a script associated with it that was generated by the Template Application wizard. The script creates an instance of a wizard-generated user object that serves as a sheet (window) manager. The Open script then calls the ue_postopen event.

The ue_postopen event registers sheet windows with the sheet manager. The event is posted to the end of the messaging queue for the window and is not processed until the remainder of the Open script (that you add in this lesson) is executed.

1. Select File>Open from the menu bar.

   Make or verify the following selections in the Open dialog box:

   Table 2.4:

Open dialog box item	Selection to make (or verify)
Target	pbtutor -- currently your only target
Libraries	pbtutor.pbl -- currently your only library; if you cannot see the full library name, you can drag the edge of the dialog box to increase its size
Object Name	w_pbtutor_frame
Objects of Type	Windows -- you must make sure this is selected before you can select a named object of this type
Using	Painter

   The w_pbtutor_frame object is the main application window created by the Template Application wizard.

2. Click OK.

   The Window painter opens and displays views of the main application window in the painter workspace.

   If the Script view is not open, you can open it by selecting View>Script in the workspace menu bar or by double-clicking inside the Layout view.

3. Move the cursor to the top of the Script view.

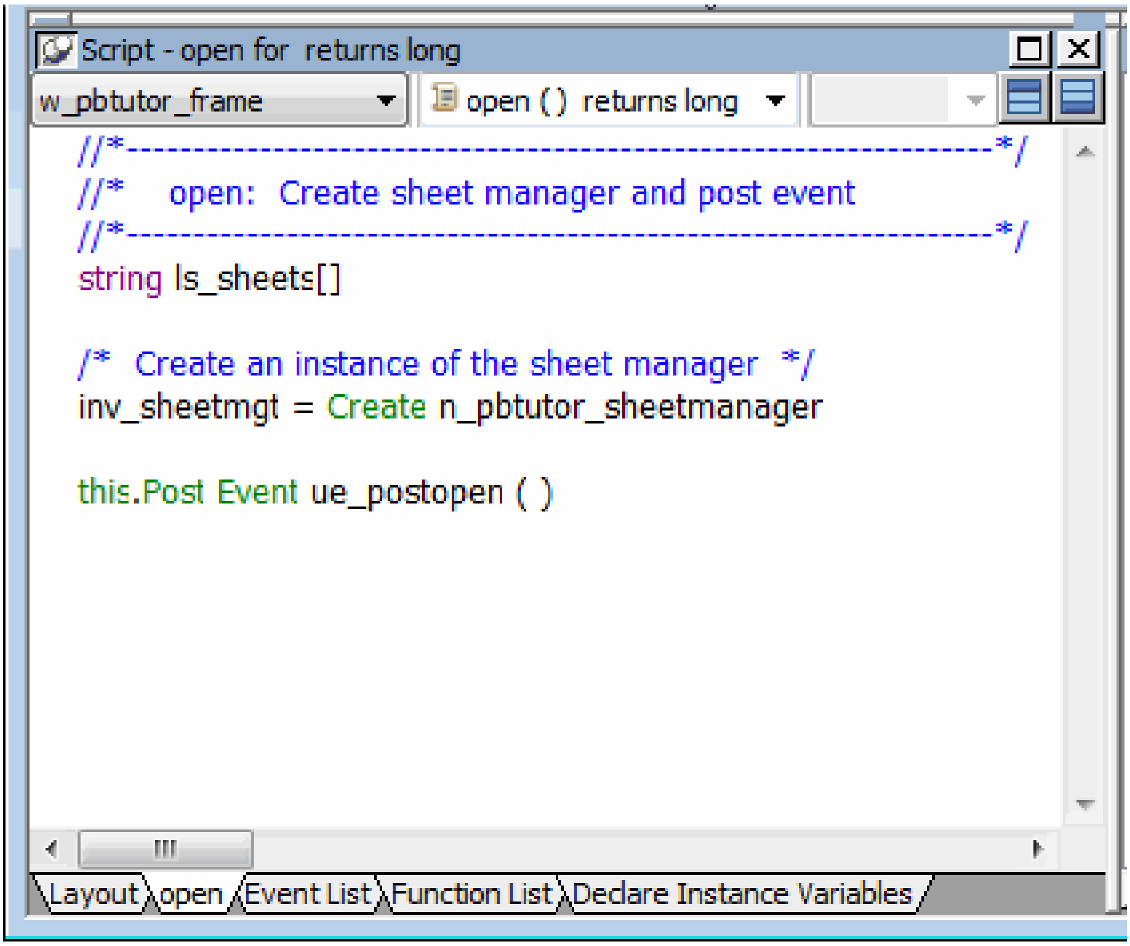
   The view title bar rolls down. It contains a pushpin button on the left and a maximize/ minimize button and a close button on the right.

4. Click the pushpin in the title bar of the Script view.

   The title bar of the Script view reads:

   ```
   Script - open for w_pbtutor_frame returns long
   ```

5. Make sure the Open event displays in the Script view.



```
//*-------------------------------------------------------------------*/
//*    open:  Create sheet manager and post event
//*-------------------------------------------------------------------*/
string ls_sheets[]

/*  Create an instance of the sheet manager  */
inv_sheetmgt = Create n_pbtutor_sheetmanager

this.Post Event ue_postopen ( )
```

The PowerScript code in the Script view is preceded by comments.

Using comments

A comment in PowerScript 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. Comments are not parsed by the script compiler.

You now modify the Open event script to cause the login window to display.

6. Click after the parentheses in the line that reads:

```
this.Post Event ue_postopen ( )
```

The ue_postopen event does not take any arguments.

7. Press Enter twice.

This adds a blank line in the Script view for legibility. The cursor moves to a new line following the blank line.

8. Type Open the login window on the new line in the Script view.

9. Click the Comment button (Comment) in PainterBar2.

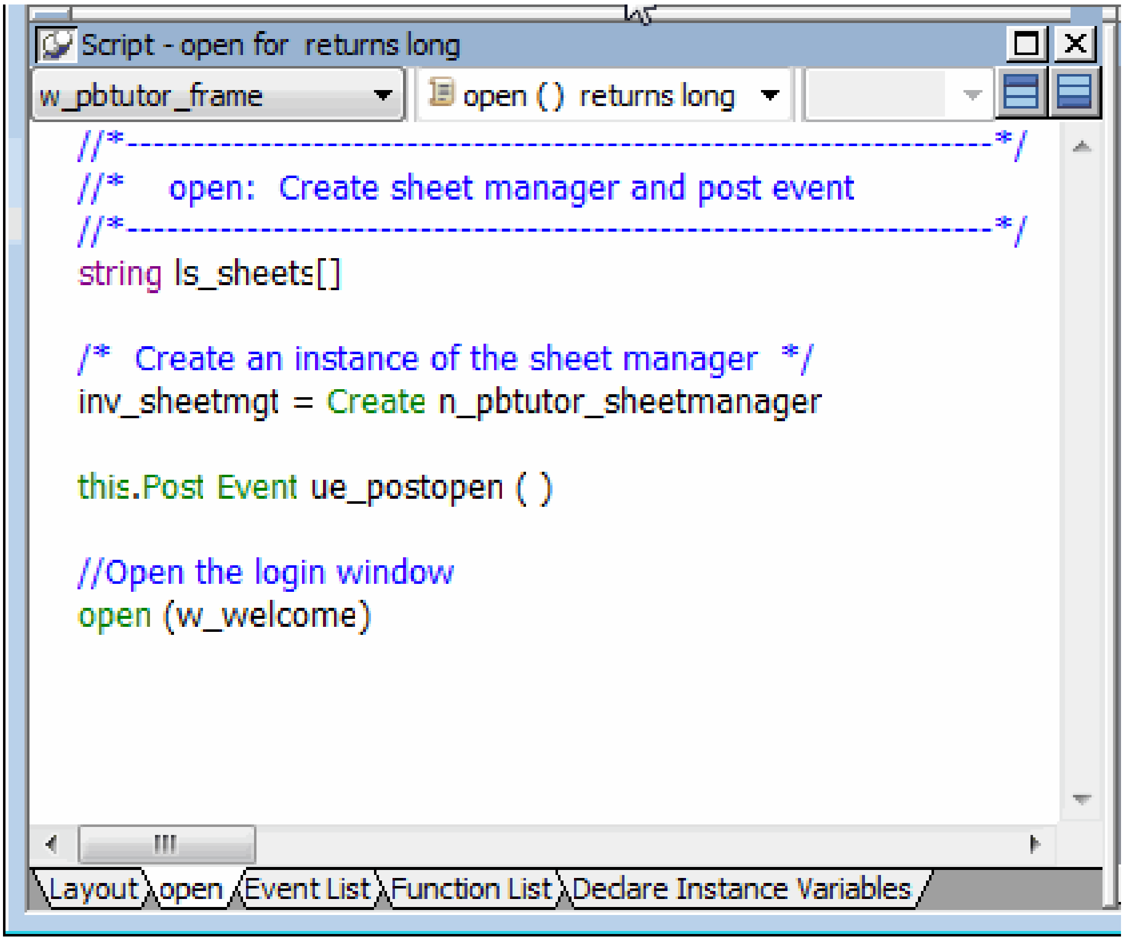
   Two slashes appear in front of the line you typed -- it is changed into a comment.

10. Move the cursor to the end of the comment.

   Press Enter to add a new line.

   Type open (w_welcome) on the new line.

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



```
//*-----------------------------------------------------------*/
//*    open:  Create sheet manager and post event
//*-----------------------------------------------------------*/
string ls_sheets[]

/*  Create an instance of the sheet manager  */
inv_sheetmgt = Create n_pbtutor_sheetmanager

this.Post Event ue_postopen ( )

//Open the login window
open (w_welcome)
```

Accessing context-sensitive Help

To access context-sensitive Help for a function or reserved word (such as Open), select the function or reserved word in the Script view and press Shift+F1. You can always open the main Help screen by pressing F1.

2.3.5.2 Compile the script

Now you compile the script you just typed. In this exercise, you use a pop-up menu item to compile the script. PowerBuilder also compiles a script when you close the Script view or when you select a different object, event, or function for display in the Script view.

Handling errors in scripts

When there is an error in a script, an error window displays at the bottom of the Script view 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, you can try to compile the script again.

Commenting out errors

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

1. Right-click anywhere in the Script view script area.

   Select Compile from the pop-up menu.

   The script compiles. You do not leave the Script view or the Window painter workspace.

2. Select File>Save from the menu bar.

   Select File>Close from the menu bar.

   The Window painter closes.

2.4 LESSON 4 Connecting to the Database

This lesson shows you how to make the application connect to the Enterprise Application Sample demonstration database (Demo Database) at execution time and how to use the Database painter to look at the table definitions and database profile for this database.

In this lesson you:

- Look at the Demo Database
- Run the Connection Object wizard
- Declare a global variable
- Modify the connection information
- Complete the login and logout scripts
- Run the application

How long does it take?

About 30 minutes.

2.4.1 Look at the Demo Database

Where you are

> Look at the Demo Database

Run the Connection Object wizard

Declare a global variable

Modify the connection information

Complete the login and logout scripts

Run the application

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

Defining a data source

Using the ODBC administrator or other database connection utilities, you can define a database as a data source for your application. You can access the ODBC Administrator from the DataBase Profiles dialog box. The definitions of ODBC data sources are stored in the odbc.ini registry key.

Using database profiles to connect

Once you define a data source, you can create a database profile for it. 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. When you are developing an application, you can change database profiles to connect to a different data source.

When database connections occur

PowerBuilder can establish a connection to the database in either the design-time or runtime environment. PowerBuilder connects to a database when you open certain painters, when you compile or save a PowerBuilder script that contains embedded SQL statements, or when you run a PowerBuilder application that accesses the database.

To maintain database definitions with PowerBuilder, you do most of your work using the Database painter. The Database painter allows 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

In this exercise you:

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

2.4.1.1 Look at the database profile for the Demo Database

If you installed PowerBuilder with standard options, you already have a data source and a database profile defined for the Demo Database. You use the Demo Database in this tutorial.

Demo Database is an SQL Anywhere database that is accessed through ODBC. In this lesson you look at the database profile for the Demo Database. PowerBuilder stores database profile parameters in the registry.

1. Click the Database Profile button (DB Prof) in the PowerBar

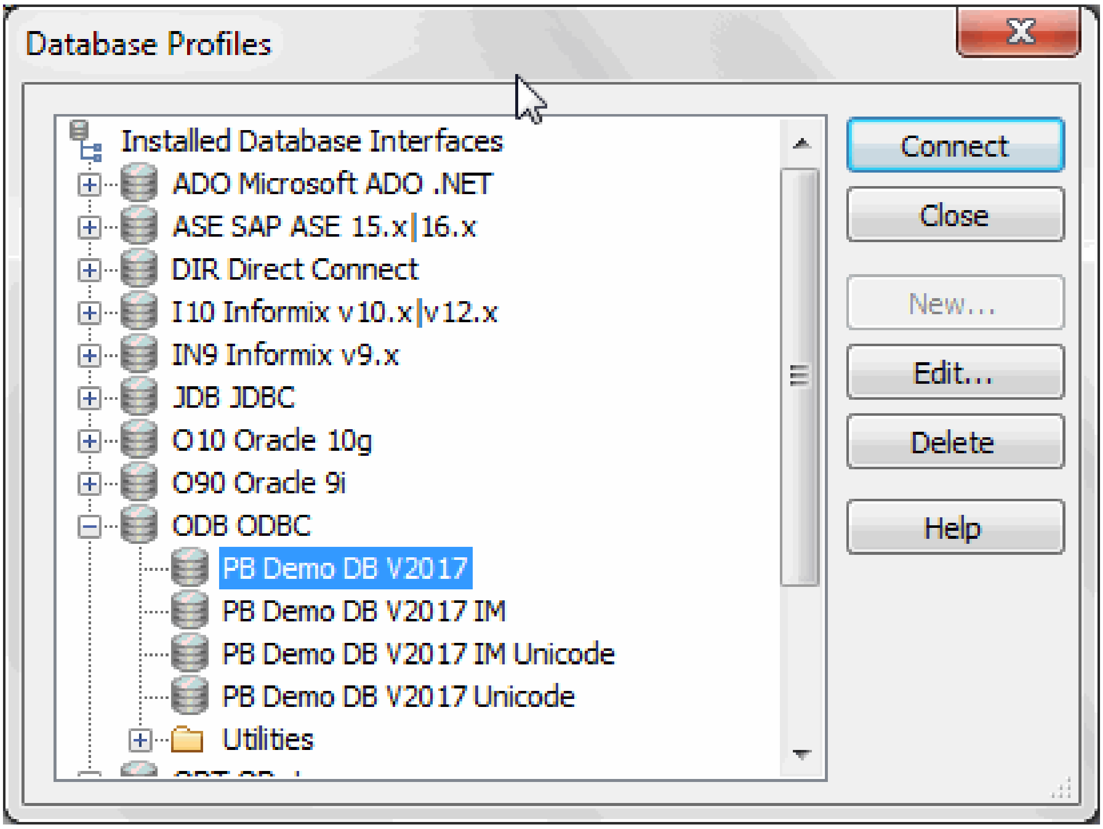
   or

   Select Tools>Database Profile 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. You can click the + signs or double-click the icons next to items in the tree view to expand or contract tree view nodes.

2. Expand the ODB ODBC node by clicking on the plus sign, and select PB Demo DB V2019R3.

   PowerBuilder created this profile during installation.



If you do not see the PB Demo DB V2019R3 database profile

If there is no profile for the PB Demo DB V2019R3 database, you may not have installed the database. You can install it now from the product installation package.

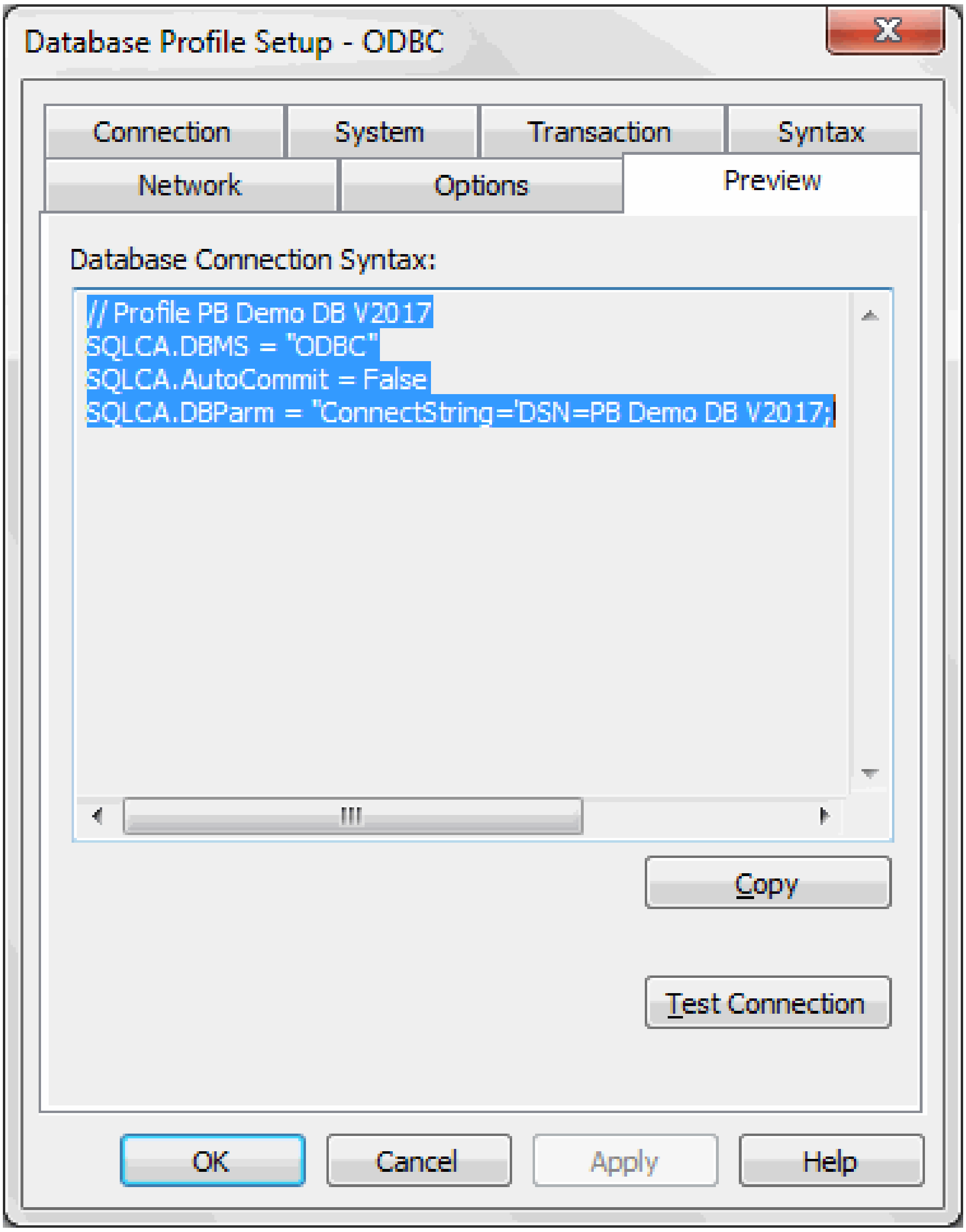
If you did install the database and it is defined as a data source in the ODBC Administrator, select ODBC in the tree view of the Database Profile painter and click New. In the Database Profile Setup dialog box, select the data source from the Data Source drop-down list and type PB Demo DB V2019R3 in the Profile Name text box. Type dba for the user ID and sql for the password, then click OK to return to the painter.

3. Click Edit.

   PowerBuilder displays the Connection page of the Database Profile Setup dialog box.

4. Select the Preview tab.

   The PowerScript connection syntax for the selected profile is shown on the Preview tab. If you change the profile connection options, the syntax changes accordingly.



```
// Profile PB Demo DB V2017
SQLCA.DBMS = "ODBC"
SQLCA.AutoCommit = False
SQLCA.DBParm = "ConnectString='DSN=PB Demo DB V2017;
```

5. Click the Test Connection button.

   A message box tells you that the connection is successful.

   If the message box tells you the connection is not successful

   Close the message box and verify that the information on the Connection page of the Database Profile Setup dialog box is correct. Then check the configuration of the data source in the ODBC Administrator. You can run the ODBC Administrator by expanding the Utilities folder under the ODB ODBC node of the Database Profile painter and double-clicking the ODBC Administrator item.

6. Click OK to close the message box.

   Click Cancel to close the Database Profile Setup dialog box.

   Click Close to close the Database Profiles dialog box.

2.4.1.2 Look at table definitions in the Demo Database

Now you look at the definitions for the Customer and Product tables in the Demo Database. This helps you become familiar with the Database painter and the tables you will use in the tutorial.

What happens when you connect

To look at the table definitions, you have to connect to the database. When you connect to a database in the development environment, PowerBuilder writes the connection parameters to the Windows registry.

Each time you connect to a different database, PowerBuilder overwrites the existing parameters in the registry with those for the new database connection. When you open a PowerBuilder painter that accesses the database, you automatically connect to the last database used. PowerBuilder determines which database this is by reading the registry.

1. Click the Database button (Database) in the PowerBar.

   PowerBuilder connects to the database and the Database painter opens. The Database painter title bar identifies the active database connection.

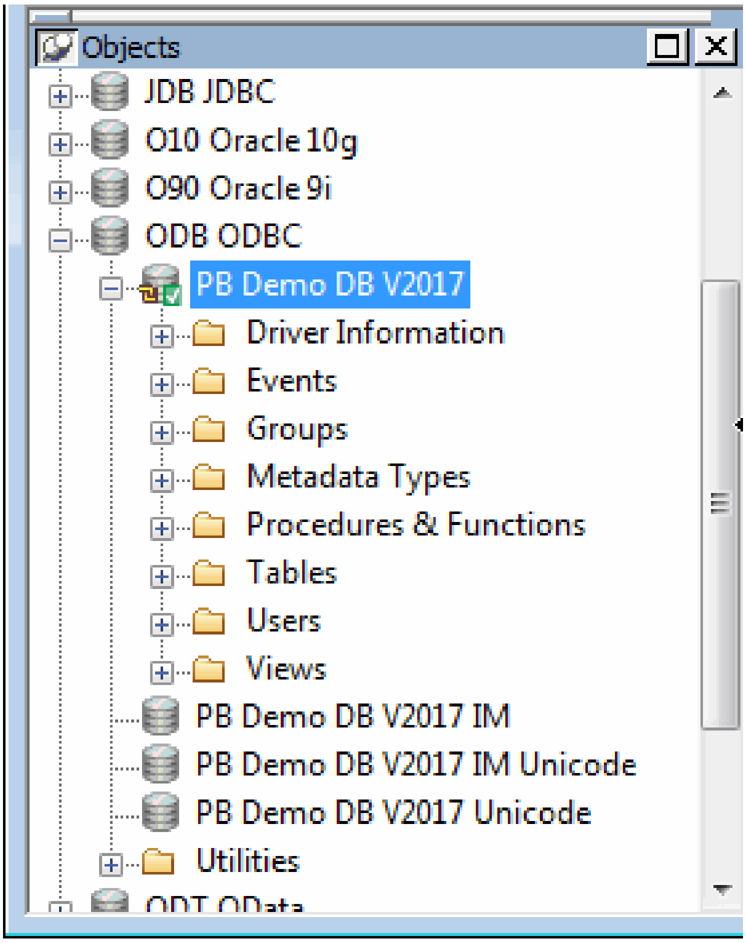
   The Objects view of the Database painter displays all existing database profiles in a tree view under the Installed Database Interfaces heading. The PB Demo DB V2019R3 database is visible under the ODB ODBC node in the tree view.

   If the Objects view is not open

   The Objects view is part of the default view layout scheme. To reset to this scheme, select View>Layouts>Default. You can also open an Objects view by selecting View>Objects from the menu bar.

2. Expand the PB Demo DB V2019R3 database node in the Objects view.

   Notice the folders under the PB Demo DB V2019R3 database node.



3. Expand the Tables folder.

   You see the list of tables in the database.

Table names might have a prefix

The table names in the Select Tables dialog box might have a prefix such as dba or dbo. This depends on the login ID you are using. You can ignore the prefix.

4. Right-click the customer table and select Add To Layout from the pop-up menu

   or

   Drag the customer table from the Objects view to the Object Layout view.

Dragging an object from one view to another

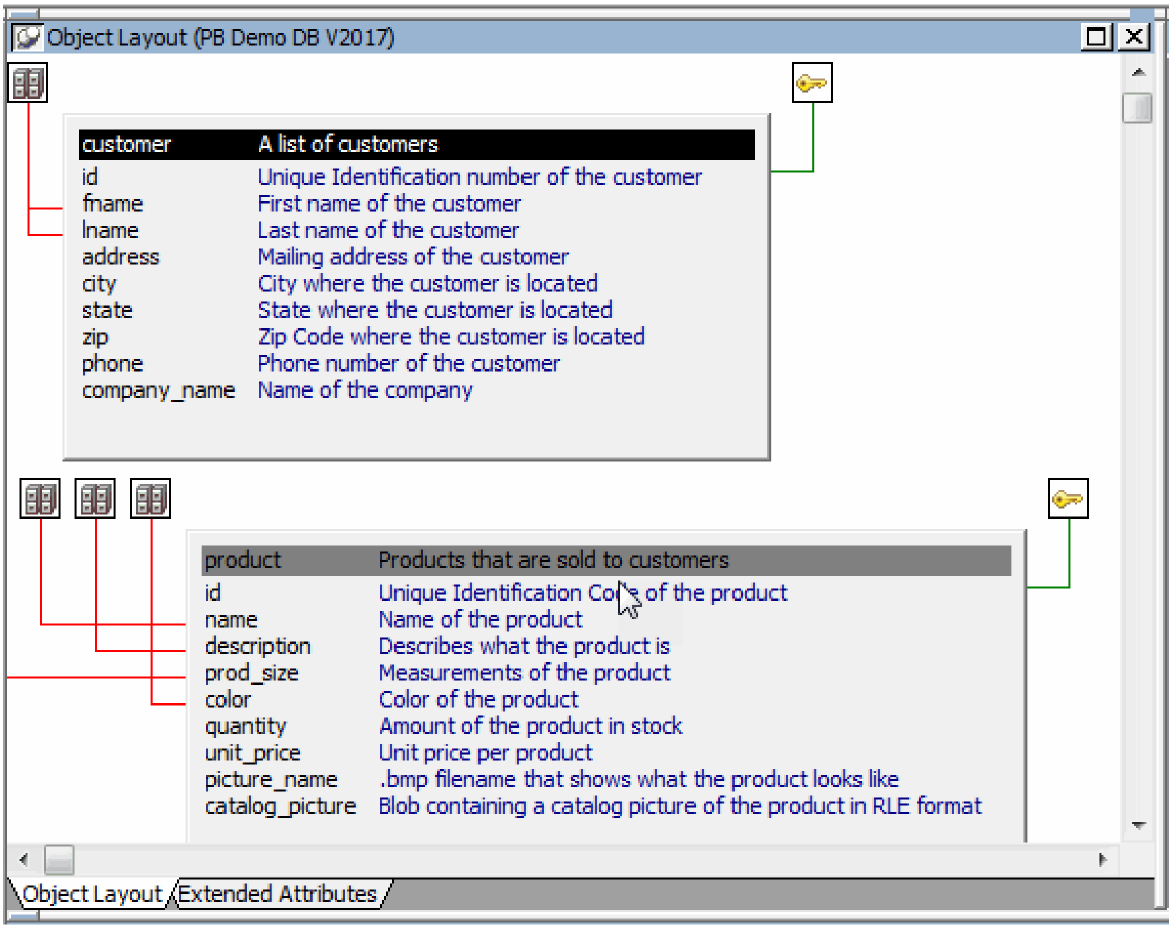
When you start dragging an object from the Objects view to another view, the pointer changes to a barred circle. If you continue moving the cursor to a view that can accept the object, the barred circle changes back to a pointer with an additional arrow symbol in a small box. When you see this symbol, you can release the object.

5. Repeat step 4 for the product table.

Widening the Object Layout view

You can widen the Object Layout view by dragging its separator bars toward the painter frame. If the Object Layout view is part of a stack, you might find it easier to separate it from the stack before you change its size.

The Object Layout view shows the two tables you selected.



customer	A list of customers
id	Unique Identification number of the customer
fname	First name of the customer
lname	Last name of the customer
address	Mailing address of the customer
city	City where the customer is located
state	State where the customer is located
zip	Zip Code where the customer is located
phone	Phone number of the customer
company_name	Name of the company

product	Products that are sold to customers
id	Unique Identification Code of the product
name	Name of the product
description	Describes what the product is
prod_size	Measurements of the product
color	Color of the product
quantity	Amount of the product in stock
unit_price	Unit price per product
picture_name	.bmp filename that shows what the product looks like
catalog_picture	Blob containing a catalog picture of the product in RLE format

Viewing table data types, comments, keys, and indexes

In the Object Layout view, you can see a description for each column, as well as icons for keys and indexes. If you do not see this, right-click a blank area inside the view and select Show Comments, Show Referential Integrity, and then Show Index Keys from the pop-up menu. If you select Show Datatypes, you also see the data type for each column in the selected tables.

6. Right-click the title bar of the customer table in the Object Layout view and select Alter Table from the pop-up menu

   or

   Right-click the customer table in the Objects tree view and select Alter Table from the pop-up menu.

The Columns view displays the column definitions for the table.

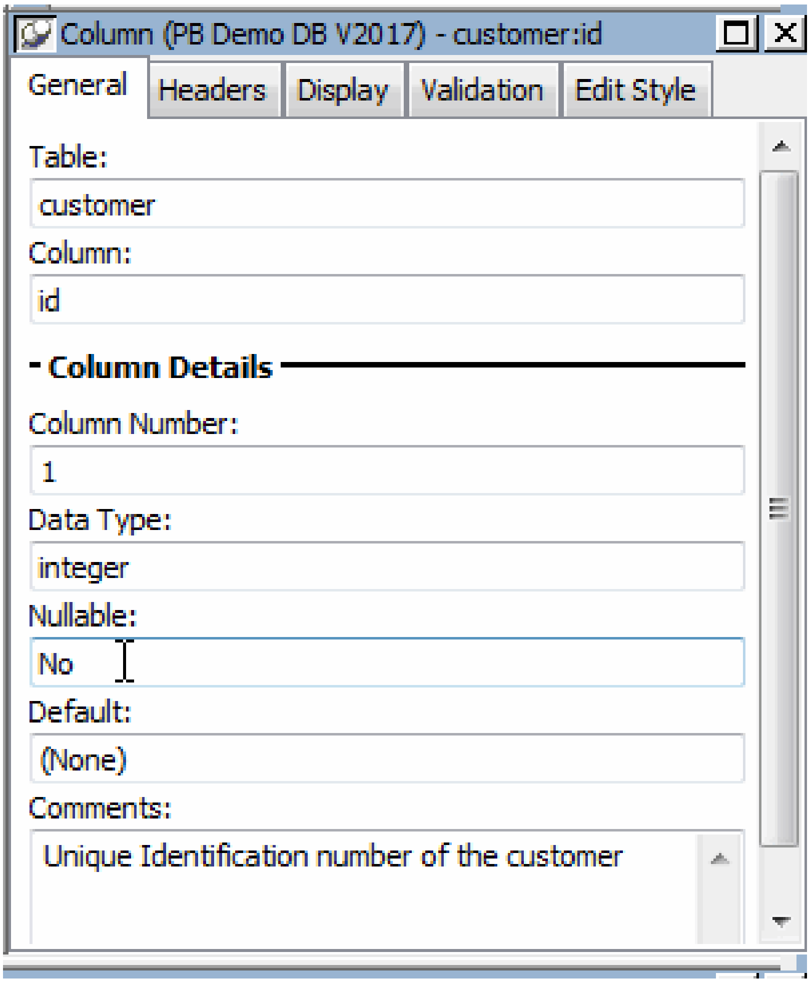
7. Right-click a column in the customer table in the Object Layout view.

   Select Properties from the pop-up menu.

   In the Database painter, the Properties view is also called the Object Details view.

   The title bar and tab headings for the Object Details view change dynamically depending on the current object selection. The title bar gives the object type, the database connection, and the object identifier.

   The Object Details view for a column has five tabs, one for general database properties, one for column header information, and the others for column extended attributes.



Column (PB Demo DB V2017) - customer:id

Table: customer

Column: id

Column Details

Column Number: 1

Data Type: integer

Nullable: No

Default: (None)

Comments: Unique Identification number of the customer

About extended attributes

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

You can define new extended attributes or change the definitions of existing extended attributes from the pop-up menus of items in the Extended Attributes view of the Database painter.

8. Close the Database painter.

2.4.2 Run the Connection Object wizard

Where you are

Look at the Demo Database

> Run the Connection Object wizard

Declare a global variable

Modify the connection information

Complete the login and logout scripts

Run the application

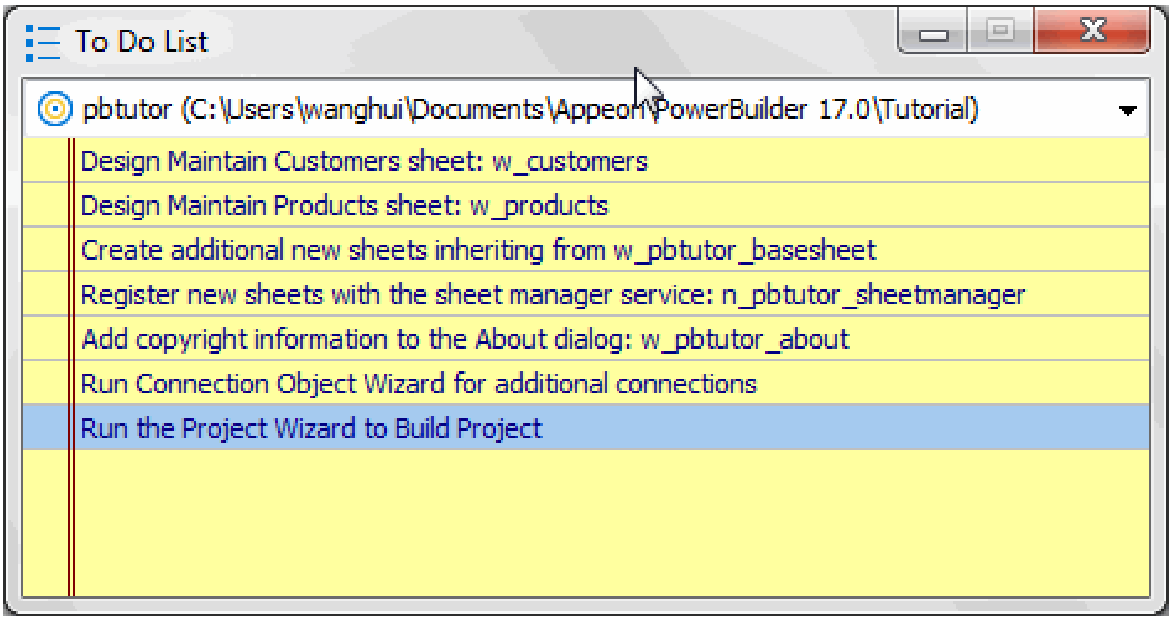
Now you run the Connection Object wizard to create a connection service manager, which you use to establish the runtime database connection.

The connection service manager is a nonvisual user object. It is nonvisual because it does not have a graphic representation in the runtime application; it is a user object because it is a customized object. You use it to perform database connection processing in a PowerBuilder application.

> **Why you run a second wizard**
>
> If you had specified connection information in the Template Application wizard, you would have created the connection service manager when you generated the application. You can use multiple wizards in building your application.

1. Click the To-Do List button (To-Do List) in the PowerBar.

   The To-Do List was generated by the Template Application wizard.



pbtutor (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial)

Design Maintain Customers sheet: w_customers
Design Maintain Products sheet: w_products
Create additional new sheets inheriting from w_pbtutor_basesheet
Register new sheets with the sheet manager service: n_pbtutor_sheetmanager
Add copyright information to the About dialog: w_pbtutor_about
Run Connection Object Wizard for additional connections
Run the Project Wizard to Build Project

2. Double-click the Run Connection Object Wizard item in the list

or

Right-click the Run Connection Object Wizard item.

Select Go To Link from the pop-up menu.

This is the next-to-last item in the list. The To-Do List lists what you need to do to complete the application. You can also use the list to make comments to yourself or other developers working on the application.

You can also run the Connection Object wizard from the PB Object page of the New dialog box. You used the New dialog box to run the Template Application wizard in Starting PowerBuilder

The first page of the wizard tells you what it can do.

3. Click Next until the Choose Database Profile page displays.

   You accept the wizard's default selections for the destination library (pbtutor.pbl) and the database connectivity options (SQL). The Choose Database Connection Profile page lists all the database profiles stored in the registry.

4. Choose the PB Demo DB V2019R3 in the Database Profiles list box if it is not already selected.

5. Click Next until the Ready To Create Connection Object page displays.

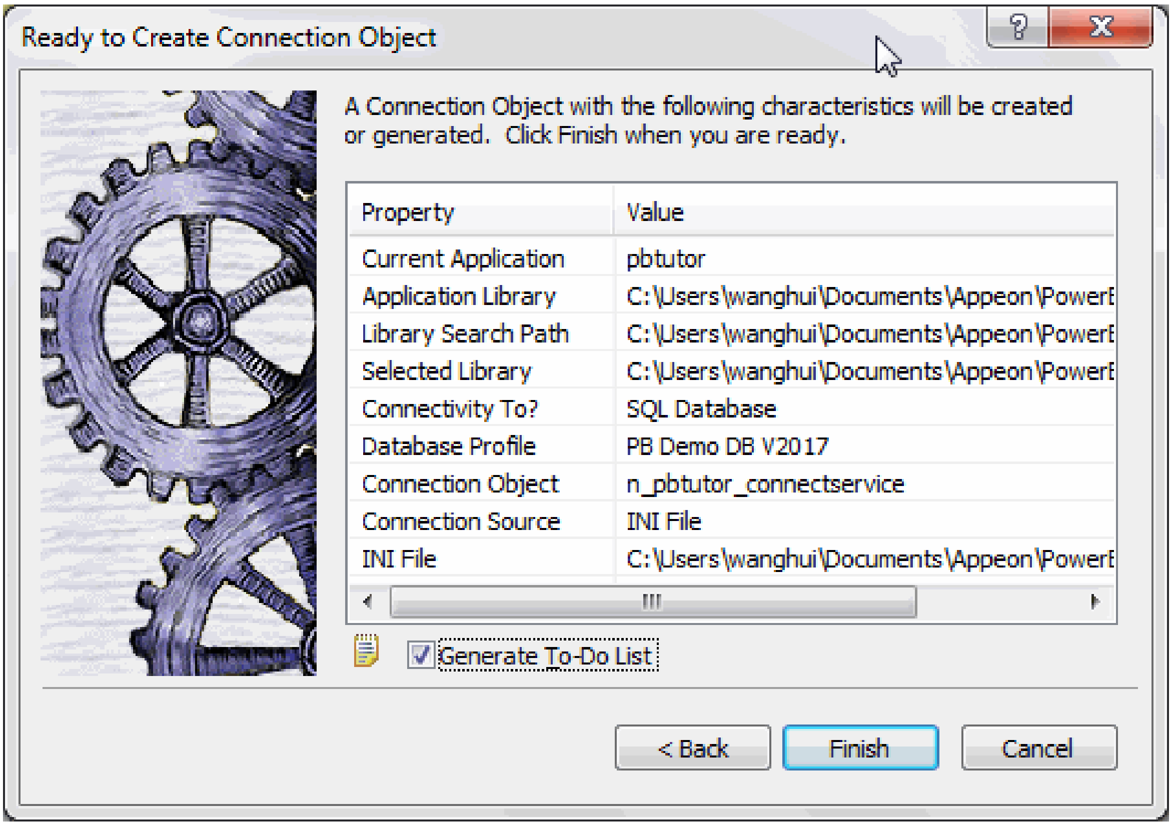
   You accept the default settings for the following items:

Table 2.5:

Wizard page	Option	Default selection
Specify Connectivity Source Info	Source of Connection Information	Application INI File
	Connection Service Object	n_pbtutor_connectservice
Name Application INI File	Application INI File	pbtutor.ini

The wizard creates the n_pbtutor_connectservice user object to manage your database connections. If you change an instance variable in this connection service object, you can change the source of connection information to the registry or to a script file. Otherwise, the pbtutor.ini file -- created by the wizard -- is used for application connection information.

The last wizard page contains a summary of your choices, including the default selections.



Ready to Create Connection Object

A Connection Object with the following characteristics will be created or generated. Click Finish when you are ready.

Property	Value
Current Application	pbtutor
Application Library	C:\Users\wanghui\Documents\Appeon\PowerE
Library Search Path	C:\Users\wanghui\Documents\Appeon\PowerE
Selected Library	C:\Users\wanghui\Documents\Appeon\PowerE
Connectivity To?	SQL Database
Database Profile	PB Demo DB V2017
Connection Object	n_pbtutor_connectservice
Connection Source	INI File
INI File	C:\Users\wanghui\Documents\Appeon\PowerE

☑ Generate To-Do List

6. Click Finish.

   The wizard creates the connection service object and opens it in the User Object painter. You can see n_pbtutor_connectservice in the System Tree. The wizard also creates the application INI file. The To-Do List is still open.

7. Close the To-Do List.

2.4.3 Declare a global variable

Where you are

Look at the Demo Database

Run the Connection Object wizard

> Declare a global variable

Modify the connection information

Complete the login and logout scripts

Run the application

You will next examine the new connection service manager and create a global variable to reference it. A global variable is available to all objects in the application.

In more complex applications, you might prefer to reference the connection service manager with local variables. This would release more memory as soon as the local variable went out of scope. But in the tutorial, you should keep an instance of the connection service manager available as long as the database connection is open.

Establishing a connection

To make it possible for an application to connect to the database at execution time, the connection service manager calls a wizard-generated function to set properties for a Transaction object that serves as a communications area between the application and the database.

SQLCA Transaction object

The connection service manager uses a built-in nonvisual system object, the SQL Communications Area (SQLCA) object, as the default Transaction object. The SQLCA object has several default properties (including database name, login ID, and password) that are populated by the connection service manager.

If an application communicates with multiple databases, you can create additional Transaction objects as needed, one for each database connection.

> **What is required and what is not**
>
> You must have a Transaction object to connect to a database. The connection service manager is not required, but is used in the tutorial because it generates Transaction object properties you would otherwise have to type in an application script.

1. Make sure n_pbtutor_connectservice is open in the User Object painter.

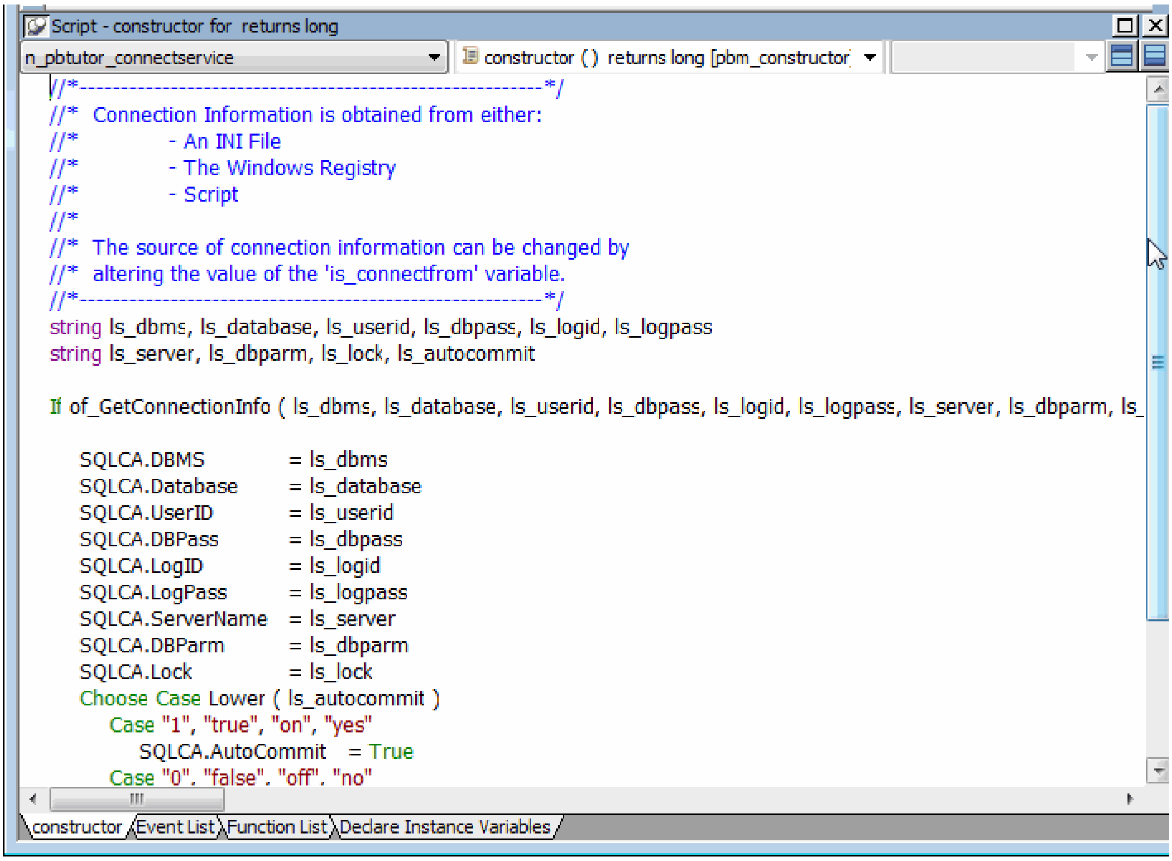
   > **Opening the connection service manager**
   >
   > If the n_pbtutor_connectservice object is not open in the User Object painter, double-click n_pbtutor_connectservice in the System Tree.

   The default view layout scheme for the User Object painter includes a Script view and a Declare Instance Variables view as part of a stack of tabbed panes.

2. Make sure n_pbtutor_connectservice is selected in the first drop-down list box of the Script view.

   Make sure the Constructor event is selected in the second drop-down list box.

   The Script view displays the script created by the Connection Object wizard for the Constructor event.



Script - constructor for returns long
n_pbtutor_connectservice
constructor ( ) returns long [pbm_constructor]
constructor
Event List
Function List
Declare Instance Variables

```
//*-----------------------------------------------------*/
//*  Connection Information is obtained from either:
//*              - An INI File
//*              - The Windows Registry
//*              - Script
//*
//*  The source of connection information can be changed by
//*  altering the value of the 'is_connectfrom' variable.
//*-----------------------------------------------------*/
string ls_dbms, ls_database, ls_userid, ls_dbpass, ls_logid, ls_logpass
string ls_server, ls_dbparm, ls_lock, ls_autocommit

If of_GetConnectionInfo ( ls_dbms, ls_database, ls_userid, ls_dbpass, ls_logid, ls_logpass, ls_server, ls_dbparm, ls_

    SQLCA.DBMS          = ls_dbms
    SQLCA.Database      = ls_database
    SQLCA.UserID        = ls_userid
    SQLCA.DBPass        = ls_dbpass
    SQLCA.LogID         = ls_logid
    SQLCA.LogPass       = ls_logpass
    SQLCA.ServerName    = ls_server
    SQLCA.DBParm        = ls_dbparm
    SQLCA.Lock          = ls_lock
    Choose Case Lower ( ls_autocommit )
        Case "1", "true", "on", "yes"
            SQLCA.AutoCommit   = True
        Case "0", "false", "off", "no"
```

The script calls the function of_GetConnectionInfo to obtain connection information. You will next look at the script for this function.

3. Select Functions in the first drop-down list box in a Script view.

4. Select of_GetConnectionInfo in the second drop-down list box.

   The script for this function passes database connection information to the Constructor event of the connection service manager. The information passed depends on an instance variable. In this case, the value of the is_connectfrom variable is 1. You will verify this in a moment. The instance variable is available to all functions and events of the n_pbtutor_connectservice object.

   Because the is_connectfrom variable is 1, the connection service manager looks to the Database section of the named INI file to get database connection information using ProfileString function calls. In this case, the named INI file is pbtutor.ini. You created this file with the Connection Object wizard.

   Later you modify the pbtutor.ini file and the of_GetConnectionInfo function to make sure that user ID and password information comes from the login window instead of the INI file.

5. Select of_ConnectDB in the second drop-down list box.

   This is the connection service manager function that actually connects to the database using the SQLCA Transaction object. You call this function from the login window you created in Building a Login Window

Notice that the wizard-generated script for this function also opens a message box if the database connection fails.

6. Select of_DisconnectDB in the second drop-down list box.

   This is the connection service manager function that disconnects from the database. You call this function from the application Close event.

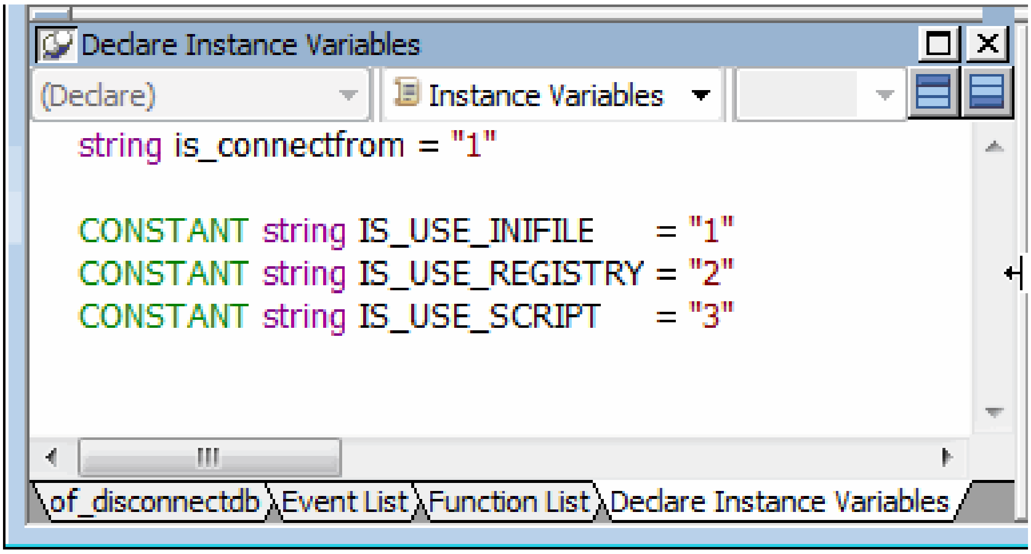
7. Click the Declare Instance Variables tab.

   Make sure Instance Variables is selected in the second drop-down list box.

Selecting Declare in Script views

The Declare Instance Variables view is a special instance of the Script view. It displays when you select Declare in the first drop-down list box of the Script view. However, you cannot select Declare if a second Script view already displays instance variables.

You can now verify that the value of the is_connectfrom variable is 1.



```
string is_connectfrom = "1"

CONSTANT string IS_USE_INIFILE   = "1"
CONSTANT string IS_USE_REGISTRY = "2"
CONSTANT string IS_USE_SCRIPT    = "3"
```

8. Select Global Variables in the second drop-down list box.

   Drag n_pbtutor_connectservice from the System Tree to the Script view.

   Dragging object and function names from the System Tree to the Script view saves time and helps avoid typing errors.

9. Complete the line by typing the variable name after the object name:

```
n_pbtutor_connectservice gnv_connect
```

Although you declare this object in the Script view for the n_pbtutor_connectservice user object, it is available everywhere in the application.

Naming conventions for variables

To make scripts easier to read, it is best to follow a standard naming convention. The recommended standard is to give each variable a 2-letter or 3-letter prefix followed by an underscore (_). The first letter of the prefix identifies the scope of the variable (for example: g for global, l for local) and the next letter or letters identify the data type (for example: s for string, l for long, or nv for nonvisual object).

10.Click the Save button in the PainterBar

or

Select File>Save from the menu bar.

PowerBuilder compiles the script and saves it. If you had typed the global variable data type (instead of dragging it from the System Tree) and you made a typing error, an error message would display. You would then correct the error and select Save again.

2.4.4 Modify the connection information

Where you are

Look at the Demo Database

Run the Connection Object wizard

Declare a global variable

> Modify the connection information

Complete the login and logout scripts

Run the application

You can now call the connection service manager to establish a database connection, but you should open a database connection only if the user enters a valid ID and password in the login window. You will therefore add the connection service call to the Clicked event of the OK button on this window, substituting user-entered information for information from the pbtutor.ini file.

However, before you add the call to the OK button, you remove or comment out the ProfileString calls that the connection service manager makes to get user ID and password information from the INI file. Then you modify the DBParm parameter in the pbtutor.ini file, because it includes hard-coded user ID and password values that were copied from the pb.ini file.

In this exercise you:

- Modify the of_GetConnectionInfo function
- Call the connection service manager

2.4.4.1 Modify the of_GetConnectionInfo function

You looked at the of_GetConnectionInfo function in the last exercise. Now you comment out the information that the function returns for the user ID and password information.

If you closed the User Object painter, you must open it again for the n_pbtutor_connectservice user object. You can use the File>Recent Objects menu to redisplay it.

1. Select Functions in the first drop-down list box in the Script view.

2. Select of_GetConnectionInfo in the second drop-down list box.

3. Select the two ProfileString assignment lines that begin:

   ```
   as_userid       = ProfileString (...)
   as_dbpass       = ProfileString (...)
   ```

   The four arguments of a ProfileString call are the INI file name or variable, the INI file section, the INI file key, and the default value to be used if the INI file name, section, or key is incorrect. These lines are part of the IS_USE_INIFILE case of the CHOOSE CASE statement for the of_GetConnectionInfo function.

4. Click the Comment button (Comment) in PainterBar2.

   By commenting out these lines, you make sure that the user ID and password information do not come from the pbtutor.ini file.

5. Click anywhere in the line that begins:

   ```
   as_dbparm       = ProfileString ( ... )
   ```

6. Click the Comment button in PainterBar2.

   The DBParm parameter in the pbtutor.ini file includes hard-coded values for the user ID and password as well as the database name. You do not use these values. Instead, you assign values to the DBParm parameter from user-entry information for user ID and password.

   About the SQLCA DBParm parameter

   Although the user ID and password are not required for the DBParm ConnectString, assigning them to the ConnectString overwrites SQLCA user ID and password values in the data source definition for an SQL Anywhere database. For this DBMS, the DBParm parameter also takes precedence over the SQLCA UserID and DBPass parameters.

7. Click the Save button in PainterBar1.

8. Click the Close button in PainterBar1.

2.4.4.2 Call the connection service manager

You will next call the connection service manager to connect to the database. Because you eventually need to add user-entry information from the login window, you add the call to the Clicked event for the OK button on this window.

An object is considered to be the parent of the controls that are added to it. The login window is therefore the parent of the OK button.

When referring to a parent object in a script, it is usually better practice to use the qualifier parent than to name the object explicitly. This allows the code to be reused more easily for controls placed on a different object. In the script for the Clicked event, you refer to the login window as parent.

Using a single wizard to create the application and connection

If you had created the connection service user object with the Template Application wizard, the code you enter in this exercise to call the connection service manager would have been generated automatically, but it would have been added to the application Open event, not to a Clicked event in a login window. It would also have used a local variable, not a global variable.

1. Double-click w_welcome in the System Tree.

   The Window painter opens.

2. Select cb_ok in the first drop-down list box of the Script view

   or

   Double-click the OK button in the Layout view.

   The Clicked event should be the selected event in the second drop-down list box. If it is not, select it. The Clicked event script is empty.

3. Type these lines:

   ```
   // 1) Instantiate the Transaction object
   // 2) Close login window if connection successful
   ```

   These lines explain the code you add to the Clicked event. Adding double slash marks at the front of a line turns it into a comment.

4. Type the following assignment statement below the comments:

   ```
   gnv_connect = CREATE &
            n_pbtutor_connectservice
   ```

   Do not type the ampersand (&) if you combine the lines of the script into a single line. The ampersand character indicates that a line of script is continued on the next line.

   The CREATE statement instantiates the SQLCA Transaction object with all the values retrieved by the of_GetConnectionInfo function from the pbtutor.ini file. Because you previously commented out the lines for the user ID and password, this information is not retrieved.

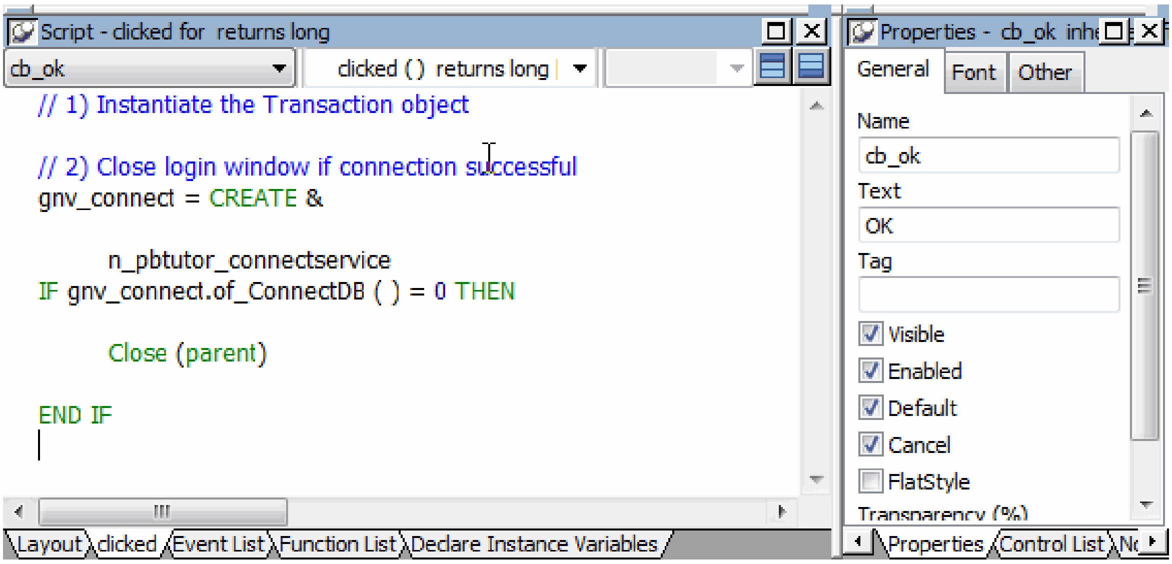
   For ease of reading, you can add blank lines between the comments and the assignment statement for the global variable gnv_connect.

5. Type the following lines below the CREATE statement:

   ```
   IF gnv_connect.of_ConnectDB ( ) = 0 THEN
            Close (parent)
   END IF
   ```

The of_ConnectDB function connects the application to the database. As you saw earlier in this lesson, if the connection fails (the SQLCode is not 0), a message box opens and displays the SQL error text.

If of_ConnectDB returns a zero (the SQLCode for a successful connection), the lines that follow the IF-THEN statement line are parsed. In this case, the parent window for the cb_ok control (w_welcome) closes.



```
// 1) Instantiate the Transaction object

// 2) Close login window if connection successful
gnv_connect = CREATE &

        n_pbtutor_connectservice
IF gnv_connect.of_ConnectDB ( ) = 0 THEN

        Close (parent)

END IF
```

Name: cb_ok

Text: OK

Tag:

☑ Visible

☑ Enabled

☑ Default

☑ Cancel

□ FlatStyle

Transparency (%)

6. Click the Compile button in PainterBar2

   or

   Right-click inside the Script view and click Compile in the pop-up menu.

   The script should compile without error. If you get an error message, make sure you have typed object and function names correctly and saved gnv_connect as a global variable.

Toggling the Error window of the Script view

You can show or hide the Error window by clicking the icon at the far right of the Script view just under the title bar.

You still need to code the Clicked event to instantiate the Transaction object with user-entered connection information.

2.4.5 Complete the login and logout scripts

Where you are

Look at the Demo Database

Run the Connection Object wizard

Declare a global variable

Modify the connection information

> Complete the login and logout scripts

Run the application

Earlier in this lesson, you called the connection service manager from the Clicked event for the login window OK button. Next you add code to the same Clicked event to instantiate the Transaction object with information entered by the user.

You also add code to the login window Cancel button Clicked event and to the application Closed event.

Minimizing typing errors in the Script view

If you right-click inside the scripting area, you open a pop-up menu that includes Paste Special commands. You can use these commands to paste statements, objects, functions or even the contents of text files into the event script. This reduces the risk of typing errors. You can also use AutoScript to complete code, as you will see in this lesson.

In this exercise you:

- Set up shortcuts for AutoScript
- Add code to the OK button Clicked event
- Add code to the Cancel button Clicked event
- Add code to the application Close event

2.4.5.1 Set up shortcuts for AutoScript

When you are coding scripts, AutoScript provides help by completing the statement you are typing or displaying a list of language elements that are appropriate to insert in the script at the cursor location.

1. Select Design>Options from the menu bar and click the AutoScript tab.

2. Make sure all the check boxes in the first three group boxes are selected.

3. Make sure the Activate Only After A Dot and Automatic Popup check boxes in the fourth group box are cleared, and click OK.

   With these settings, AutoScript provides Help wherever it has enough information to determine what code to insert, but it does not pop up automatically when you pause. By selecting the Statement Templates check box (not selected by default), you can use AutoScript to enter structures for a multiple line PowerScript statement.

4. Select Tools>Keyboard Shortcuts from the menu bar.

   Expand the Edit node in the tree.

5. Scroll down and select Activate AutoScript.

   With your cursor in the Press Keys For Shortcut box, press Ctrl+Space.

6. Expand the Edit>Go To node in the Current Menu list and select Next Marker.

   With your cursor in the shortcut box, press Ctrl+M.

Now whenever you want help completing code, you can press Ctrl+space to see a list of possible completions. If you paste a statement or function with comments, you can press Ctrl+M to move to the next comment.

7. Click OK.

2.4.5.2 Add code to the OK button Clicked event

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

For the tutorial application, you should not get the user ID and password from the tutorial INI file. Get them directly from the user in the login window and then pass the database information in a script.

1. Make sure you are looking at the Clicked event script for the cb_ok control.

   This is the script in which you added the call to the Connection Service object.

2. Click before the IF-THEN statement.

   Type the following lines:

   ```
   //Local variable declarations
   string ls_database, ls_userid, ls_password

   //Assignment statements
   ls_userid = Trim ( sle_userid.text )
   ls_password = Trim ( sle_password.text )
   ls_database="ConnectString='DSN=PB Demo DB V2019R3;"
   ```

   With these lines you declare local variables and assign them values. Do not use blank spaces around the = signs in the ConnectString text. Do not worry about the lone single quotation mark. You will add a single quotation mark in the next step to complete the connection script.

   Using AutoScript to help code the assignment statements

   When you type the assignment statements, if you type the letters before the underscore in a variable name and then press Ctrl+space, AutoScript pops up a list of possible completions. Use the arrow keys to move to the correct completion and the Tab key to paste it into your script. If you type the underscore and the first letter after the underscore and then press Ctrl+space, AutoScript pastes the completion directly into your script, as long as there is a unique completion.

   The Trim function removes leading and trailing spaces from the user ID and password values passed as arguments to the function from the SingleLineEdit boxes on the login window.

3. Click after the lines you just added (which follow the CREATE statement) but before the IF-THEN statement.

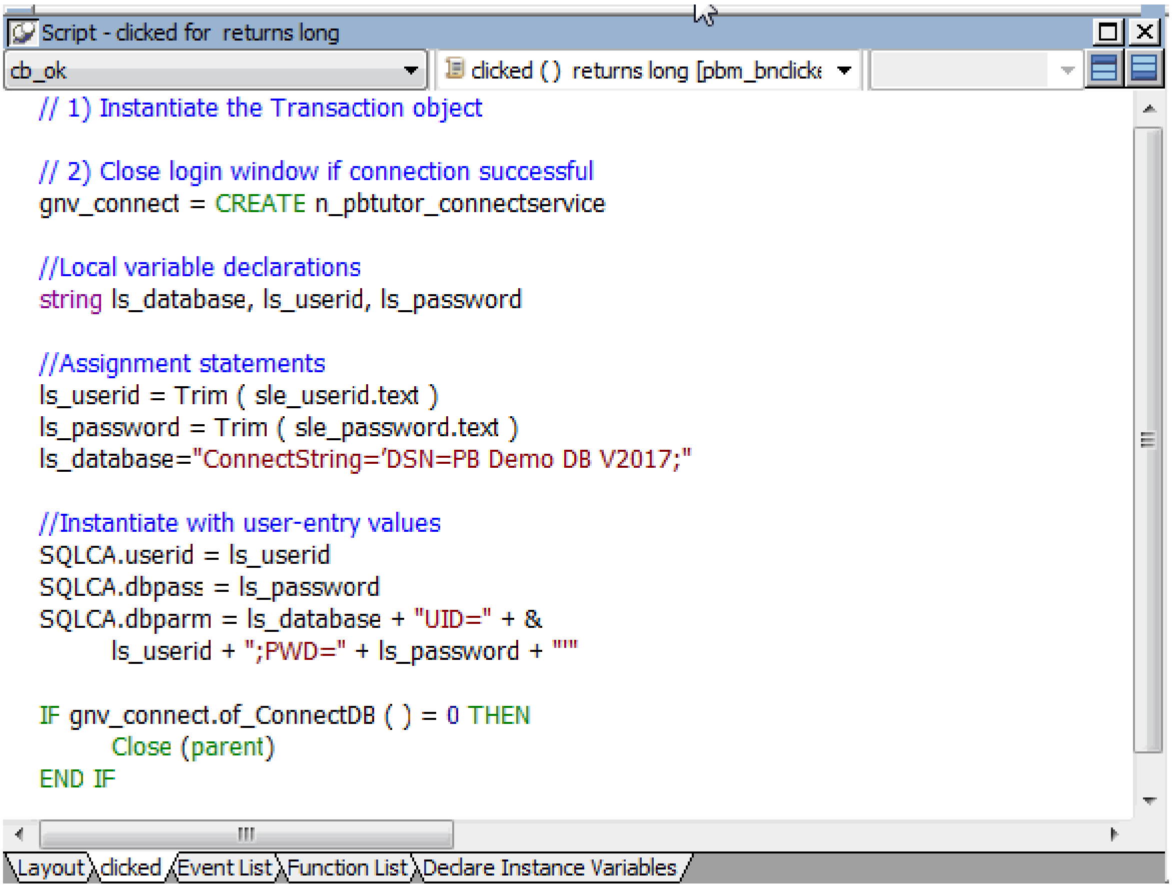
   Type the following lines:

   ```
   //Instantiate with user-entry values
   SQLCA.userid = ls_userid
   ```

```
SQLCA.dbpass = ls_password
SQLCA.dbparm = ls_database + "UID=" + &
         ls_userid + ";PWD=" + ls_password + "'"
```

These lines instantiate SQLCA parameters with values from the SingleLineEdit text boxes.

The lines must be added to the script after the CREATE statement to keep them from being overwritten with blank values from the Constructor event of the connection service manager. They must be added before the IF-THEN statement or their values are not used by the Transaction object when it is called by the of_ConnectDB function of the connection service manager.


Script - clicked for returns long
cb_ok
clicked ( ) returns long [pbm_bnclick
Layout
clicked
Event List
Function List
Declare Instance Variables

```
// 1) Instantiate the Transaction object

// 2) Close login window if connection successful
gnv_connect = CREATE n_pbtutor_connectservice

//Local variable declarations
string ls_database, ls_userid, ls_password

//Assignment statements
ls_userid = Trim ( sle_userid.text )
ls_password = Trim ( sle_password.text )
ls_database="ConnectString='DSN=PB Demo DB V2017;"

//Instantiate with user-entry values
SQLCA.userid = ls_userid
SQLCA.dbpass = ls_password
SQLCA.dbparm = ls_database + "UID=" + &
      ls_userid + ";PWD=" + ls_password + "'"

IF gnv_connect.of_ConnectDB ( ) = 0 THEN
      Close (parent)
END IF
```

4. Click the Compile button in PainterBar2

   or

   Right-click inside the Script view and click Compile in the pop-up menu.

   The script should compile without error. If you get an error message, make sure you have typed object and function names correctly.

2.4.5.3 Add code to the Cancel button Clicked event

Now you add code to the Cancel button to stop the application when this button is clicked.

1. Double-click the Cancel button in the Layout view

   or

Select cb_cancel in the first drop-down list box of the Script view.

The script area for the Cancel button is blank.

2. Type this one-line script for the Clicked event:

   ```
   HALT
   ```

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

3. Click the Save button in the PainterBar

   or

   Select File>Save from the menu bar.

   PowerBuilder compiles the script.

4. Click the Close button in the PainterBar

   or

   Select File>Close from the menu bar.

   The Window painter closes.

2.4.5.4 Add code to the application Close event

Because the connection service manager was called by a global variable, it is still available to the application and does not need to be instantiated again (as it would if you had used a local variable).

Now you call the connection service manager disconnect function to close the database connection.

1. Double-click the pbtutor application icon in the System Tree.

   The Application painter displays different views of the tutorial application object. The Script view is part of a stack in the default layout, but you might find it easier to detach it from the stack or open a second Script view.

2. Select close ( ) returns (none) in the second drop-down list box of the Script view.

   There is no code yet for the application Close event.

3. Type the following lines for the Close event comment:

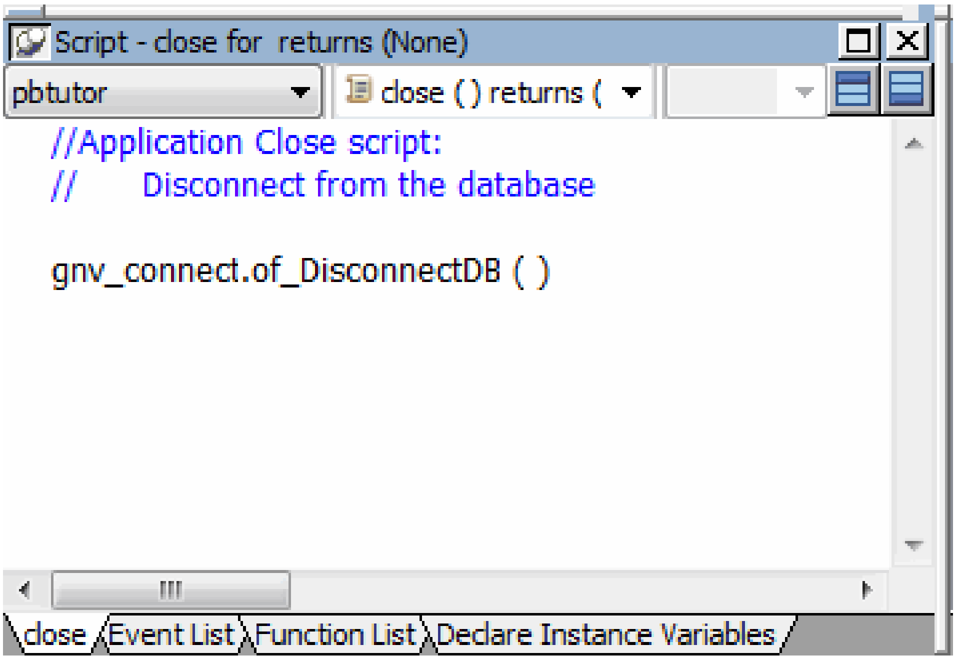
   ```
   Application Close script:
         Disconnect from the database
   ```

4. Select all or part of the lines you just added.

   Click the Comment button.

5. Type the following line below the comment you typed (you can use AutoScript to complete the variable name and the function name):

```
gnv_connect.of_DisconnectDB ( )
```



```
//Application Close script:
//      Disconnect from the database

gnv_connect.of_DisconnectDB ( )
```

Releasing memory by setting global variables to null

If this were not the application Close event and you no longer needed an instance of the global connection variable, you could release the memory it occupies by calling the SetNull function.

PowerBuilder also provides a DESTROY statement to destroy object instances. Do not use the DESTROY statement for local or global variables for nonvisual objects. PowerBuilder garbage collection removes any local variables that go out of scope.

6. Right-click anywhere in the script area of the Script view.

   Click Compile in the pop-up menu.

   PowerBuilder compiles the Close script. If you get an error message, look carefully at the lines you typed to make sure there is no mistyped variable or object name.

7. Click the Close button in PainterBar1.

   A message box asks if you want to save your changes to the Application object in the application library file.

8. Click Yes.

   This saves your changes and closes the Application painter.

2.4.6 Run the application

Where you are

Look at the Demo Database

Run the Connection Object wizard

Declare a global variable

Modify the connection information

Complete the login and logout scripts

> Run the application

Now you run the application.

1. Click the Run button (Run) in the PowerBar.

   If a message box prompts you to save changes, click Yes to save them.

   The workspace closes and your application runs.

2. Type dba in the User ID box.

   Type sql in the Password box.

   The password text is displayed as asterisks. Because you set the tab order for this window, you can tab from the User ID box to the Password box, and then to the OK button.

3. Click OK.

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

   If you enter an invalid user ID or password

   If you mistyped the user ID or password, the Connect to SQL Anywhere (ODBC Configuration) dialog box displays. You get a second chance to enter a valid user ID and password on the Login page of this dialog box. If you click the Test Connection button on the ODBC page of the dialog box without changing this information, a message box tells you that your user ID or password is not valid.

4. Select File>Exit from the menu bar.

   The application terminates and you return to the development environment.

2.5 LESSON 5 Modifying the Ancestor Window

In this lesson you create a window that inherits from the basesheet window that you generated with the Application Template wizard. You add predefined user objects containing DataWindow controls to the inherited window. You then create new w_customers and w_products windows that inherit from this extension layer window instead of from the original basesheet. Finally you make sure that the sheet windows open at runtime with the size you set at design time.

But first you add a library to the library list that contains the predefined user objects with DataWindow controls. In this lesson you:

- Add a library to the search path

- Create a new ancestor sheet window

- Add user events and event scripts
- Add scripts to retrieve data for the DataWindow controls
- Adjust a runtime setting for sheet window size

How long does it take?

About 30 minutes.

2.5.1 Add a library to the search path

Where you are

> Add a library to the search path

Create a new ancestor sheet window

Add user events and event scripts

Add scripts to retrieve data for the DataWindow controls

Adjust a runtime setting for sheet window size

Next you add a library to the tutorial application search path. You must add all libraries on which an application depends.

The library you add to the current application contains some precoded objects, including the user object (u_dwstandard) that you will later add to the base sheet window.

1. Right-click the pbtutor target (not the pbtutor application object) in the System Tree.

   The pbtutor target contains the pbtutor.pbl and the pbtutor application.

2. Select Library List from the pop-up menu.

   The pbtutor target Properties dialog box displays the Library List page.

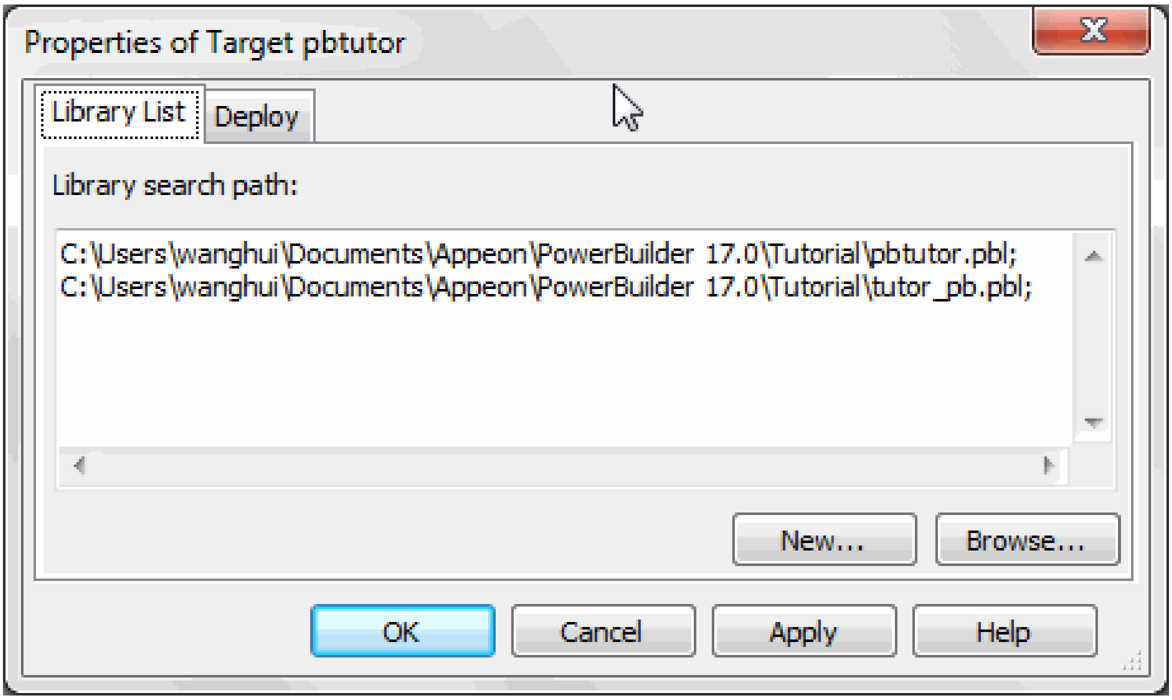
3. Click Browse.

   The Select Library dialog box displays.

4. Navigate to the Tutorial folder.

   Select tutor_pb.pbl and click Open.

   You return to the Library List page. The tutor_pb.pbl file is now included in the search path for the tutorial application.



Library search path:

C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial\pbtutor.pbl;
C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial\tutor_pb.pbl;

5. Click OK.

2.5.2 Create a new ancestor sheet window

Where you are

Add a library to the search path

> Create a new ancestor sheet window

Add user events and event scripts

Add scripts to retrieve data for the DataWindow controls

Adjust a runtime setting for sheet window size

Now you create a window that inherits from the basesheet window you generated with the Template Application wizard and add DataWindow controls to it. In the Source editor, you change the inheritance of the generated sheet windows (w_customers and w_products) to use the new window.

The DataWindow controls you add to the new ancestor window inherit their definitions from a user object that was created for the tutorial application. The user object is provided in the PBL file that you just added to the target library list. The user object is a customized DataWindow control that includes scripts to perform standard database error checking.

Why use a user object

You can build a user object in PowerBuilder to perform processing that you use frequently in applications. Once you have defined a user object, you can reuse it as many times as you need without any additional work.

In this exercise you:

- Create a new sheet window inheritance hierarchy
- Add a DataWindow control for the master DataWindow
- Add a DataWindow control for the detail DataWindow
- View the scripts inherited from the user object

2.5.3 Create a new sheet window inheritance hierarchy

The new window you create now is an extension layer between the basesheet window and application sheet windows. Later in this lesson you make changes to the extension layer window. The changes you make are automatically extended to any new sheet windows that you inherit from the extension layer window.

In the current tutorial application, the w_customers and w_products windows already inherit from the w_pbtutor_basesheet window. Because you have not yet added any non-generic property values or functions to these sheet windows (other than their names and display text), you can write over these wizard-generated windows without having to transfer any code to the replacement windows. In this lesson you overwrite these windows with new windows that inherit from the extension layer window.

1. Select File>Inherit from the PowerBuilder menu.

   The Inherit from Object dialog box displays.

2. Make sure that pbtutor.pbl is selected in the Libraries list box and that Windows is selected in the Objects Of Type drop-down list.

   If you cannot see the full library list, you can change the size of the dialog box by clicking on one of its edges and holding down the mouse button while you drag the edge toward a corner of the screen. The pbtutor.pbl should be the first of two libraries listed in the Libraries list box.

3. Select w_pbtutor_basesheet in the Object column of the main list box and click OK.

4. Select File>Save As, and in the Save Window dialog box, select w_master_detail_ancestor in the Windows field for the new window name.

5. (Optional) Type the following text in the Comments box:

   ```
   New ancestor basesheet for the w_customers and w_products sheet windows.
   ```

6. Make sure that pbtutor.pbl is selected in the Application Libraries list box and click OK.

   Select File>Close to close the new ancestor basesheet.

   You cannot create descendant windows if an ancestor window is open in the Window painter.

7. Select File>Inherit from the PowerBuilder menu.

8. Make sure that pbtutor.pbl is selected in the Libraries list box and that Windows is selected in the Objects Of Type drop-down list box.

   Select w_master_detail_ancestor and click OK.

9. Type Maintain Customers in the Tag text box on the General page of the Properties view.

   Select File>Save As from the PowerBuilder menu and select w_customers in the Windows list box.

10. Change the Comments text to:

    ```
    Customer sheet window inheriting from w_master_detail_ancestor.
    ```

11. Click OK, then click Yes in the Save Window message box that asks if you want to replace the existing w_customers window.

    The new sheet window inherits from w_master_detail_ancestor instead of from w_pbtutor_basesheet.

12. Repeat steps 7-11, with the following modifications:

    Table 2.6:

Step	Modified instruction
9	Type Maintain Products in the Tag text box on the General page of the Properties view. Select File>Save As from the PowerBuilder menu and select w_products in the Windows list box.
10	Change the Comments text to: Product sheet window inheriting from w_master_detail_ancestor.
11	The message box prompts you to replace the existing w_products window.

13. Close the new w_customers and w_products windows.

    You cannot open an ancestor window in the Window painter if any of its descendants are already displayed in the painter.

14. From the PowerBuilder menu, select Run>Full Build Workspace.

    You should rebuild the workspace after changing the inheritance hierarchy and before making modifications to the new ancestor window. You can see the status of the build in the Output window, which displays below the System Tree at the bottom of the PowerBuilder main window. The build is finished when the Output window displays Finished Full build of workspace MyWorkspace.

15. Close the Output window.

2.5.4 Add a DataWindow control for the master DataWindow

Now you add a DataWindow control (saved as the user object, u_dwstandard) to the w_master_detail_ancestor window. It serves as the master DataWindow for the ancestor window and its descendants.

How to create a user object like u_dwstandard

You can create a user object based on a DataWindow control by clicking the New button and selecting Standard Visual from the PB Object page of the New dialog box. This opens the Select Standard Visual Type dialog box. You can then select

DataWindow in the Types text box and add user events as needed. You see how to add user events later in this tutorial.

1. Double-click w_master_detail_ancestor in the System Tree.

   The w_master_detail_ancestor window opens in the Window painter. You generated this window with the Template Application wizard. The wizard also created and attached a menu to this window, m_pbtutor_sheet. The menu is indicated in the Properties view for the window. You change this property later.

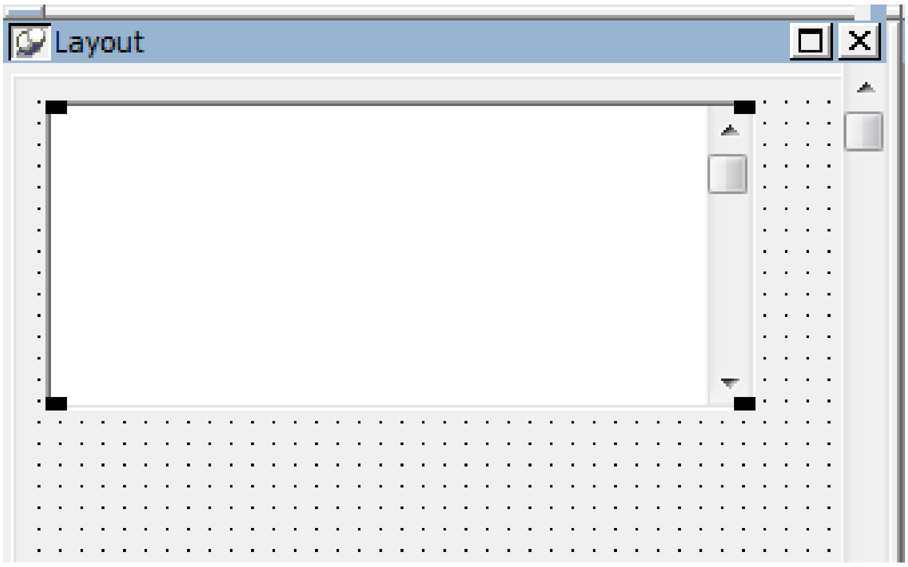
2. Make sure the Layout view is visible in the Window painter.

3. Expand tutor_pb.pbl by double-clicking it in the System Tree.

4. Drag u_dwstandard from the System Tree to the w_master_detail_ancestor window in the Layout view.

5. Widen the window so that the control is completely visible inside the window.

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



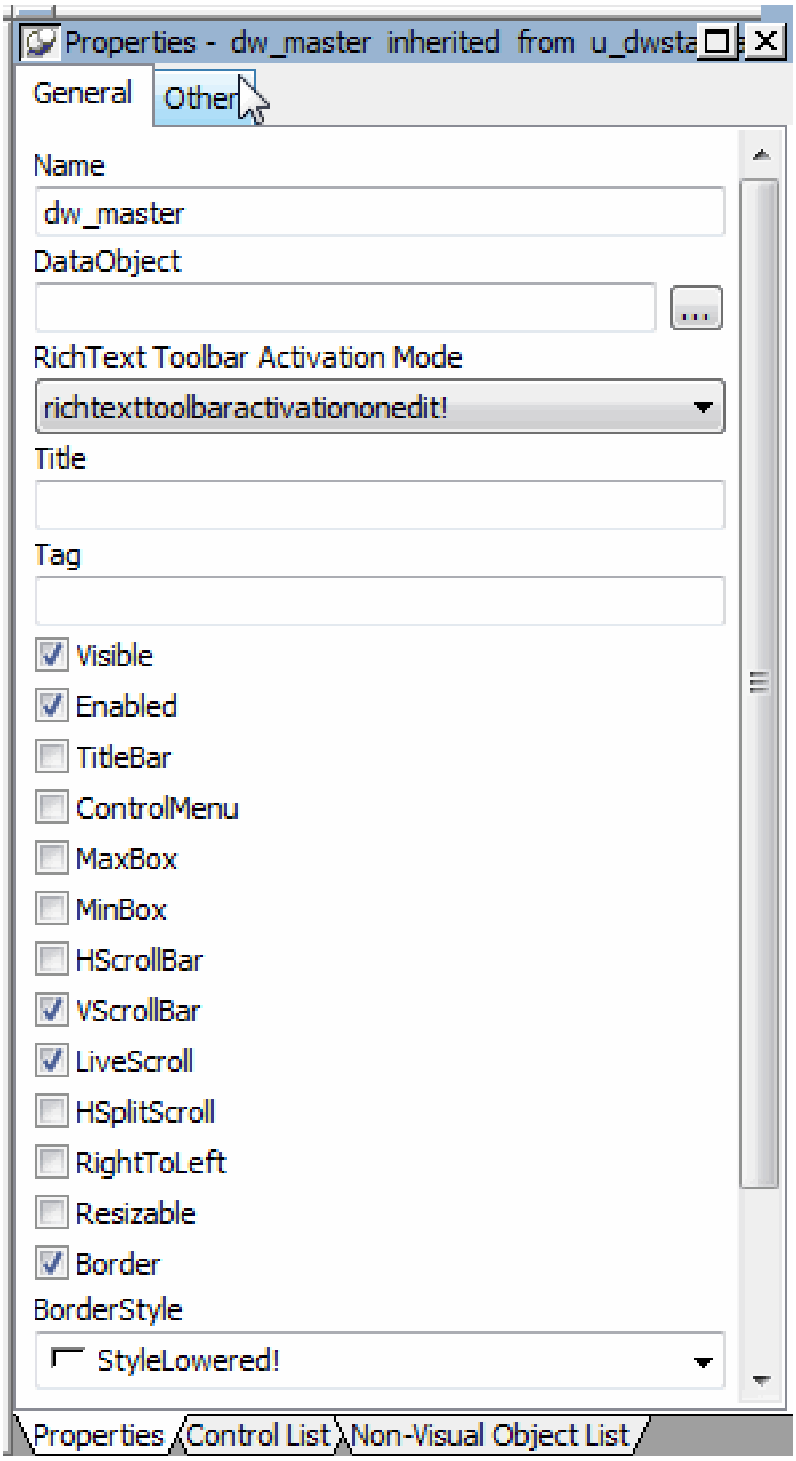
6. Make sure the new control is selected in the Layout view.

   Small black squares at the corners indicate that the control is selected. The Properties view displays the properties of the selected control.

7. Select the text dw_1 in the Name text box in the Properties view.

8. Type dw_master in the Name text box.

   Select the VScrollBar check box.



Name: dw_master

DataObject:

RichText Toolbar Activation Mode: richtexttoolbaractivationonedit!

Title:

Tag:

☑ Visible
☑ Enabled
☐ TitleBar
☐ ControlMenu
☐ MaxBox
☐ MinBox
☐ HScrollBar
☑ VScrollBar
☑ LiveScroll
☐ HSplitScroll
☐ RightToLeft
☐ Resizable
☑ Border

BorderStyle: StyleLowered!

PowerBuilder adds a vertical scroll bar to the control. It also changes its name to dw_master. The prefix dw_ is standard for DataWindow controls.

2.5.5 Add a DataWindow control for the detail DataWindow

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

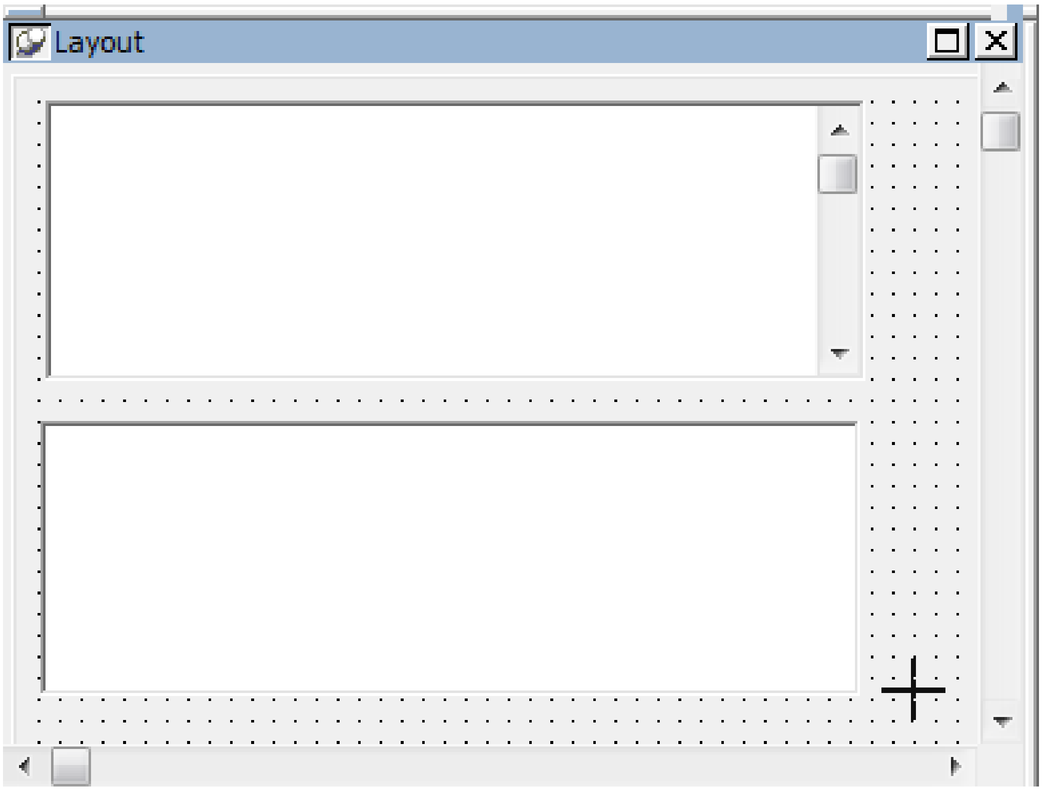
1. Resize the window so that there is room for a second DataWindow control below the first.

Drag u_dwstandard from the System Tree to below the dw_master control in the Layout view.

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

2. Move the DataWindow control so that it is completely visible inside the window.

   If you need to, you can maximize the Layout view and enlarge the window object inside it to make more room for the DataWindow controls.



3. Make sure that the new control is selected in the Layout view.

   The Properties view displays the properties of the selected control.

4. Replace the text dw_1 in the Name text box in the Properties view with

   dw_detail

   PowerBuilder changes the name of the control to dw_detail.

2.5.6 View the scripts inherited from the user object

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

1. Double-click the dw_detail DataWindow in the Layout view

   or

   Select dw_detail from the first drop-down list box in the Script view if it is not already selected.

The Script view opens to the empty script for the dw_detail control's ItemChanged event.

Unscripted events are alphabetized separately from scripted events. Scripted events are listed at the top of the drop-down list box. You will next look at the dberror event, which contains an ancestor script, so you need to scroll up in the event drop-down list box to find it.

2. Select dberror from the second drop-down list box in the Script view.

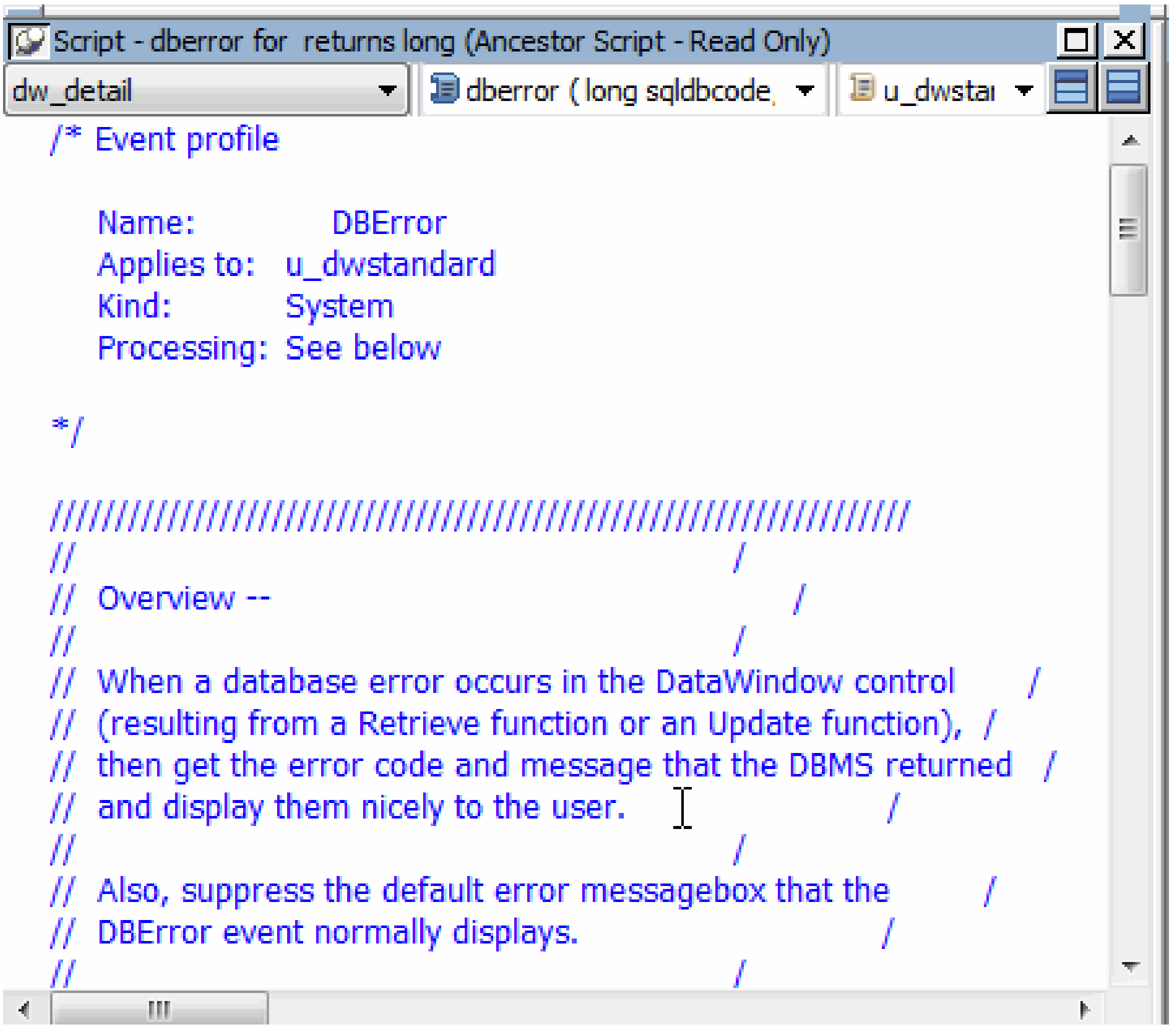
   This script is also empty, but a purple script icon displays next to the event name. This indicates that the ancestor control (u_dwstandard) has an associated script.

3. Select Edit>Go To>Ancestor Script from the menu bar

   or

   Select u_dwstandard in the third drop-down list box.

   PowerBuilder displays the script for the DBError event in the Script view. The ancestor script is read-only when it is accessed from the Script view for one of its descendants.


Script - dberror for returns long (Ancestor Script - Read Only)
dw_detail
dberror (long sqldbcode,
u_dwstar

```
/* Event profile

    Name:           DBError
    Applies to:   u_dwstandard
    Kind:           System
    Processing:  See below

*/

///////////////////////////////////////////////////////////////////////
//                                                        /
//  Overview --                                           /
//                                                        /
//  When a database error occurs in the DataWindow control      /
//  (resulting from a Retrieve function or an Update function),  /
//  then get the error code and message that the DBMS returned   /
//  and display them nicely to the user.                  /
//                                                        /
//  Also, suppress the default error messagebox that the        /
//  DBError event normally displays.                      /
//                                                        /
```

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

The script suppresses the default error message that the DBError event normally displays. Instead, it causes an appropriate message to be displayed for each database error that might occur. The script makes calls to user events that were declared for the user object.

Because you used the u_dwstandard object to define both DataWindow controls in the window, this logic is automatically reused in both controls.

5. Select Edit>Go To>Descendant Script from the menu bar

   or

   Right-click inside the script area of the Script view.

   Select Go To>Descendant Script from the pop-up menu.

   The third drop-down list box again displays w_master_detail_ancestor, the identifier of the object that contains the current control. The script for the DBError event of this control (dw_detail) is still blank.

2.5.7 Add user events and event scripts

Where you are

Add a library to the search path

Create a new ancestor sheet window

> Add user events and event scripts

Add scripts to retrieve data for the DataWindow controls

Adjust a runtime setting for sheet window size

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 want to declare your own events. Events that you define are called user events.

Purpose of user events

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, a task like updating the database can be performed by clicking a button, selecting a menu item, or closing a window. Instead of writing the code to update the database in each of these places, you can define a user event, then trigger that user event in each place in which you update the database.

Now you define some user events to handle retrieval, insert, update, and delete operations against the tutorial database. You make these changes in the Script view of the Window painter. Later in the tutorial, you add code in the Menu painter to trigger these events.

1. Select w_master_detail_ancestor in the first drop-down list box of the Script view.

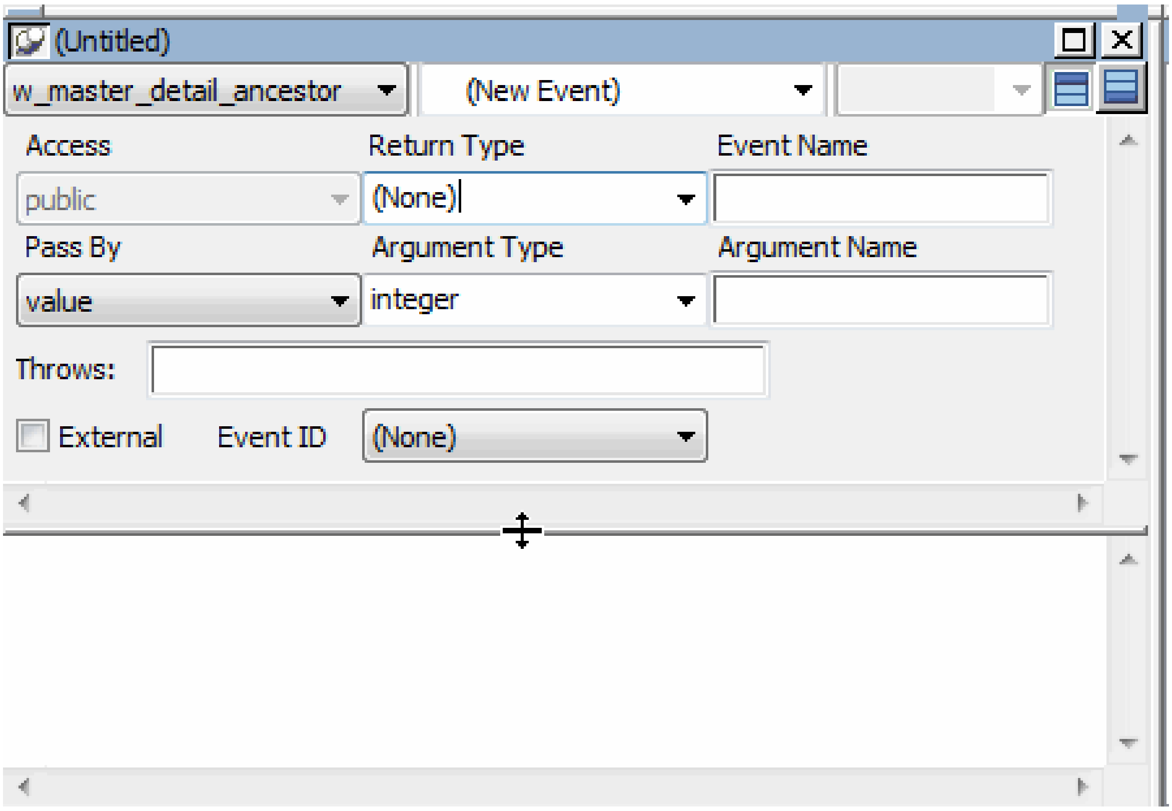
2. Select Insert>Event from the menu bar

   or

   Select New Event in the second drop-down list box of the Script view.

   The Script view displays the Prototype window for defining a new event.

The first button to the right of the third drop-down list box is a toggle switch that displays or hides the Prototype window.



w_master_detail_ancestor | (New Event)

Access	Return Type	Event Name
public	(None)	
Pass By	**Argument Type**	**Argument Name**
value	integer	

Throws:

□ External   Event ID (None)

3. Type ue_retrieve in the Event Name text box in the Prototype window.

   Click inside the Script view below the Prototype window.

   Type these lines (or use AutoScript as described below):

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

Using AutoScript instead of typing

You can use AutoScript to paste in the IF THEN template as well as the variables and function names:

Type IF, then press Ctrl+space.

Press Tab to paste an IF THEN statement.

Type dw_m, then press Ctrl+space.

Place the cursor after dw_master, type a dot, then type Ctrl+space.

Scroll and select retrieve( ), press Tab, and type the rest of the line.

Press Ctrl+M to jump to the next comment.

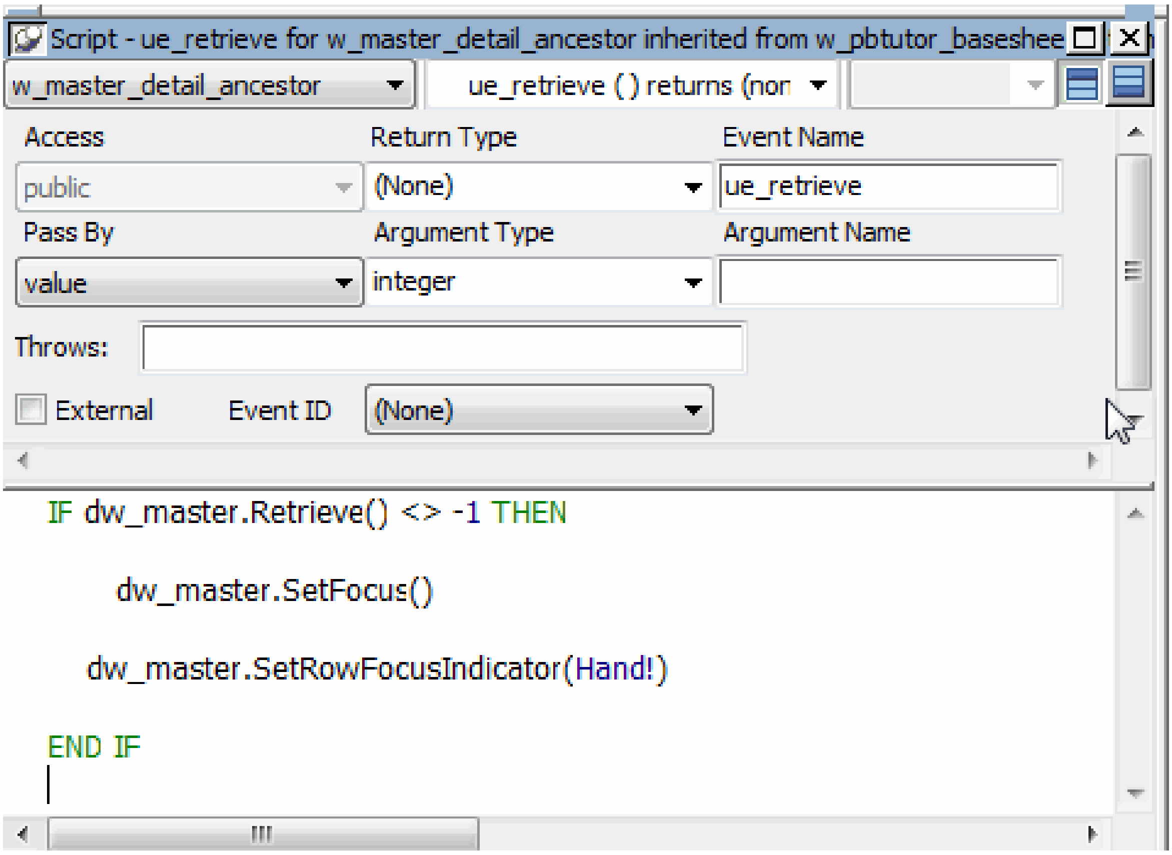
Enter the other function calls by typing them or using AutoScript.

As soon as you clicked in the script area, the text in the second drop-down list box of the Script view changed from New Event to ue_retrieve. It has no arguments, does not return a value, and does not throw user-defined exceptions. For information on throwing user-defined exceptions, see Exception Handling

The script lines you entered execute the Retrieve function and place the retrieved rows in the dw_master DataWindow control. If the retrieval operation succeeds, the script sets the focus to the first row in the DataWindow control and establishes the hand pointer as the current row indicator.

If the retrieve fails

If the retrieval operation does not succeed, PowerBuilder triggers the DBError event. The logic for the DBError event is handled in the user object u_dwstandard. You looked at this script in the previous exercise.



Script - ue_retrieve for w_master_detail_ancestor inherited from w_pbtutor_basesheet
w_master_detail_ancestor
ue_retrieve ( ) returns (non

Access: public
Return Type: (None)
Event Name: ue_retrieve
Pass By: value
Argument Type: integer
Argument Name:
Throws:
□ External
Event ID: (None)

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

4. Select File>Save from the menu bar.

   Right-click the Prototype window and select New Event from the pop-up menu.

   PowerBuilder compiles the script you entered for the ue_retrieve event. The Script view displays the Prototype window for another new user event.

If you get an error message

Mistyped or incomplete script entries generate compiler errors. If you select No when prompted to ignore compilation errors, a compiler error area displays at the bottom of the Script view, identifying your error. If this happens, retype the script for the ue_retrieve event.

You can display or hide the compiler error area by clicking the second toggle switch at the top right of the Script view.

5. Repeat steps 3 and 4 for the following entries:

Table 2.7:

Event name	Script

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

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

ue_delete	`dw_detail.DeleteRow(0)`

What the scripts do

The first line of the script for the ue_insert event clears the dw_detail DataWindow control. The second line inserts a new row after the last row in the DataWindow (the argument zero specifies the last row). The third line positions the cursor in the dw_detail control.

The ue_insert and ue_delete events operate on the DataWindow buffer, not on the database. When these events are triggered, a row is not inserted or deleted from the database unless the Update function is also called (the ue_update event calls this function). If the Update function returns the integer 1, changes made to the buffer are committed to the database. If it returns a different integer, changes to the buffer are rolled back.

In the script for the ue_delete event, the argument zero in the DeleteRow function specifies that the current row in the dw_detail control be deleted.

6. Make sure your work is saved.

   If you repeated step 4 for each new event and script that you added, you have already saved your work.

2.5.8 Add scripts to retrieve data for the DataWindow controls

Where you are

Add a library to the search path

Create a new ancestor sheet window

Add user events and event scripts

> Add scripts to retrieve data for the DataWindow controls

Adjust a runtime setting for sheet window size

The scripts you just typed have no effect on the dw_master DataWindow control, but now that you have a script for the ue_retrieve event, you need only trigger this event to retrieve data into the dw_master DataWindow.

You trigger the ue_retrieve event from the sheet window Open event. This retrieves data into the dw_master DataWindow as soon as the window (or one of its descendant windows) opens. Then you add a script for the RowFocusChanged event of dw_master to retrieve data into the dw_detail DataWindow. The RowFocusChanged event is triggered each time the focus is changed inside the dw_master DataWindow.

RowFocusChanged occurs upon DataWindow display

The RowFocusChanged event also occurs when the w_master DataWindow is first displayed. This allows the application to retrieve and display detail information for the first row retrieved in the master DataWindow.

Here is how the script works for the w_master_detail_ancestor window and its descendants when you are done:

- When a sheet window first opens, a list (of all customers or products) displays in the top DataWindow control. Detail information for the first item in the list 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.

1. Select open from the second drop-down list box in the Script view for w_master_detail_ancestor.

   The Open event has a purple script icon indicating it has an ancestor script. If you check the ancestor script, you see that it calls the ue_postopen event and posts it to the end of the window's message queue.

2. Type these new lines in the script area for the w_master_detail_ancestor Open event:

   ```
   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 w_master_detail_ancestor window is the current object.

3. Select dw_master in the first drop-down list box of the Script view.

4. Select rowfocuschanged in the second drop-down list box.

Read the event name carefully

Make sure you select the RowFocusChanged event, and not the RowFocusChanging event.

You now add a script for the RowFocusChanged event of the dw_master DataWindow control. This script sends a retrieval request and the ID number of the selected row to the dw_detail DataWindow control.

5. Type this line in the script area for the RowFocusChanged event:

   ```
   long ll_itemnum
   ```

   This line declares the local variable ll_itemnum (l is a letter, not a number), which has the long data type.

6. Type this line below the variable declaration line you just typed:

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

   Use square brackets

   The expression shown above requires square brackets, not parentheses.

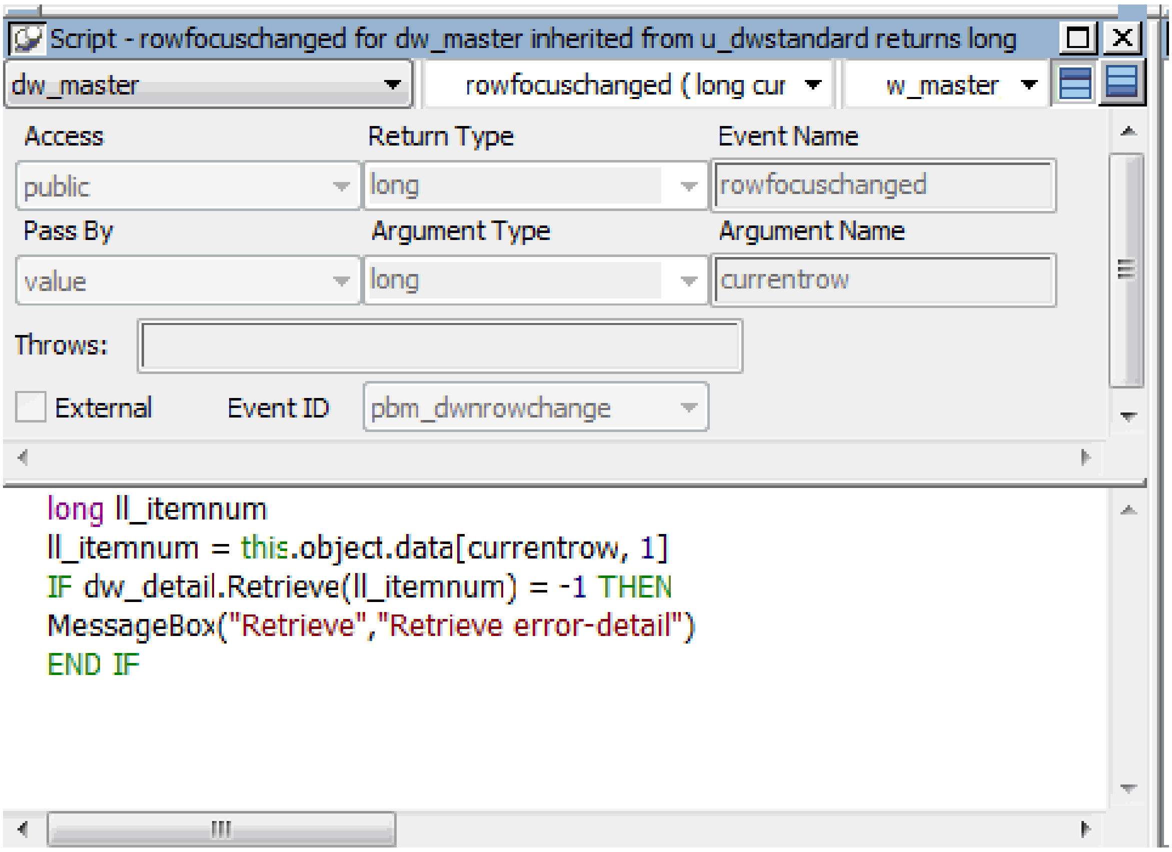
   This line uses a DataWindow data expression to obtain the item number in column 1 of the currently selected row of dw_master. It stores the number in the variable ll_itemnum.

   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.

7. Type these lines below the data expression line you just typed:

   ```
   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 (an ID number stored in the ll_itemnum variable). The IF statement that encloses the Retrieve function checks for successful completion. If the retrieval operation fails, it displays an error message box.



Script - rowfocuschanged for dw_master inherited from u_dwstandard returns long

dw_master | rowfocuschanged (long cur | w_master

Access: public
Return Type: long
Event Name: rowfocuschanged
Pass By: value
Argument Type: long
Argument Name: currentrow
Throws:
□ External
Event ID: pbm_dwnrowchange

```
long ll_itemnum
ll_itemnum = this.object.data[currentrow, 1]
IF dw_detail.Retrieve(ll_itemnum) = -1 THEN
MessageBox("Retrieve","Retrieve error-detail")
END IF
```

8. Click the Save button in PainterBar1.

9. Click the Close button in PainterBar1.

   PowerBuilder compiles the script you typed and saves it.

10.Click the Full Build Workspace button in the PowerBar.

   It is a good idea to rebuild all your objects after modifying an ancestor object.

11.Close the Output window.

2.5.9 Adjust a runtime setting for sheet window size

Where you are

Add a library to the search path

Create a new ancestor sheet window

Add user events and event scripts

Add scripts to retrieve data for the DataWindow controls

> Adjust a runtime setting for sheet window size

The Template Application wizard creates a sheet manager that makes the OpenSheet function call to open a sheet window. The OpenSheet function has an argument that can affect the sheet window size at runtime. By default the wizard sets this argument to the Cascaded! value that overrides the sheet window size you set at design time. Now you change this value to allow the runtime window size to be the same as the design time size.

1. Double-click n_pbtutor_sheetmanager in the System Tree

   or

   Right-click n_pbtutor_sheetmanager in the System Tree and select Edit from the pop-up menu.

2. In the Script view, select (Functions) in the first drop-down list box.

   Select of_opensheet in the second drop-down list box.

3. Go to the following line in the script:

   ```
   li_rc = OpenSheet ( lw_sheet, as_sheetname, w_pbtutor_frame, 0, Cascaded! )
   ```

4. Change the Cascaded! argument to Original!:

   ```
   li_rc = OpenSheet ( lw_sheet, as_sheetname, w_pbtutor_frame, 0, Original! )
   ```

5. Click the Save button in PainterBar1.

   Click the Close button in PainterBar1.

   The next time you run the tutorial application, the sheet windows will open in the size you set at design time. They will still be cascaded relative to other open sheets.

2.6 LESSON 6 Setting Up the Menus

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

- Modify the frame menu

- Create a new sheet menu

- Add menu scripts to trigger user events

- Attach the new menu and run the application

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 does it take?

About 30 minutes.

2.6.1 Modify the frame menu

Where you are

> Modify the frame menu

Create a new sheet menu

Add menu scripts to trigger user events

Attach the new menu and run the application

The frame menu was created automatically by the Template Application wizard. The m_pbtutor_frame menu is the ancestor of all the other menus you work with in the tutorial. Changes you make to this menu are automatically propagated to descendant menus.

In the WYSIWYG (What You See Is What You Get) view of the Menu painter, you see menus as they appear when the application is running. In this tutorial, you use the WYSIWYG view to make changes to the application menus, but you can make the same changes from the Tree Menu view. You use the Properties view to change a toolbar button.

In this exercise you:

- Modify the File menu
- Enable Help menu items

2.6.1.1 Modify the File menu

Now you modify the File cascading menu of the m_pbtutor_frame menu.

1. Double-click m_pbtutor_frame in the System Tree.

   The Menu painter displays the menu associated with the MDI frame window in the application. Because m_pbtutor_sheet inherits from m_pbtutor_frame, changes you make to the frame menu are propagated to the sheet menu.

   In the WYSIWIG view, the menu items appear across the top of the view. If a WYSIWYG view is not open, you can select it from the View menu in the Menu painter menu bar.

2. Click the File menu in the WYSIWYG view of m_pbtutor_frame.

   When you click a menu item in the WYSIWYG view, its menu items appear just as they would at runtime.

   If the File menu does not display its menu items

   The File menu is selected when you display the WYSIWYG view. You might need to click one of the other menus (Edit, Window, or Help) and then click again on the File menu to display its menu items.

3. Right-click New under the File menu.

   Select Edit Menu Item Text in the pop-up menu.

   Click in the New field and replace &New with &Report. Press Enter.

   You change the display name of the New menu item to Report. The menu name remains m_new and its purpose (to open new sheet windows) remains the same.

   Define accelerator keys with the ampersand character

   The character following the ampersand is used as an accelerator key (mnemonic) and appears with an underscore in the WYSIWYG display. In the runtime application, the user can access the File>Report menu by pressing Alt+F+R.

4. Make sure that &Report appears in the Text text box in the Properties view.

   Click the Toolbar tab in the Properties view.

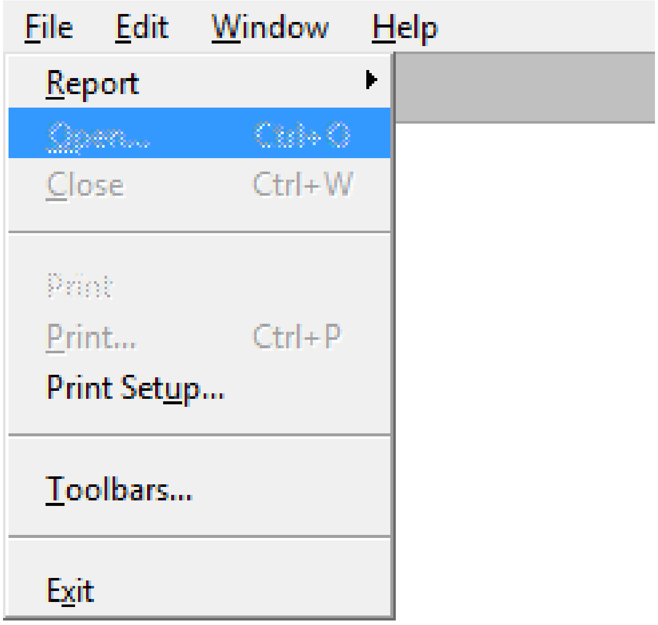
   The toolbar item text is New, Open New Sheet. There is no selection in the ToolbarItemName box, so no toolbar button appears at runtime for the Report menu item. You do not add a toolbar button here, because you use the Report menu item to access cascading menu items rather than as a command to open a new sheet.

5. Click the Open menu item under the File menu in the WYSIWYG view.

   Click the General tab in the Properties view.

   Clear the Visible and Enabled check boxes on the General page of the Properties view.

   You hide the Open menu item in all runtime menus. When you clear the Visible property, the WYSIWYG view displays the menu item with a dithered (broken) appearance. It is not visible at runtime. When you clear the Enabled property, the WYSIWIG view displays the menu item with a very faint and inverted relief appearance. If it is visible at runtime, the menu item will still be grayed out.



6. Click the Toolbar tab of the Properties view.

   Clear the ToolbarItemVisible check box.

   This prevents the toolbar button for this menu item from appearing in the frame menu toolbar (a button can be included in a toolbar even if its corresponding menu item is not visible or enabled).

7. Click Exit under the File menu in the WYSIWYG view.

   The Toolbar page of the Properties view remains open. The ToolbarItemText and ToolbarItemName values change to show the values for the m_exit menu item.

2.6.1.2 Enable Help menu items

The Help menu has three items, but only one, Help>About, is enabled. Now you enable the other menu items, using commented-out code that was provided by the Template Application wizard and the pbtutor.chm file in the Tutorial directory.

You call application Help topics with the ShowHelp function, passing it an enumerated value that identifies whether you want the Help contents or index to display, or a specific topic or keyword. ShowHelp can open Windows Help or compiled HTML Help (CHM) files.

1. Click the Help menu in the WYSIWYG view and then double-click the Help Index menu item.

   You can double-click the Help Index item even though it is not currently enabled. The full name of the Help Index menu item, m_help.m_helpindex, displays in the Script view. It includes the m_help prefix to indicate that it is in the Help menu.

2. Select Clicked in the second drop-down box if it is not already selected.

3. Position the cursor in the line that contains the ShowHelp function and click the Uncomment button in the PowerBar.

   Change myapp.hlp to pbtutor.chm:

   ```
    ShowHelp ("pbtutor.chm", Index!)
   ```

   This displays the default topic in the Help file.

4. Select the Enabled check box on the General page of the Properties view for the m_helpindex menu item.

5. Repeat the preceding steps for the Search For Help On menu item using the following Show Help function.

   ```
   ShowHelp ("pbtutor.chm", Keyword!, "")
   ```

   If the third argument contained a string that was a keyword in the Help file, the associated topic would display. Because the argument is an empty string, the Help Search window displays.

6. Select File>Save from the main PowerBuilder menu bar.

   Select File>Close from the main PowerBuilder menu bar.

2.6.2 Create a new sheet menu

Where you are

Modify the frame menu

> Create a new sheet menu

Add menu scripts to trigger user events

Attach the new menu and run the application

Now you create a new menu that displays whenever the user opens an MDI sheet to look at customer or product information. The menu you create is a descendant of the m_pbtutor_sheet menu that was generated by the Template Application wizard.

The m_pbtutor_sheet menu inherits in turn from m_pbtutor_frame, but has some additional menu items enabled. In the menu you create, you add menu items that are not present in the ancestor menus.

In this exercise you:

- Inherit and save a new menu
- Add items to the new menu
- Add a new toolbar for the new menu items

2.6.2.1 Inherit and save a new menu

By inheriting from the application sheet menu, you retain the sheet menu characteristics without modifying the ancestor menu. It is good practice to save the new menu immediately, then save it again after you modify it.

1. Click the Inherit button (Inherit) in the PowerBar.

   The Inherit From Object dialog box displays.

2. Make sure Menus is selected in the Objects of Type drop-down list box.

   Select m_pbtutor_sheet in the Object list box and click OK.

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

   On the inherited sheet menu, the Window menu items are enabled to allow for tiling and cascading windows, just as they are for the m_pbtutor_sheet menu. These items are disabled on the m_pbtutor_frame menu.

   Changes made to the MDI frame menu

   If you click the File menu in the WYSIWYG view, you see that the first item is now Report. The Open item is dithered to indicate that it is not visible and is grayed to indicate that it is disabled. These characteristics were propagated through the inheritance chain from m_pbtutor_frame.

3. Select File>Save from the menu bar.

   The Save Menu dialog box displays.

4. Type m_my_sheet as the menu name in the Menus box.

   Type the following line in the Comments box:

   ```
   New sheet menu for w_master_detail_ancestor and its descendants.
   ```

5. Click OK.

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

   The name you just assigned to the new menu displays in the title bar of the Menu painter workspace and the m_my_sheet menu appears in the system tree.

2.6.2.2 Add items to the new menu

Next you add items to the Edit menu of the menu you just inherited from m_pbtutor_sheet. You use the WYSIWYG and Properties views.

1. Click the Edit menu in the WYSIWYG view for the new menu.

   The list of Edit menu items appears just as it would in a runtime application. All items of the Edit menu are visible but disabled (they appear gray -- but not dithered -- in the WYSIWYG view).

2. Right-click a menu item under the Edit menu in the WYSIWYG view.

   Select Insert Menu Item At End from the pop-up menu.

   The cursor moves into a blank box that appears at the end of the Edit menu list.

3. Click in the box that appears at the bottom of the menu, and type - (hyphen). Press Enter.

   The hyphen changes into a separator line. In the Properties view, the menu item name changes to m_0. Even the separator lines between menu items must have unique names. Other separator lines in the menu have a unique index number preceded by the prefix m_dash.

4. Clear the Enabled check box in the Properties view for the new separator line.

5. Right-click the new separator line in the WYSIWYG view.

6. Select Insert Menu Item At End from the pop-up menu.

7. Type &Insert in the box that appears under the new separator and press Enter.

   The menu item name is set automatically to m_insert. If PowerBuilder displays a message that the default name is incorrect, it suggests an alternative name. If this occurs, click OK to accept the suggested name.

   Alternative method of inserting menu item names

   You can type &Insert in the Text box in the Properties view instead of typing it in the box that appears in the WYSIWYG view. In this case, you do not need to press Enter afterwards. First, however, you have to clear the Lock Name check box if the Name box is grayed (otherwise, the menu name does not reset to m_insert).

8. Type Insert a row in the MicroHelp box in the Properties view.

   The new menu item is visible and enabled by default.

   The text Insert a row displays in the MicroHelp line at the bottom of the application window whenever the user selects the menu item.

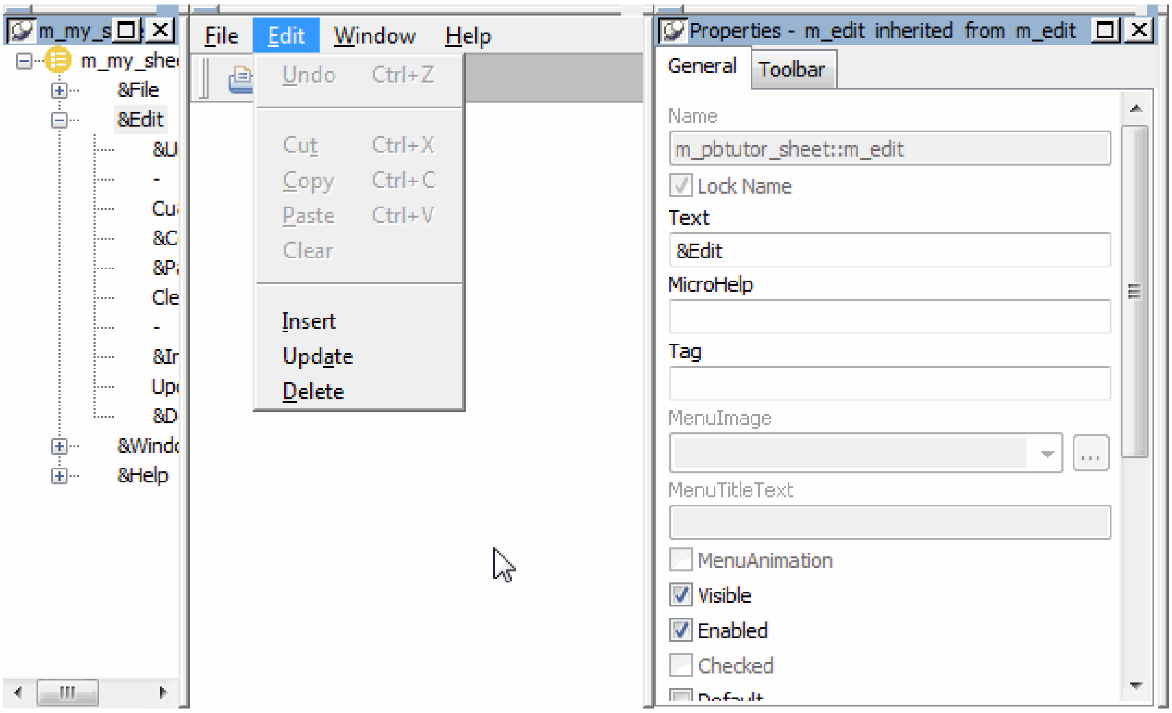
9. Repeat steps 5 and 6 for the following menu items:

   Table 2.8:

Menu item	MicroHelp text
Upd&ate	Update the database

Menu item	MicroHelp text
&Delete	Delete the current row

You add Edit menu items for updating and deleting database records. Even though it is not enabled, the Undo item already uses the letter U as an accelerator key, so you should not use the same accelerator key for the Update menu item. Instead, you use the letter A for this purpose.



Name: m_pbtutor_sheet::m_edit

☑ Lock Name

Text: &Edit

MicroHelp:

Tag:

MenuImage:

MenuTitleText:

□ MenuAnimation

☑ Visible

☑ Enabled

□ Checked

□ Default

2.6.2.3 Add a new toolbar for the new menu items

Now you add toolbar buttons for the menu items you just defined and then place them in a second toolbar.

1. Click the new Insert menu item in the WYSIWYG view Edit menu.

2. Click the Toolbar tab in the Properties view.

   Type Insert in the ToolbarItemText box.

   Type or select Insert! in the ToolbarItemName drop-down list.

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

3. Type or click to 1 in the ToolbarItemSpace spin control.

   Type or click to 1 in the ToolbarItemOrder spin control.

   Type or click to 2 in the ToolbarItemBarIndex spin control.

   When you start a new toolbar for the added menu items, the Insert button will be the first item in this toolbar.

4. Click the new Update menu item in the WYSIWYG view Edit menu.

   Make sure it is also displayed on the title bar in the Properties view.

5. Click the Toolbar tab if the Toolbar page is not already open.

   Type Update in the ToolbarItemText box.

   Type or select Update! in the ToolbarItemName drop-down list box.

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

6. Type or click to 2 in the ToolbarItemOrder spin control.

   Type or click to 2 in the ToolbarItemBarIndex spin control.

   This will add the Update button after the Insert button in the new toolbar.

7. Click the new Delete menu item in the WYSIWYG view Edit menu.

   Make sure it is also displayed on the title bar in the Properties view.

8. Click the Toolbar tab if the Toolbar page is not already open.

   Type Delete in the ToolbarItemText box.

   Type or select DeleteRow! in the ToolbarItemName drop-down list box.

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

9. Type or click to 3 in the ToolbarItemOrder spin control.

   Type or click to 2 in the ToolbarItemBarIndex spin control.

   You add the Delete button after the Update button in the new toolbar.

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

2.6.3 Add menu scripts to trigger user events

Where you are

Modify the frame menu

Create a new sheet menu

> Add menu scripts to trigger user events

Attach the new menu and run the application

Now you add scripts to trigger user events from the sheet window menu bar. You added these user events in Modifying the Ancestor Window. The Menu painter should still be open for the m_my_sheet menu. If it is not, you can open it using the Open button in the PowerBar.

1. Select m_edit.m_insert in the first list box in the Script view

   or

Double-click the Insert menu item in the WYSIWYG view.

The full name of the Insert menu item displays in the first list box of the Script view. It includes the m_edit prefix to indicate that it is in the Edit menu.

2. Select Clicked in the second drop-down box if it is not already selected.

   Type these lines for the Clicked event:

```
w_master_detail_ancestor lw_activesheet
lw_activesheet = w_pbtutor_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.

3. Repeat steps 1 and 2 for the following menu items and scripts:

Table 2.9:

Menu name	Script for Clicked event

m_edit.m_update	`w_master_detail_ancestor lw_activesheet`
	`lw_activesheet=w_pbtutor_frame.GetActiveSheet()`
	`lw_activesheet.EVENT ue_update()`

m_edit.m_delete	`w_master_detail_ancestor lw_activesheet`
	`lw_activesheet = w_pbtutor_frame.GetActiveSheet()`
	`lw_activesheet.EVENT ue_delete()`

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

   PowerBuilder compiles and saves the menu scripts.

5. Click the Close button in PainterBar1

   or

   Select File>Close from the PowerBuilder menu bar.

2.6.4 Attach the new menu and run the application

Where you are

Modify the frame menu

Create a new sheet menu

Add menu scripts to trigger user events

> Attach the new menu and run the application

Now you attach the new sheet menu and run the application again.

1. Double-click w_master_detail_ancestor in the System Tree.

If you cannot see the Properties view

Select View>Properties from the menu bar.

The menu listed in the MenuName box in the Properties view of the Window painter is still m_pbtutor_sheet.

2. Click the ellipsis button next to the MenuName box.

   The Select Object dialog box displays.

3. Select m_my_sheet in the Menus list box and click OK.

   This is the sheet menu you modified after inheriting it from m_pbtutor_sheet. It is now listed as the menu name in the Properties view.

4. Click the Save button in PainterBar1.

   Click the Run button in the PowerBar.

   The application login window displays.

5. Type dba in the User ID box.

   Type sql in the Password box and click OK.

   The database connection is established, and the MDI frame for the application displays. The File menu now includes a Report cascading menu in place of the New menu item. The Open menu item is no longer visible.

6. Select File>Report>Maintain Customers from the menu bar.

   Notice that a second toolbar appears and the Edit and Window cascading menus include enabled menu items.

7. 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.

8. Select the Window menu.

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

9. Select File>Report>Maintain 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 is because m_my_sheet is the menu for both w_customers and w_products.

10. Select the Edit menu.

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

11. Select the Window menu.

    Another menu item has been added for the second sheet you opened. The checkmark next to this menu item indicates that it is the active sheet.

12.Select File>Exit from the menu bar.

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

13.Close the Window painter.

2.7 LESSON 7 Building DataWindow Objects

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 lesson you:

- Create and preview a new DataWindow object
- Save the DataWindow object
- Make cosmetic changes to the first DataWindow object
- Create a second DataWindow object
- Make cosmetic changes to the second DataWindow object

How long does it take?

About 20 minutes.

2.7.1 Create and preview a new DataWindow object

Where you are

> Create and preview a new DataWindow object

Save the DataWindow object

Make cosmetic changes to the first DataWindow object

Create a second DataWindow object

Make cosmetic changes to the second DataWindow object

Now you create a new DataWindow object and display it in the DataWindow painter. Like other painters, the DataWindow painter has an assortment of views that you can open simultaneously.

About the Design view of the DataWindow painter

The Design view in the DataWindow painter is similar to the Layout view in other painters. You can open only one Design view at a time.

The Design view is divided into four areas called bands: header, detail, summary, and footer. You can modify the contents of these bands. For example, you can change their sizes, add objects (controls, text, lines, boxes, or ovals), and change colors and fonts.

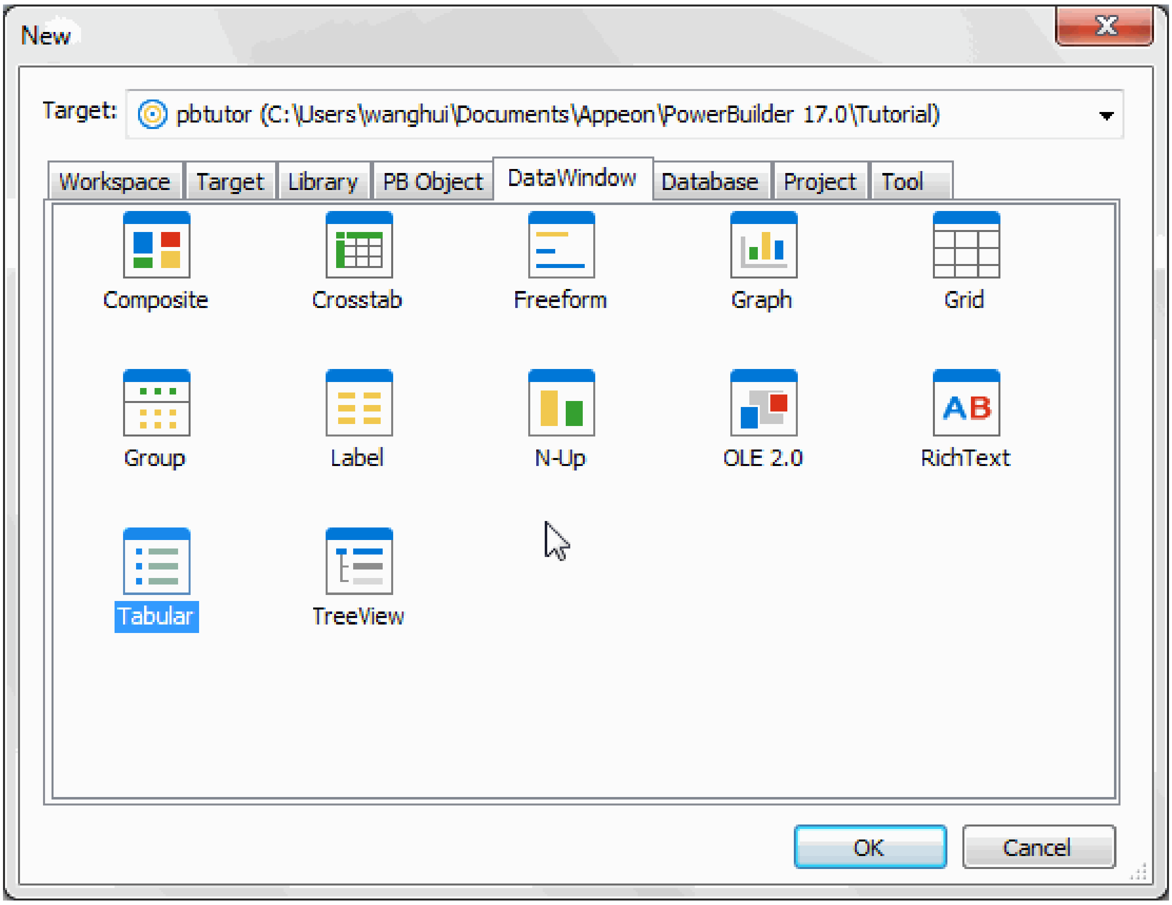
In the Preview view of the DataWindow painter, you can see how the object looks in an application at runtime, complete with table data.

1. Click the New button (New) in the PowerBar.

   The New dialog box displays.

2. Click the DataWindow tab.

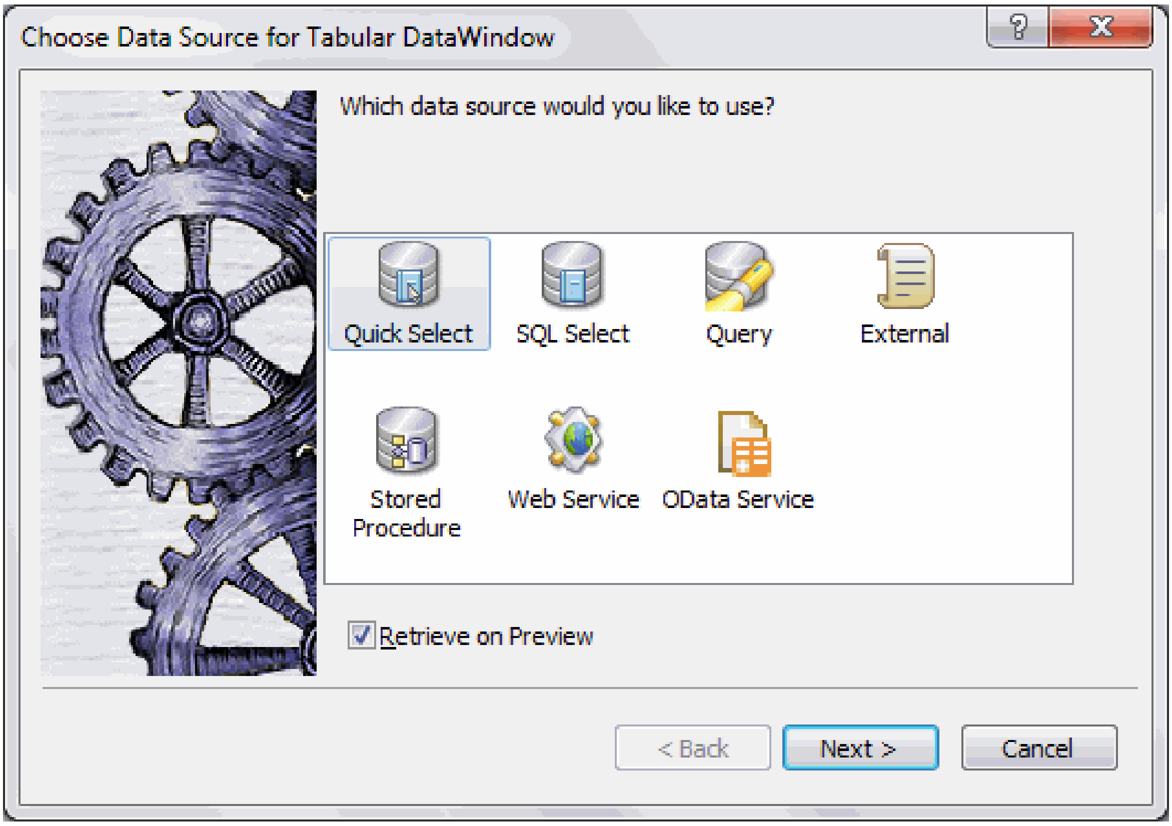
3. Select Tabular from the list of presentation styles.



Target: pbtutor (C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial)

4. Click OK.

   The Choose Data Source for Tabular DataWindow page of the DataWindow wizard displays.



Which data source would you like to use?

☑ Retrieve on Preview

5. Select Quick Select as the data source and select the Retrieve On Preview check box if it is not already selected.

   Click Next.

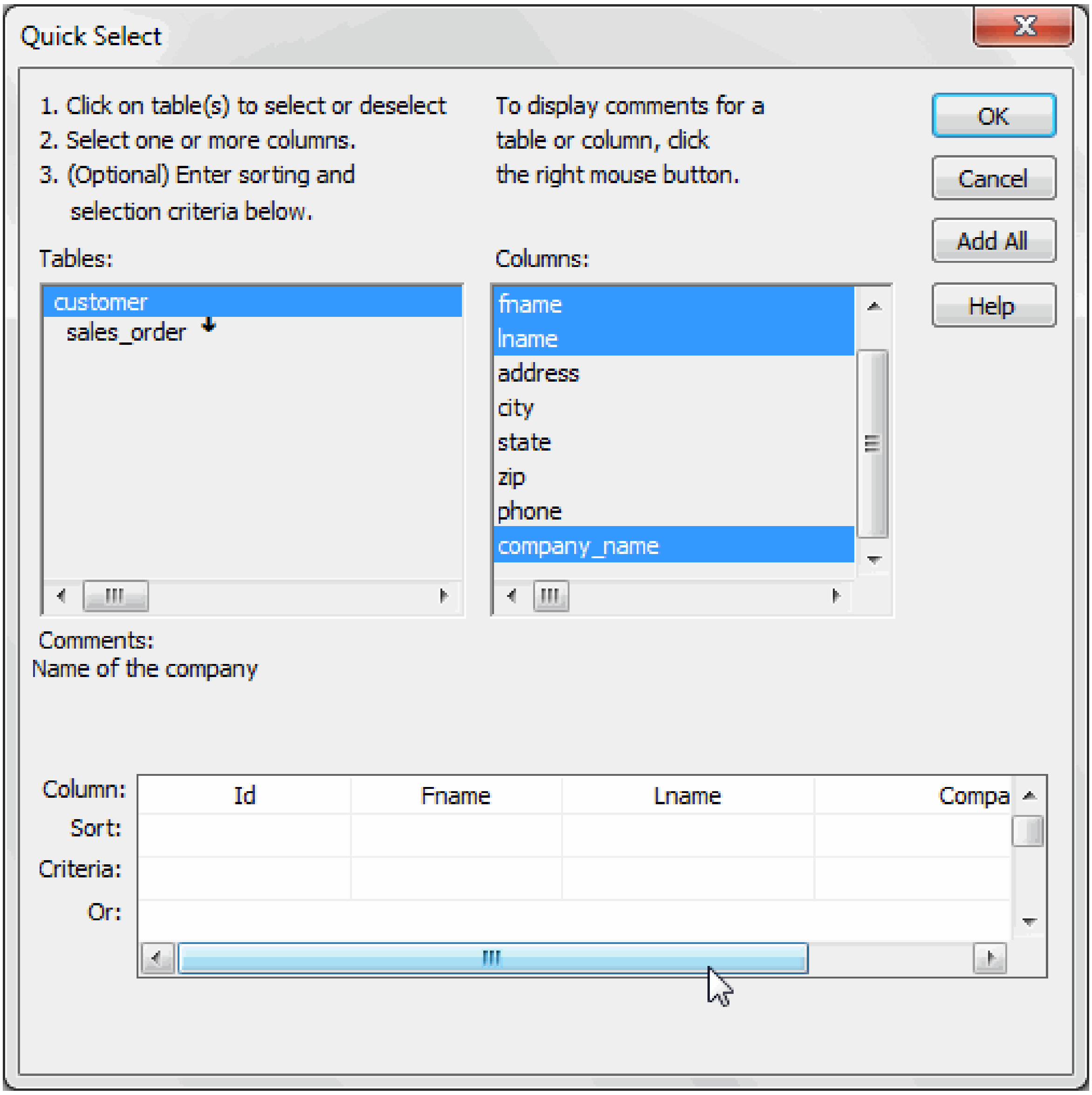
   PowerBuilder connects to the Demo Database, and the Quick Select dialog box displays.

6. Click the customer table in the Tables list box.

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

7. Click id, fname, and lname in the Columns list box in the order listed.

   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 their left-to-right display order in the DataWindow object. If you clicked a column by mistake, you can click it again to clear the selection.

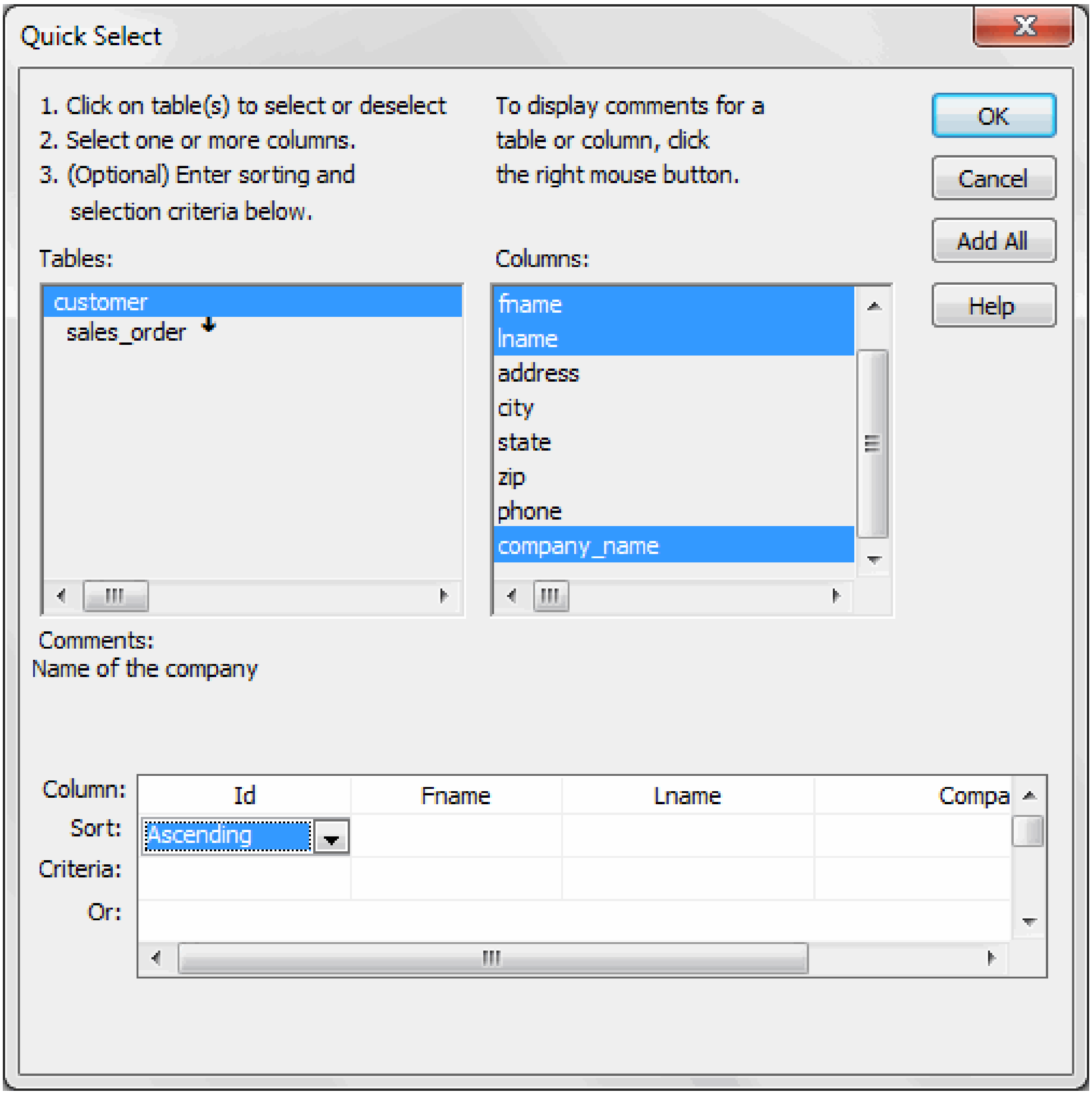
You can use the grid area at the bottom of the dialog box to specify sort criteria (for the SQL ORDER BY clause) and selection criteria (for the SQL WHERE clause). Now you specify sort criteria only. You sort the id column in ascending order.

8. In the grid area of the Quick Select dialog box, click in the cell next to Sort and below Id.

   A drop-down list box displays.

9. Choose Ascending from the drop-down list box.

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



Quick Select

1. Click on table(s) to select or deselect
2. Select one or more columns.
3. (Optional) Enter sorting and selection criteria below.

To display comments for a table or column, click the right mouse button.

Tables:

customer
sales_order

Columns:

fname
lname
address
city
state
zip
phone
company_name

Comments:
Name of the company

Column:	Id	Fname	Lname	Compa
Sort:	Ascending			
Criteria:				
Or:				

10.Click OK.

The DataWindow wizard asks you to select the colors and borders for the DataWindow object. By default, there are no borders for text or columns.

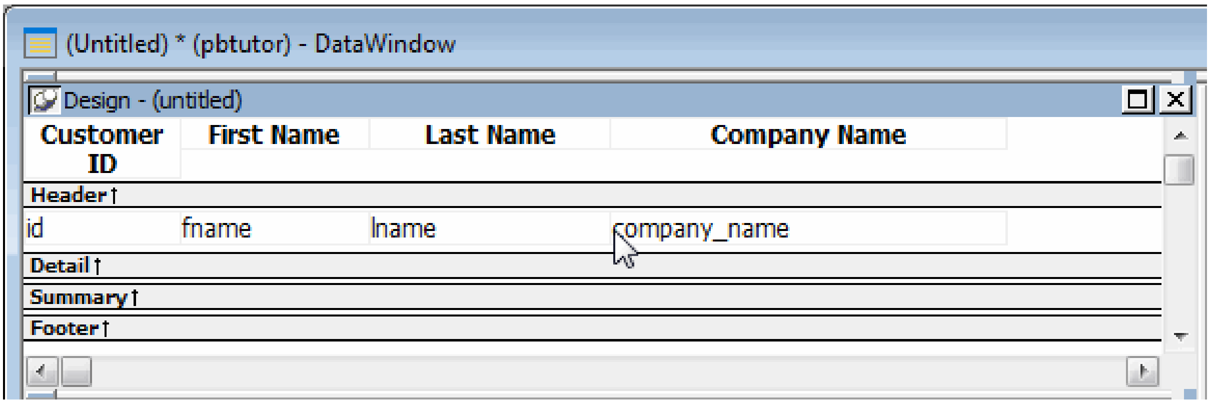
11.Click Next.

You accept the border and color defaults. The DataWindow wizard summarizes your selections.

12.Click Finish.

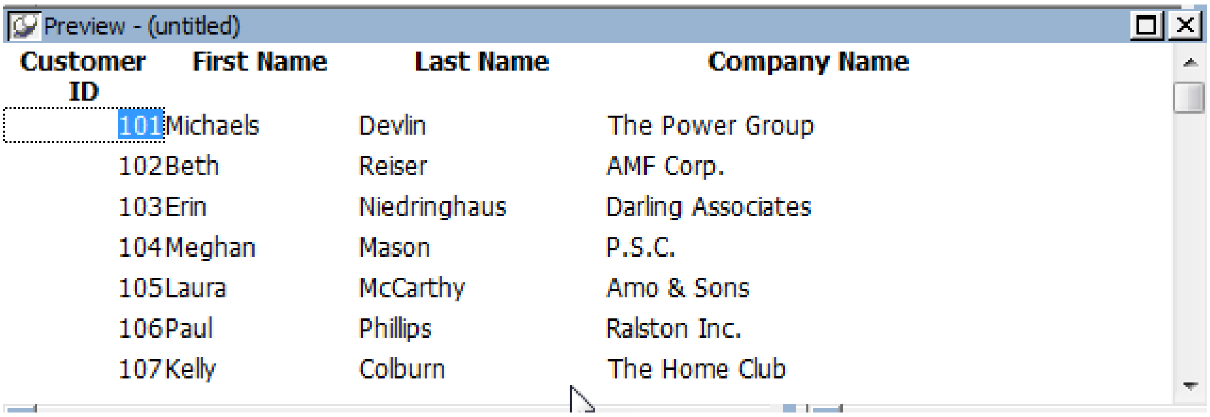
PowerBuilder creates the new DataWindow object and opens the DataWindow painter.

In the Design view, PowerBuilder displays a Header band with default headings and a Detail band with the columns you selected:



Customer ID	First Name	Last Name	Company Name
id	fname	lname	company_name

The Preview view displays the DataWindow as it appears during execution. PowerBuilder displays data for all customers. The data is sorted in ascending order by customer ID, just as you specified.



Customer ID	First Name	Last Name	Company Name
101	Michaels	Devlin	The Power Group
102	Beth	Reiser	AMF Corp.
103	Erin	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

Displaying the Preview view

If the Preview view is not displayed, select View>Preview from the menu bar. If Preview is grayed, it is already displayed and you cannot select it. You can open only one Preview view at a time.

2.7.2 Save the DataWindow object

Where you are

Create and preview a new DataWindow object

> Save the DataWindow object

Make cosmetic changes to the first DataWindow object

Create a second DataWindow object

Make cosmetic changes to the second DataWindow object

Now you name the DataWindow object and save it in the pbtutor.pbl library.

Saving to another library

You can save objects to different application libraries, but to avoid complications, you save all your new tutorial objects in one library. You can also copy or move objects from one library to another using the Library painter.

1. Select File>Save from the menu bar.

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

2. Make sure pbtutor.pbl is selected in the Application Libraries box.

   Type d_custlist in the DataWindows box.

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

3. (Optional) Type the following comments in the Comments box:

   ```
   This DataWindow object retrieves customer names and company associations.
   ```

4. Click OK.

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

2.7.3 Make cosmetic changes to the first DataWindow object

Where you are

Create and preview a new DataWindow object

Save the DataWindow object

> Make cosmetic changes to the first DataWindow object

Create a second DataWindow object

Make cosmetic changes to the second DataWindow object

Now you can make cosmetic changes to the DataWindow. You 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 move some of the columns to make them line up with their headings.

You make these changes in the Design view. You can keep the Preview view open at the same time to see how the changes you make affect the appearance of the DataWindow at runtime.

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

   or

   Press Ctrl+A.

   All of the controls in the DataWindow object are selected in the Design view.

2. Position the mouse pointer over one of the selected objects.

   Drag the object to the right about one inch.

All of the selected objects move together.

3. Click in a blank area in the Design view.

   You clear the object selection.

4. Click the Customer ID header above the Header band.

   Hold down the Ctrl key and click the id column above the Detail band.

   Release the Ctrl key and drag the id column to the left about one-half inch.

   The column and its header move together.

5. Click the Center button (Center) in the StyleBar.

6. Click in a blank area in the Design view.

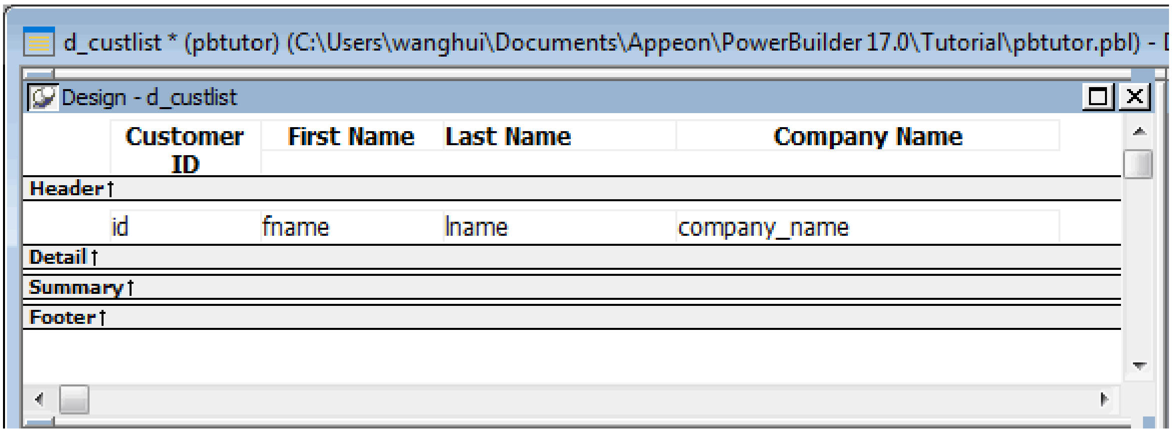
   This centers the Customer ID column header text and the column data.

7. Click the First Name header.

8. Hold down the Ctrl key and click the Last Name and Company Name headers.

9. Click the Left button (Left) in the StyleBar.

   When you have finished, the Design view should look something like this:



10.Select File>Close from the menu bar.

   A message box asks if you want to save your changes.

11.Click Yes.

   PowerBuilder saves the DataWindow object and closes the DataWindow painter.

2.7.4 Create a second DataWindow object

Where you are

Create and preview a new DataWindow object

Save the DataWindow object

Make cosmetic changes to the first DataWindow object

> Create a second DataWindow object

Make cosmetic changes to the second 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.

To build the second DataWindow object, you 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 pass the customer ID.

In this section, you:

- Select the data source and style
- Select the table and columns
- Define a retrieval argument
- Specify a WHERE clause
- View the DataWindow in the DataWindow painter
- Save the DataWindow object

2.7.4.1 Select the data source and style

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

1. Click the New button (New) in the PowerBar.

   The New dialog box appears.

2. Click the DataWindow tab if it is not already selected.

   Select Freeform from the list of presentation styles and click OK.

3. Select SQL Select as the data source and select the Retrieve On Preview check box if it is not already selected.

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

   Selecting the Retrieve On Preview check box allows you to view the data returned by a query in the development environment, but you need to provide initial values for any retrieval arguments that you specify.

4. Click Next.

2.7.4.2 Select the table and columns

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

1. Select customer in the list of tables and click Open.

   The Select painter displays the customer table and its columns.

Alternative method

If you double-click the customer table instead of selecting it and clicking Open, the Select Tables dialog box remains open. In this case, you must click Cancel to continue.

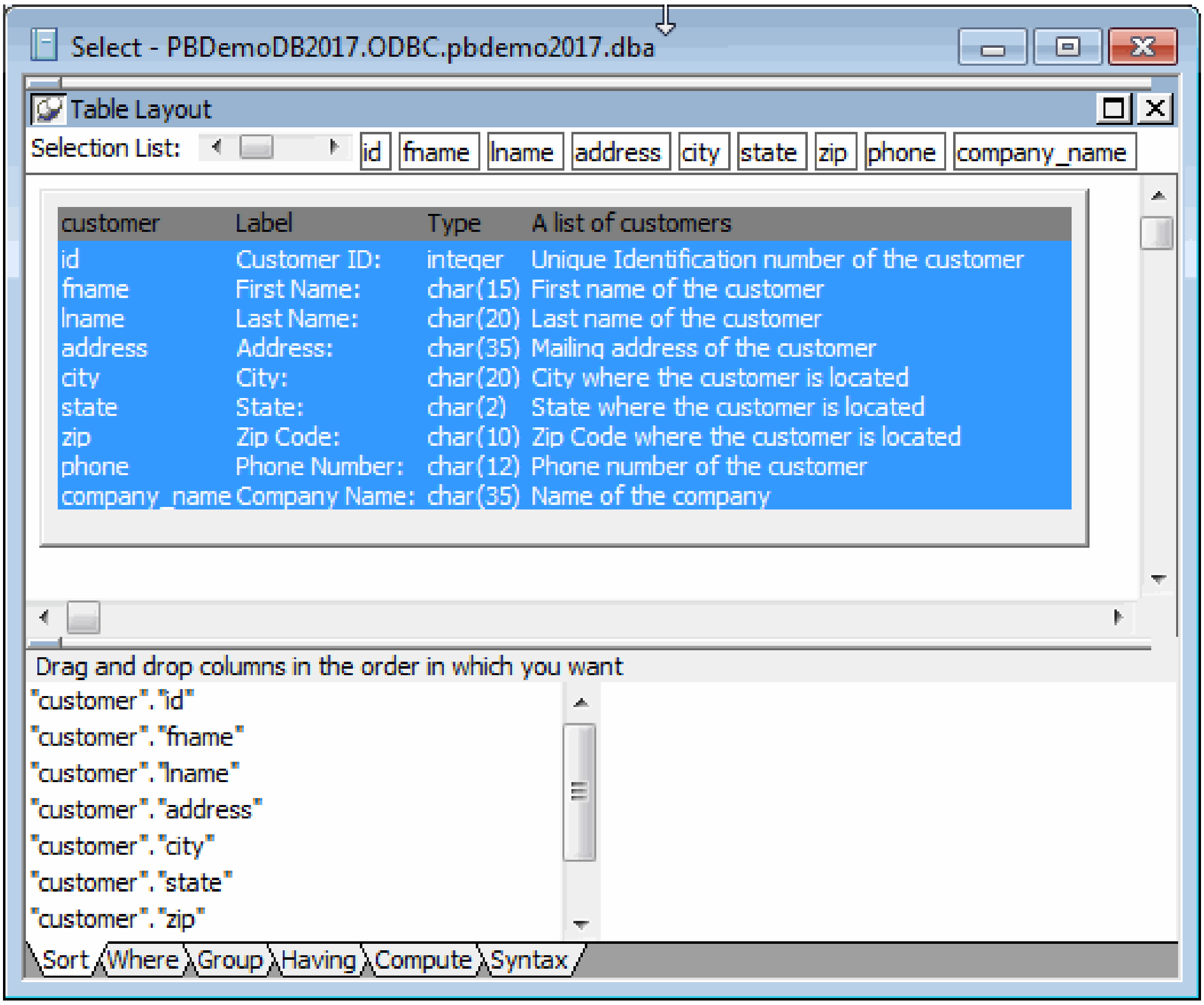
2. Right-click the header area in the Table Layout view.

   Choose Select All from the pop-up menu.

   The column names appear in the Selection List area above the table.

   The columns appear in the order in which you selected them. Because you selected all the columns at once, the original order of the columns in the database is used. You change the column presentation order later.

You can also see the selection order in the Syntax view: click the Syntax tab at the bottom of the stack of tabbed panes to display the generated Select statement.



customer	Label	Type	A list of customers
id	Customer ID:	integer	Unique Identification number of the customer
fname	First Name:	char(15)	First name of the customer
lname	Last Name:	char(20)	Last name of the customer
address	Address:	char(35)	Mailing address of the customer
city	City:	char(20)	City where the customer is located
state	State:	char(2)	State where the customer is located
zip	Zip Code:	char(10)	Zip Code where the customer is located
phone	Phone Number:	char(12)	Phone number of the customer
company_name	Company Name:	char(35)	Name of the company

Drag and drop columns in the order in which you want

```
"customer"."id"
"customer"."fname"
"customer"."lname"
"customer"."address"
"customer"."city"
"customer"."state"
"customer"."zip"
```

2.7.4.3 Define a retrieval argument

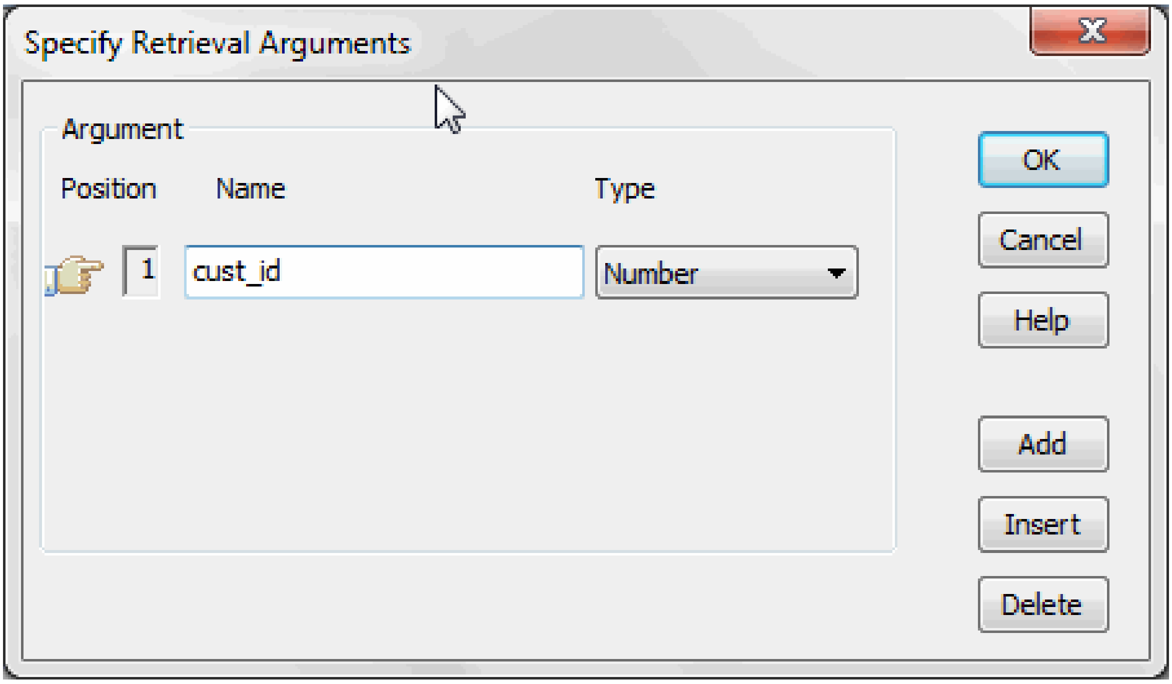
Now you define a retrieval argument.

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 pass during execution. Nonetheless, it is a good idea to make the name meaningful.

3. Click OK.

   The retrieval argument is defined.

2.7.4.4 Specify a WHERE clause

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

1. Click the Where tab in the stack.

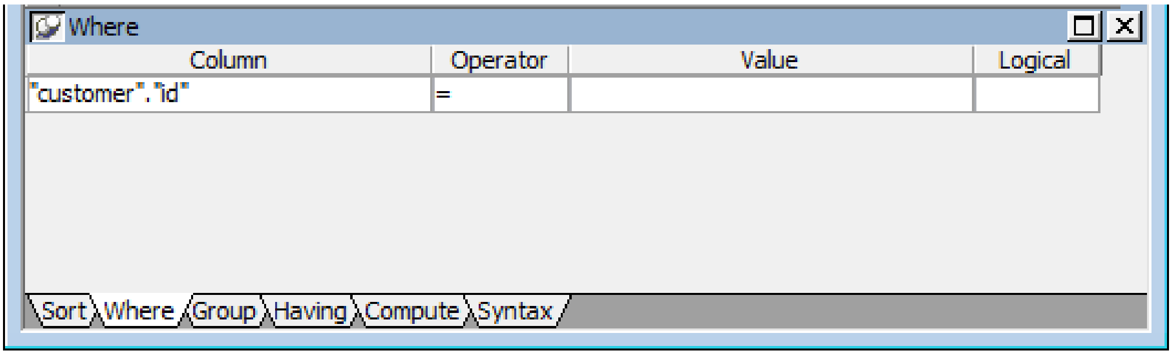
   The Where view displays.

2. Click in the box below Column in the Where view.

   A down arrow displays, and the box becomes a drop-down list box.

3. Select "customer"."id".

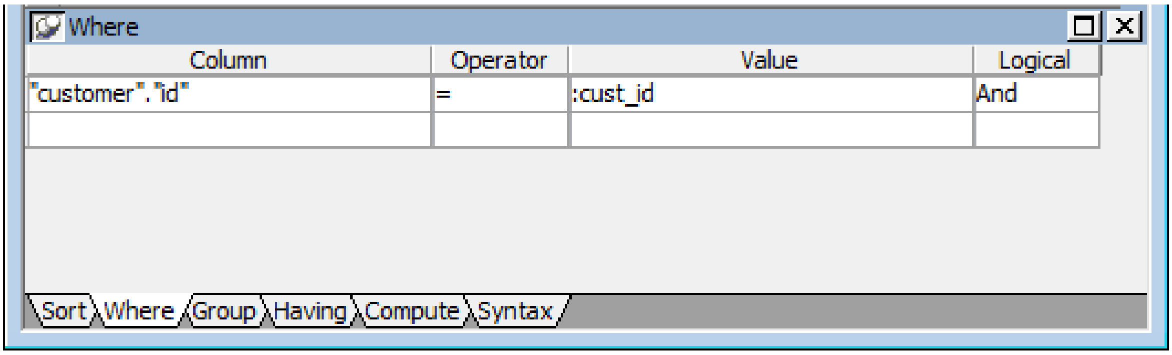
   Your selection displays immediately below the Column heading. An equal sign (=) appears in the Operator box. This is correct, so do not change it.



Column	Operator	Value	Logical
"customer"."id"	=		

4. Right-click in the box below the Value column header in the Where view.

   Select Arguments from the pop-up menu, select :cust_id, and click Paste.



Column	Operator	Value	Logical
"customer"."id"	=	:cust_id	And

5. Click the Syntax tab in the stack.

   The Syntax view displays the modified SELECT statement.

6. Scroll down until you see the generated WHERE clause.

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

2.7.4.5 View the DataWindow in the DataWindow painter

Now you view the DataWindow in the DataWindow painter using the Design and Preview views.

1. Click the Return button in the PainterBar

   or

   Select File>Return To DataWindow Painter from the menu bar.

   The DataWindow wizard asks you to select the borders and colors for the new DataWindow object.

2. Select Raised from the Border drop-down list box for columns.

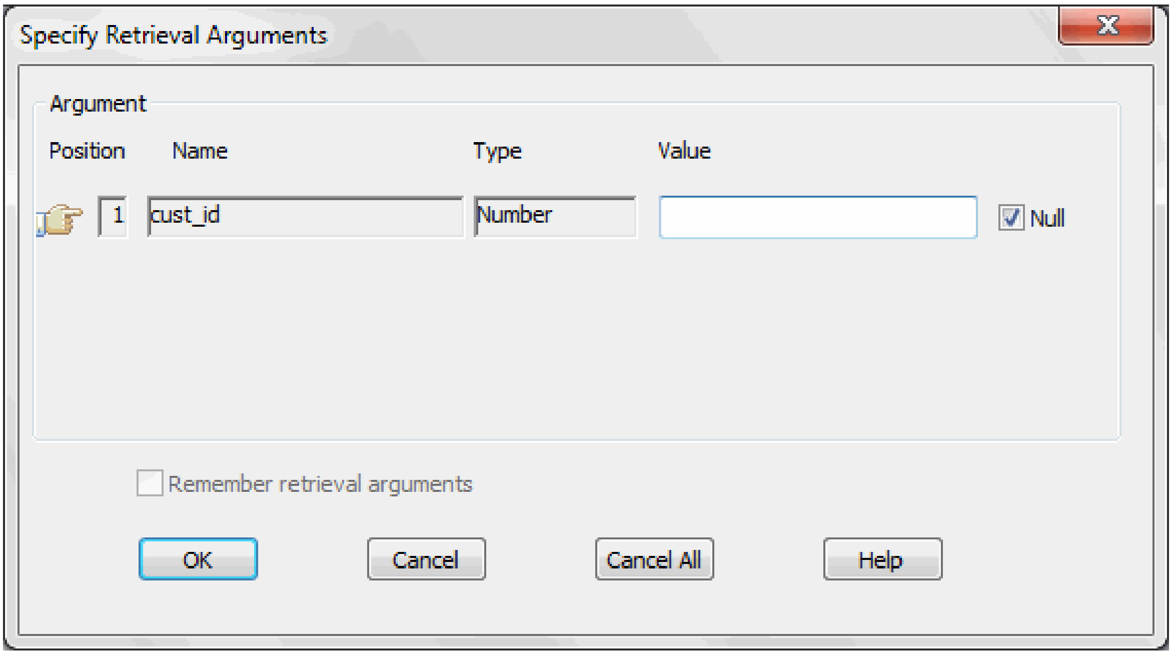
   Click Next.

   You have added raised borders to the DataWindow columns, but not to the labels. The DataWindow wizard summarizes your selections.

3. Click Finish.

   Because you selected the Retrieve On Preview check box and because the Preview view is part of the default layout scheme for the DataWindow painter, the Specify Retrieval Arguments dialog box appears.

   This dialog box prompts you for an argument value. When you put this DataWindow object into the tutorial application, you write a script that passes the required argument to the DataWindow object automatically.



Specify Retrieval Arguments

Argument

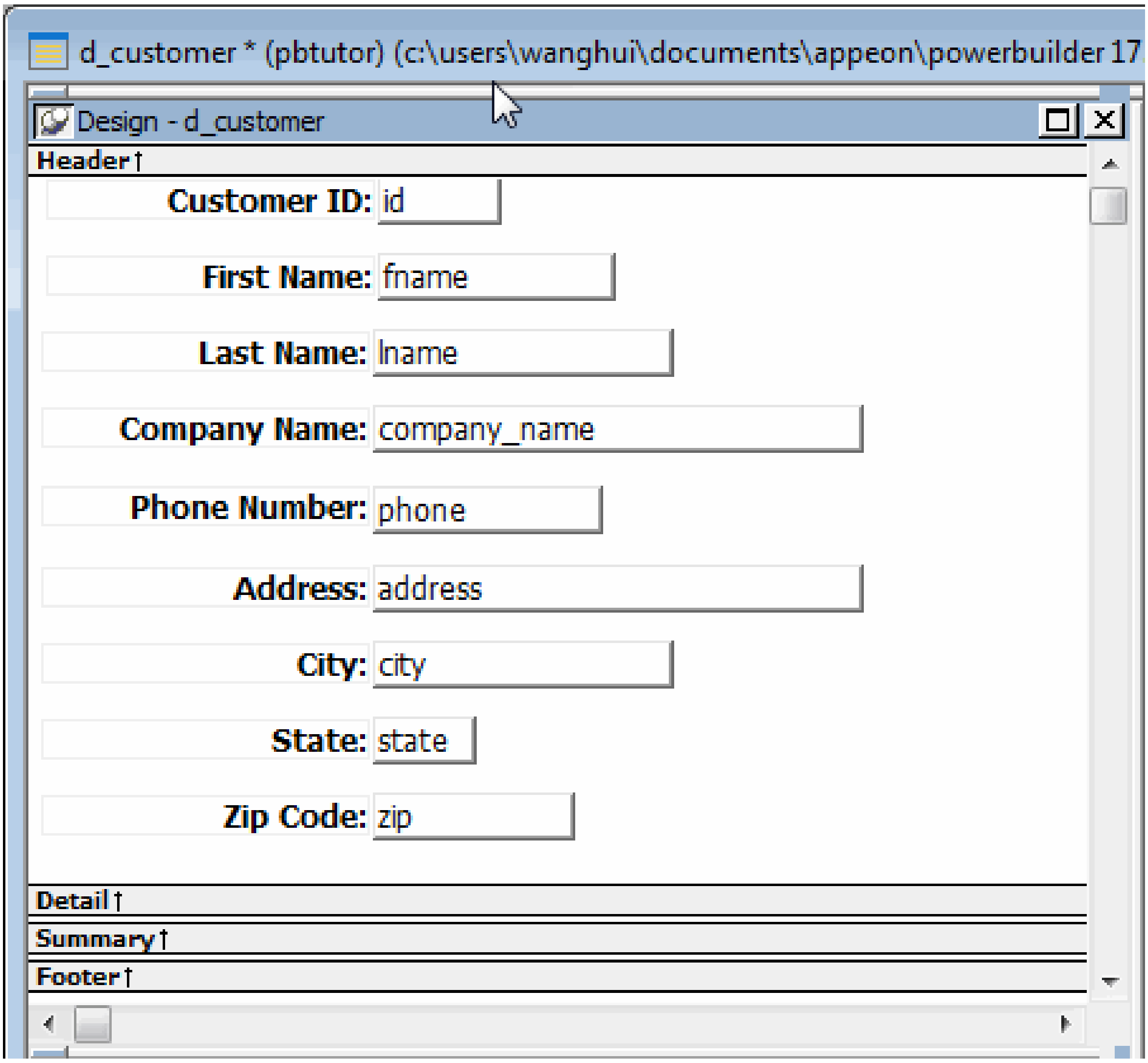
Position	Name	Type	Value	
1	cust_id	Number		☑ Null

□ Remember retrieval arguments

4. Type a customer ID (such as 101, 102, or 103) in the Value field.

   Click OK.

   The DataWindow painter opens. The Design view displays the new DataWindow object.



d_customer * (pbtutor) (c:\users\wanghui\documents\appeon\powerbuilder 17

Design - d_customer

Header↑

Customer ID: id

First Name: fname

Last Name: lname

Company Name: company_name

Phone Number: phone

Address: address

City: city

State: state

Zip Code: zip

Detail↑

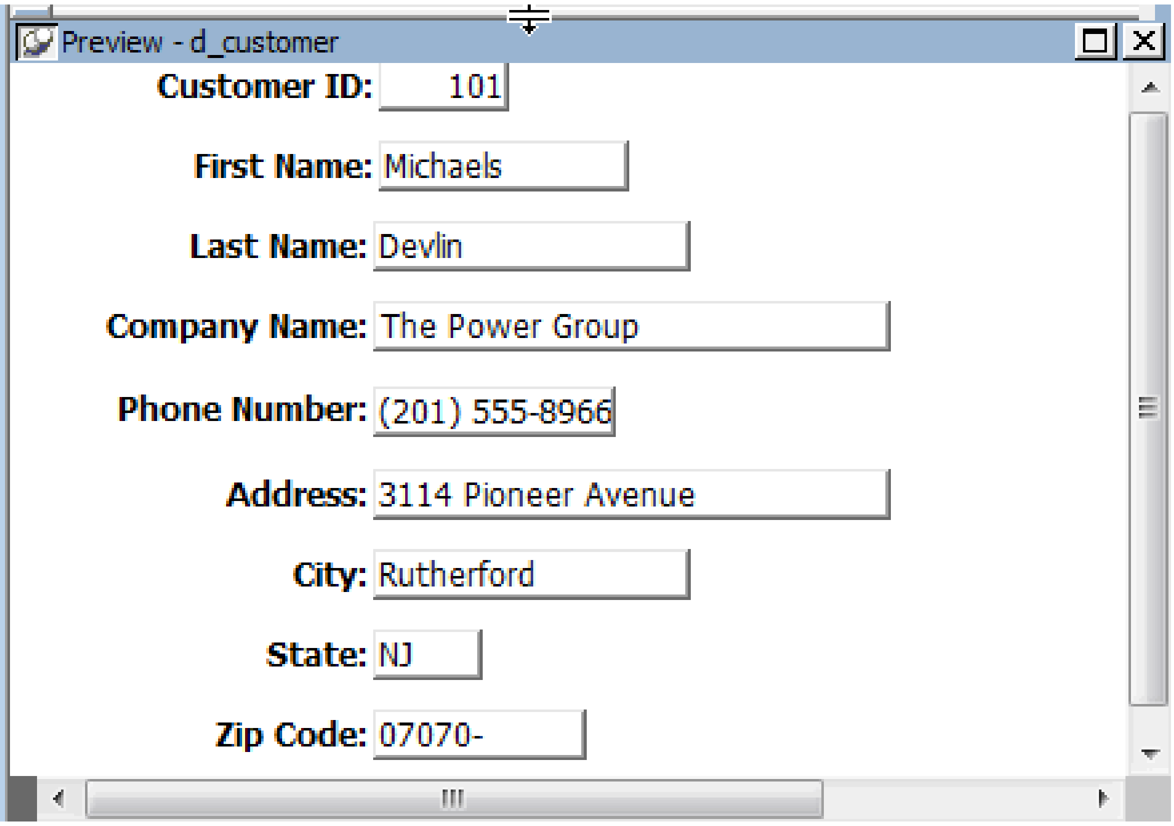
Summary↑

Footer↑

Changing font sizes

If you cannot see all letters in a label, press Ctrl+A to select all the items in the DataWindow, then select a smaller font size in the StyleBar.

The DataWindow Preview view retrieves the requested customer data.



Preview - d_customer

Customer ID: 101

First Name: Michaels

Last Name: Devlin

Company Name: The Power Group

Phone Number: (201) 555-8966

Address: 3114 Pioneer Avenue

City: Rutherford

State: NJ

Zip Code: 07070-

Retrieving other records

If you want to preview the record for another customer, you can right-click inside the DataWindow Preview view, select Retrieve from the pop-up menu, then specify a different customer ID in the Specify Retrieval Arguments dialog box.

2.7.4.6 Save the DataWindow object

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

1. Select File>Save from the menu bar.

   The Save DataWindow dialog box displays.

2. Make sure pbtutor.pbl is selected in the Application Libraries box.

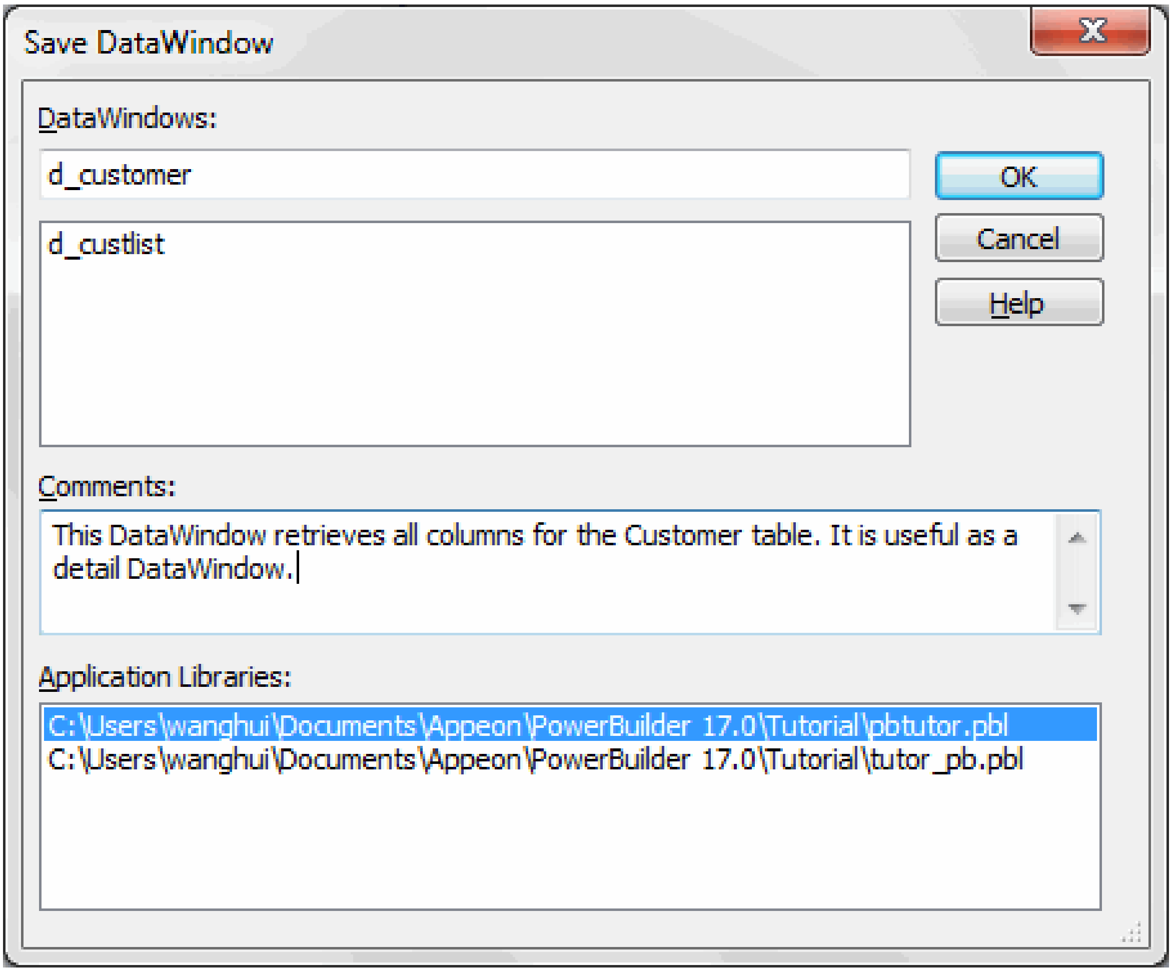
   Type d_customer in the DataWindows box.

   Earlier you saved a DataWindow object as d_custlist.

3. (Optional) Type the following comments in the Comments box.

   ```
   This DataWindow retrieves all columns for the Customer table. It is useful as a
    detail DataWindow.
   ```

4. Click OK.



DataWindows: d_customer

d_custlist

Comments: This DataWindow retrieves all columns for the Customer table. It is useful as a detail DataWindow.

Application Libraries:

C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial\pbtutor.pbl
C:\Users\wanghui\Documents\Appeon\PowerBuilder 17.0\Tutorial\tutor_pb.pbl

You return to the DataWindow painter.

2.7.5 Make cosmetic changes to the second DataWindow object

Where you are

Create and preview a new DataWindow object

Save the DataWindow object

Make cosmetic changes to the first DataWindow object

Create a second DataWindow object

> Make cosmetic changes to the second DataWindow object

Now you modify the DataWindow object. You:

- Rearrange the columns and labels
- Align the columns and labels
- Display the arrow for a drop-down DataWindow edit style

Columns on freeform DataWindows

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

2.7.5.1 Rearrange the columns and labels

Now you rearrange the columns and labels in the new DataWindow object. You can maximize the Design view for greater ease in manipulating the columns and their labels.

1. Click the Address: label in the Design view.

   Hold the Ctrl key and click the address column.

   The two items are selected.

2. Keep the Ctrl key pressed and click the following column labels and column controls:

   Table 2.10:

Label	Column
City:	city
State:	state
Zip Code:	zip

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

4. Release the Ctrl key.

5. Position the cursor on one of the selected objects and drag it to the top-right corner of the DataWindow object.

   The objects move together.

6. Use the Ctrl+click technique to move the following label and column controls to the location indicated:

   Table 2.11:

Label	Move with column	Move under
Company Name:	company_name	Last Name:
Phone Number:	phone	Company Name:

7. Drag the Detail band up below the last column label.

   This removes any extra space at the bottom of the detail area.

Some of the fields might be misaligned. You fix this in the next exercise.

2.7.5.2 Align the columns and labels

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

1. Select the Zip Code: label in the Design view.

   Move left edge of the Zip Code: label's text close to the right edge of the company_name column.

2. While the Zip Code: label is still selected, use the Ctrl+click technique to select the Address:, City:, and State: labels.

3. Select Format>Align from the menu bar.

   A cascading menu of align options displays.

4. Select the first option (Align left) (Left).

   PowerBuilder aligns the left edges of the selected objects with the left edge of the first item you selected (the Zip Code: label).

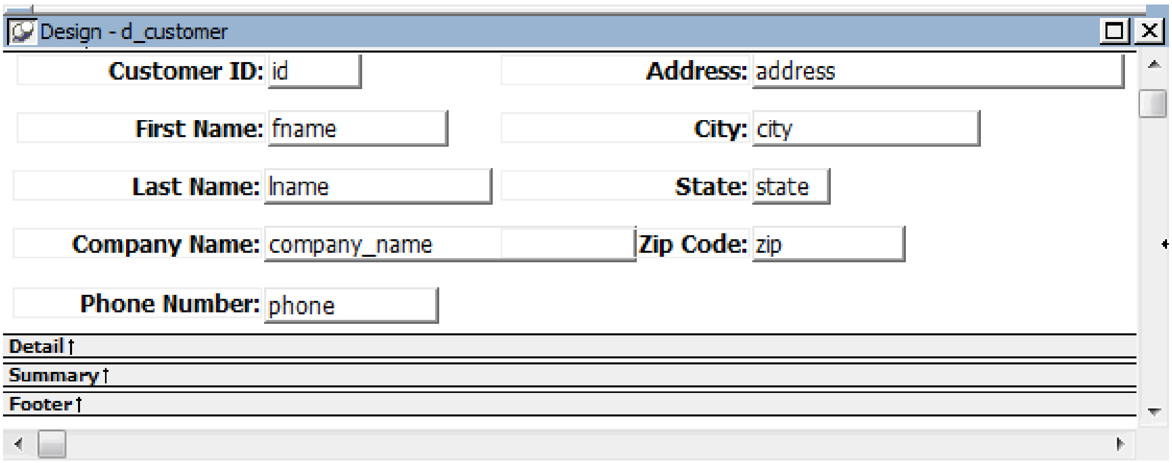
Selecting an alignment tool from the PainterBar

You can access a drop-down list of alignment tools by clicking the Align button on PainterBar2.

5. Move the zip column so that it is next to the Zip Code: label.

6. Align the address, city, and state columns with the zip column, just as you aligned the column labels.

   The DataWindow should now look like this in the Design view:



Customer ID: id

First Name: fname

Last Name: lname

Company Name: company_name

Phone Number: phone

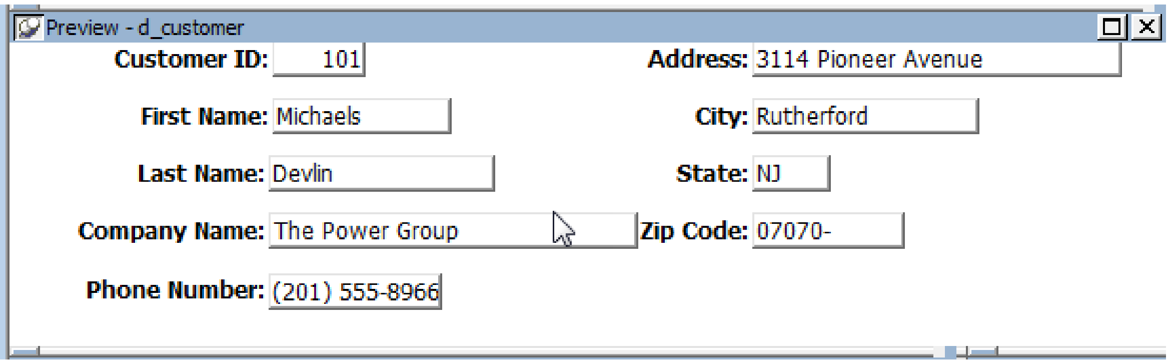
Address: address

City: city

State: state

Zip Code: zip

The DataWindow Preview view looks like this:



Preview - d_customer

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

2.7.5.3 Display the arrow for a drop-down DataWindow edit style

The column for the customer state of residence has a DropdownDataWindow edit style. This is an extended attribute associated with the State column in the Demo Database. The (drop-down) DataWindow with which the column is associated has a list of states and their two-letter postal codes.

You can make the state selection list visible at all times in your application or you can display an arrow at all times to indicate that a selection list is available. Now you change the property for the state column to show the arrow at all times.

1. Click the state column in the Design view.

   Make sure the Properties view displays.

   The Properties view displays properties of the column.

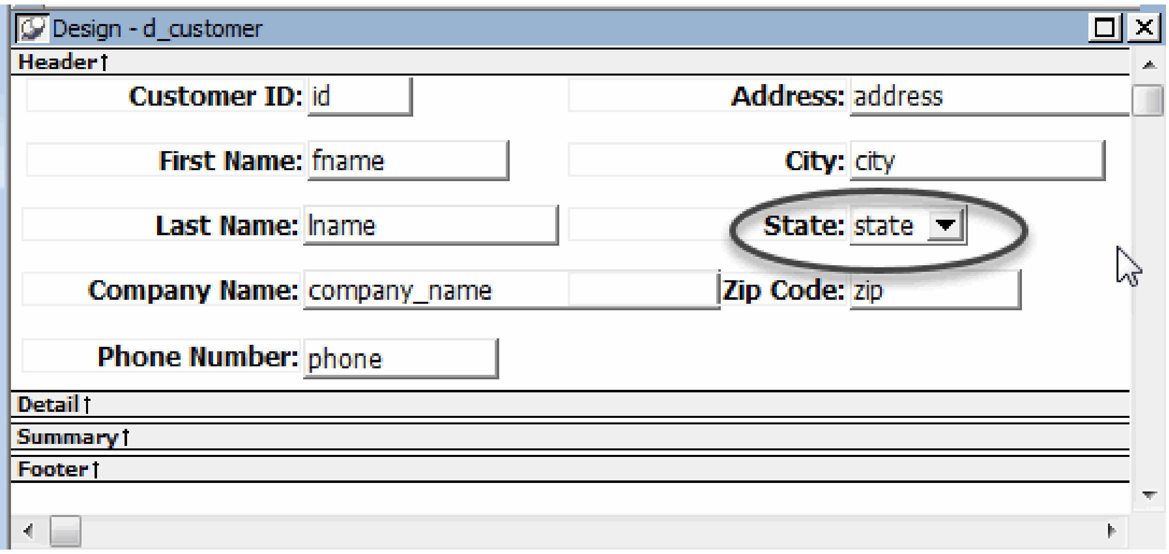
2. Click the Edit tab in the Properties view.

   You might need to click the arrow keys near the top of the Properties view to display the Edit tab before you can click it. Notice that the Style Type selection is DropDownDW.

3. Select the Always Show Arrow check box.

   Make sure the state column in the Design view is wide enough to display two characters plus the arrow symbol.

   An arrow appears next to the state column in the Design and Preview views. While the column is selected in Design view, you can make the column wider by holding the cursor over the right edge of the column until the cursor symbol changes to a double-headed arrow, then dragging the edge toward the rightmost frame of the view.



4. Click the Save button in PainterBar1.

5. Click the Close button in PainterBar1.

2.8 LESSON 8 Attaching the DataWindow Objects

After you create and save a DataWindow object, you can use it in a window. You have already created the d_custlist and the d_customer DataWindow objects. Now you associate each of these DataWindow objects with a DataWindow control in the w_customers window.

In this lesson you:

- Attach the DataWindow object to the master DataWindow control
- Attach the DataWindow object to the detail DataWindow control
- Run the application
- Attach DataWindow objects to the Product window
- Run the application again

How long does it take?

About 15 minutes.

2.8.1 Attach the DataWindow object to the master DataWindow control

Where you are

> Attach the DataWindow object to the master DataWindow control

Attach the DataWindow object to the detail DataWindow control

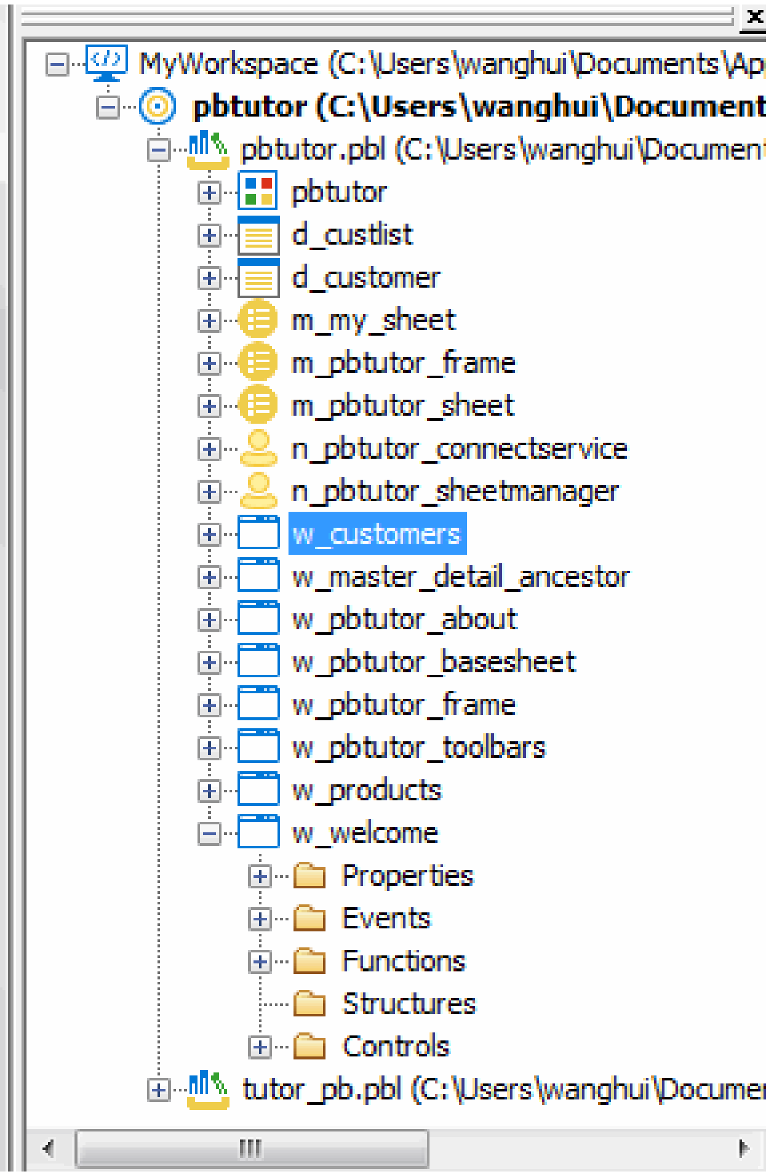
Run the application

Attach DataWindow objects to the Product window

Run the application again

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

1. Expand the pbtutor.pbl branch in the System Tree.



2. Right-click w_customers and select Edit from the pop-up menu

   or

   Double-click w_customers in the System Tree.

   The Window painter displays the w_customers window.

3. Right-click the top DataWindow control (dw_master) in the Layout view.

   If the Properties view is not displayed, select Properties from the pop-up menu.

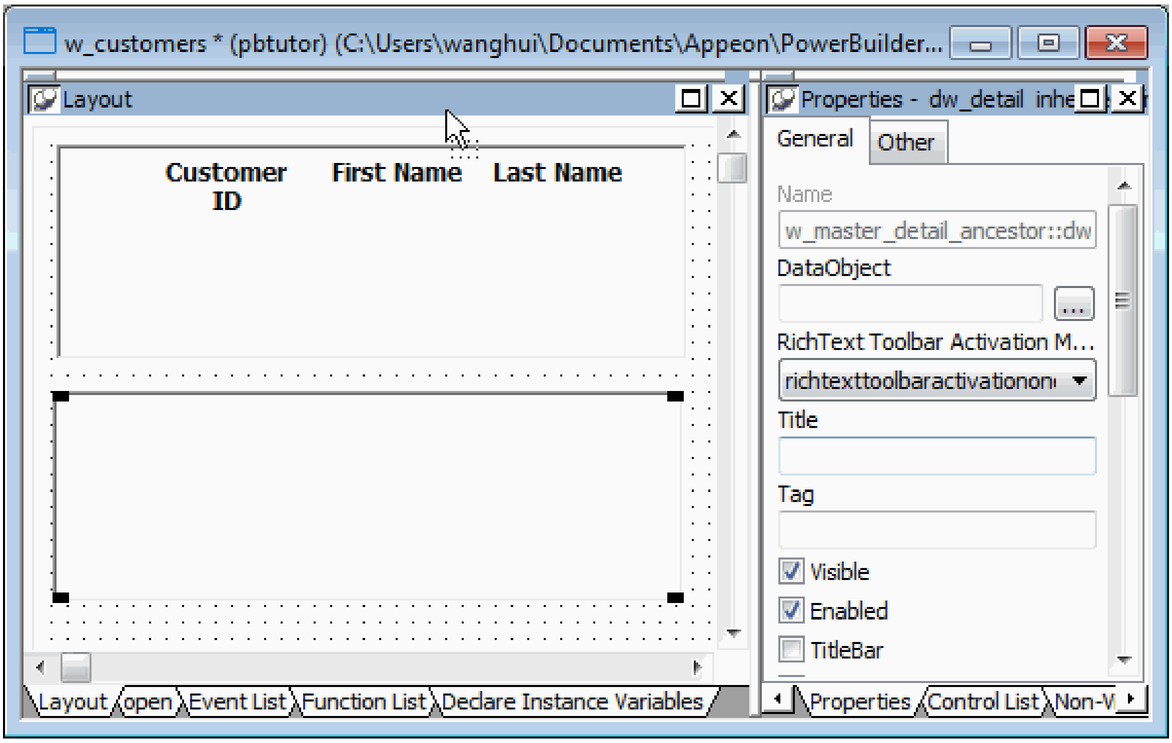
   Click the ellipsis button next to the DataObject box in the Properties view.

The Select Object dialog box displays.

4. Select d_custlist in the DataWindows list box and click OK.

   PowerBuilder associates the d_custlist DataWindow object with the dw_master DataWindow control.

   The Layout view now shows the d_custlist DataWindow headings inside the dw_master control, but you do not see any data yet. The DataWindow does not execute its SELECT statement until you run the application.



w_customers * (pbtutor) (C:\Users\wanghui\Documents\Appeon\PowerBuilder...

Properties - dw_detail inhe

Name: w_master_detail_ancestor::dw

DataObject:

RichText Toolbar Activation M...: richtexttoolbaractivationon

Title:

Tag:

☑ Visible

☑ Enabled

□ TitleBar

Adding DataWindow objects to the window using drag and drop

In this tutorial, you use a custom DataWindow control that has built-in error handling. If you want to use the standard DataWindow control, you do not need to add the control to the window and then attach a DataWindow object to it as you did in this lesson. You can simply select the DataWindow object you want from the System Tree and drag it onto the window in the Layout view. PowerBuilder creates the DataWindow control for you.

2.8.2 Attach the DataWindow object to the detail DataWindow control

Where you are

Attach the DataWindow object to the master DataWindow control

> Attach the DataWindow object to the detail DataWindow control

Run the application

Attach DataWindow objects to the Product window

Run the application again

Now you attach a DataWindow object to the detail DataWindow control. The Window painter should still be open for the w_customers window.

1. Select the bottom DataWindow control (dw_detail) in the Layout view.

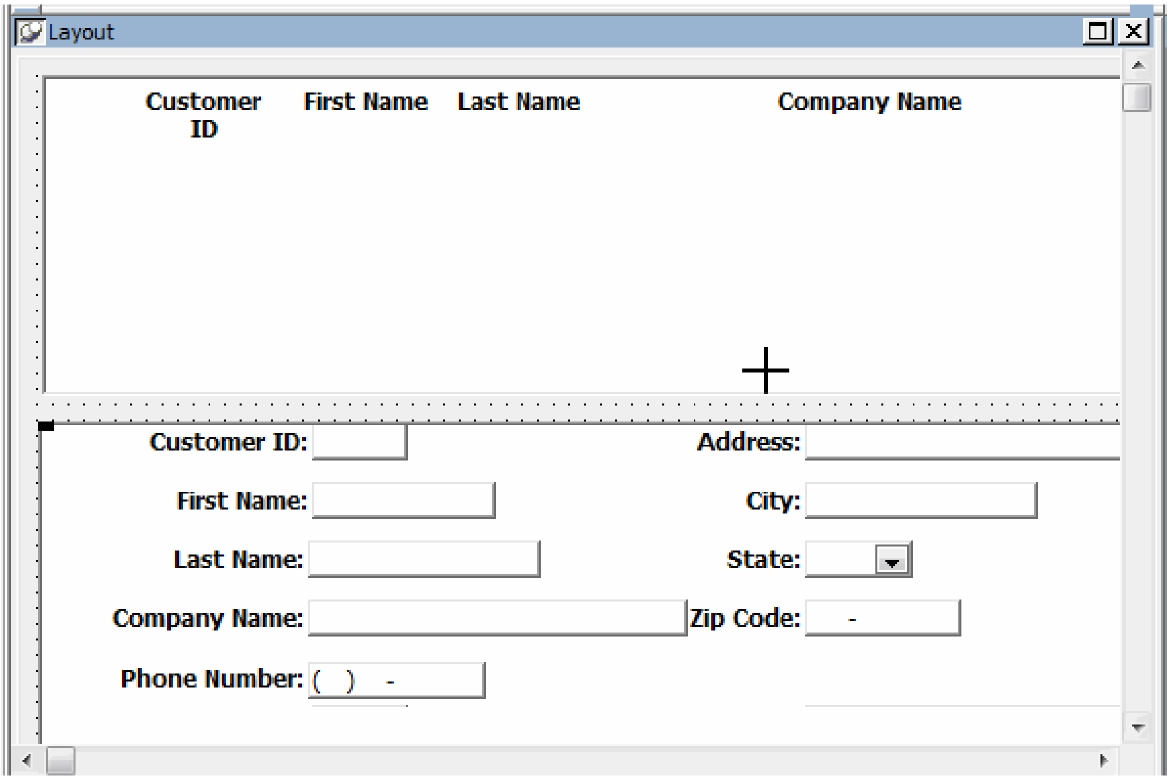
   Click the ellipsis button next to the DataObject box in the Properties view.

   Select d_customer in the Select Object dialog box and 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.

2. In the Layout view, make the dw_detail control larger so that 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.



2.8.3 Run the application

Where you are

[Attach the DataWindow object to the master DataWindow control]

[Attach the DataWindow object to the detail DataWindow control]

> [Run the application]

[Attach DataWindow objects to the Product window]

[Run the application again]

Now you 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 login window displays.

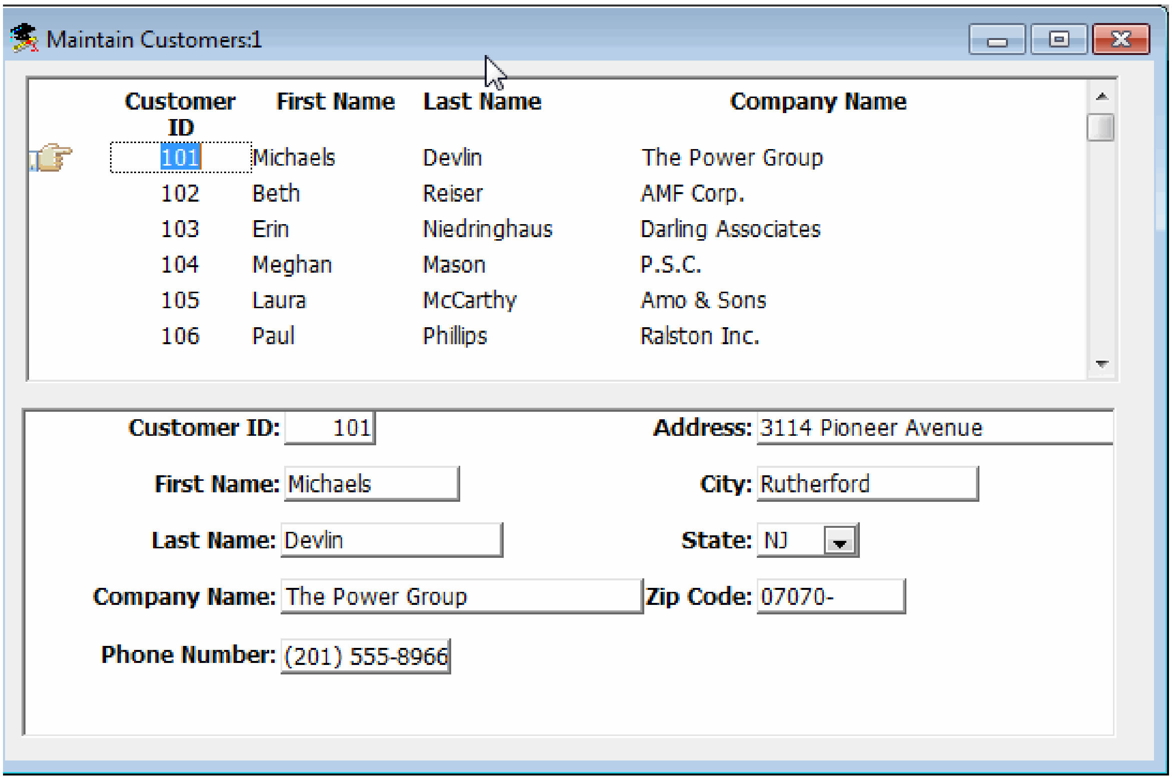
3. Type dba in the User ID box.

   Type sql in the Password box and click OK.

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

4. Select File>Report>Maintain Customers from the menu bar.

   The Customer window displays.



Customer ID	First Name	Last Name	Company Name
101	Michaels	Devlin	The Power Group
102	Beth	Reiser	AMF Corp.
103	Erin	Niedringhaus	Darling Associates
104	Meghan	Mason	P.S.C.
105	Laura	McCarthy	Amo & Sons
106	Paul	Phillips	Ralston Inc.

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		

   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 further information about the selected customer.

5. In the running application, click the Insert button in the toolbar

   or

   Select Edit>Insert from the Frame window menu bar.

This clears (resets) the dw_detail DataWindow, allowing you to add information for a new row that you will insert into the data source. 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 detail DataWindow.

   Typing information for a new customer

   The Customer ID number must be unique. To avoid duplicate numbers, use a four-digit number for your new database entry, or scroll down the list of three-digit customer numbers in the master DataWindow and select an ID number that does not appear in the list.

   Enter values for the remaining fields.

   The phone number and zip code use edit masks to display the information you type. You must enter numbers only for these data fields. To specify the state in which the customer resides, you must click the arrow next to the state column and select an entry from the drop-down list box.

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.

   The new customer does not yet display in the master DataWindow. (You could add code to include this feature). However, if you open another instance of the w_customers sheet, the new customer data is visible in both the master and detail DataWindow controls.

8. Click OK in the message box.

   Click a customer in the master DataWindow.

   That customer data displays in the lower DataWindow.

9. Change the customer address in the detail DataWindow.

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. Click OK in the message box.

    Select another customer in the master DataWindow.

    That customer data displays in the detail 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 is not 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 Demo Database may refer to the row that you are 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.

2.8.4 Attach DataWindow objects to the Product window

Where you are

Attach the DataWindow object to the master DataWindow control

Attach the DataWindow object to the detail DataWindow control

Run the application

> Attach DataWindow objects to the Product window

Run the application again

Now you add two DataWindow objects to the w_products window. These DataWindow objects are provided for you in the tutor_pb.pbl library.

1. Right-click w_products in the System Tree and select Edit from the pop-up menu

   or

   Double-click w_products in the System Tree.

   The Window painter displays the w_products window.

2. If the Control List view is not open, select View>Control List from the View menu.

   Select the dw_master DataWindow control in the Control List view.

   Click the ellipsis button next to the DataObject box in the Properties view.

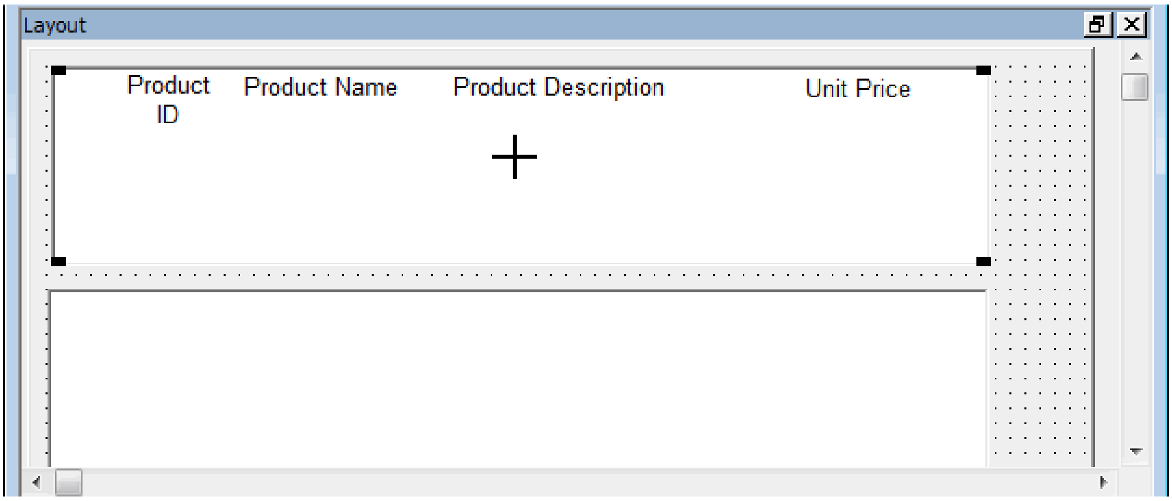
   PowerBuilder displays the Select Object dialog box.

3. Select tutor_pb.pbl in the Application Libraries box.

   Select d_prodlist in the DataWindows box and click OK.

   The Data Object box in the Properties view of the Window painter now displays d_prodlist.

   PowerBuilder associates the d_prodlist DataWindow object with the dw_master DataWindow control in the w_products window. You see the headings for the DataWindow object in the Layout view. You might need to resize the control and/or the window.



4. Click the dw_detail DataWindow control in the Control List view.

   Click the ellipsis button next to the DataObject box in the Properties view.

   The Select Object dialog box displays.

5. Select tutor_pb.pbl in the Application Libraries box.

   Select d_product in the DataWindows list box and click OK.

   PowerBuilder associates the d_product DataWindow object with the dw_detail DataWindow control. The Layout view 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, in the Layout view, make the dw_detail control larger so that you can see all of the columns in the DataWindow object. You can also enlarge the window to make more room.

2.8.5 Run the application again

Where you are

Attach the DataWindow object to the master DataWindow control

Attach the DataWindow object to the detail DataWindow control

Run the application

Attach DataWindow objects to the Product window

> Run the application again

Now you 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 (Run) in the PowerBar.

PowerBuilder prompts you to save your changes.

2. Click Yes.

   The application begins running, and the login window displays.

3. Type dba in the User ID box.

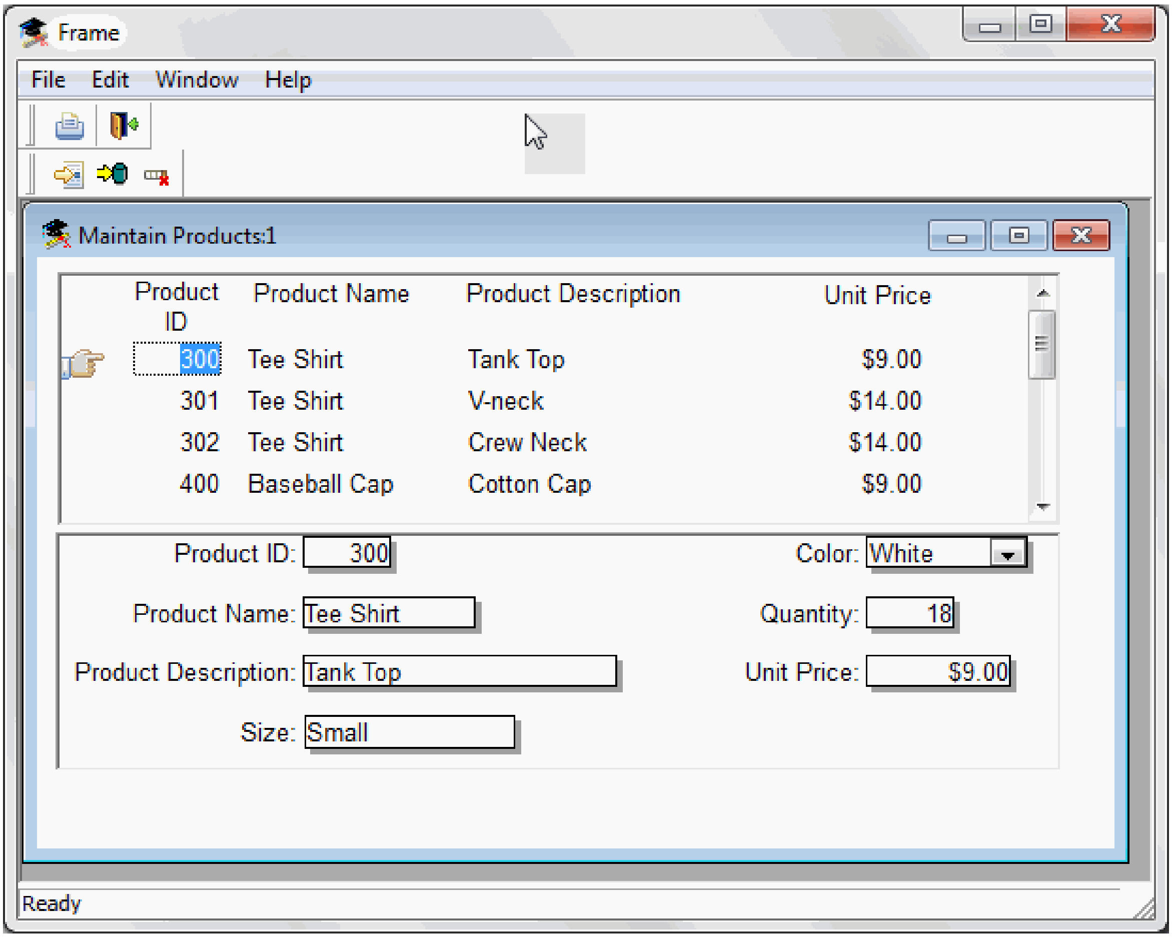
   Type sql in the Password box and click OK.

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

4. Select File>Report>Maintain Products from the menu bar.

   The Product window displays. The top DataWindow control shows all of the rows retrieved from the Product table.

   The bottom DataWindow control shows information about the product selection in the top DataWindow control.



Product ID	Product Name	Product Description	Unit Price
300	Tee Shirt	Tank Top	$9.00
301	Tee Shirt	V-neck	$14.00
302	Tee Shirt	Crew Neck	$14.00
400	Baseball Cap	Cotton Cap	$9.00

Product ID: 300

Product Name: Tee Shirt

Product Description: Tank Top

Size: Small

Color: White

Quantity: 18

Unit Price: $9.00

5. Select Edit>Insert from the menu bar.

   This clears the dw_detail DataWindow and allows you to add 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.

   The new product does not display yet in the top DataWindow, but if you open another product sheet, the new information displays. If you want, you can add code to the Clicked event of the update button to automatically refresh the data in the master DataWindow control.

8. Click OK in the message box.

   Click a product in the master DataWindow.

   That product data displays in the detail DataWindow.

9. Change the product's unit price.

   Select Edit>Update from the menu bar.

   This sends the revised product data to the database and displays another confirmation message.

10. Click OK in the message box.

    Select another product in the master DataWindow.

    That product's data displays in the detail DataWindow.

11. Select Edit>Delete from the menu bar.

    The product is deleted from the DataWindow immediately but is not deleted from the database until you select the Update option on the Edit menu.

12. Select File>Exit from the menu bar.

    The application closes and you return to the Window painter.

13. Close the Window painter.

2.9 LESSON 9 Running the Debugger

Sometimes your application does not behave the way you think it will. Perhaps a variable is not being assigned the value you expect, or a script does not 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 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 correct the values of variables.

In this lesson you:

- Add breakpoints in application scripts

- Run in debug mode

- Set a watch and a conditional breakpoint

How long does it take?

About 20 minutes.

2.9.1 Add breakpoints in application scripts

Where you are

> Add breakpoints in application scripts

Run in debug mode

Set a watch and a conditional breakpoint

Now you open the Debugger and add breakpoints to examine the behavior of the login and Customer windows. When PowerBuilder runs the application in debug mode, it stops just before executing a line containing a breakpoint.

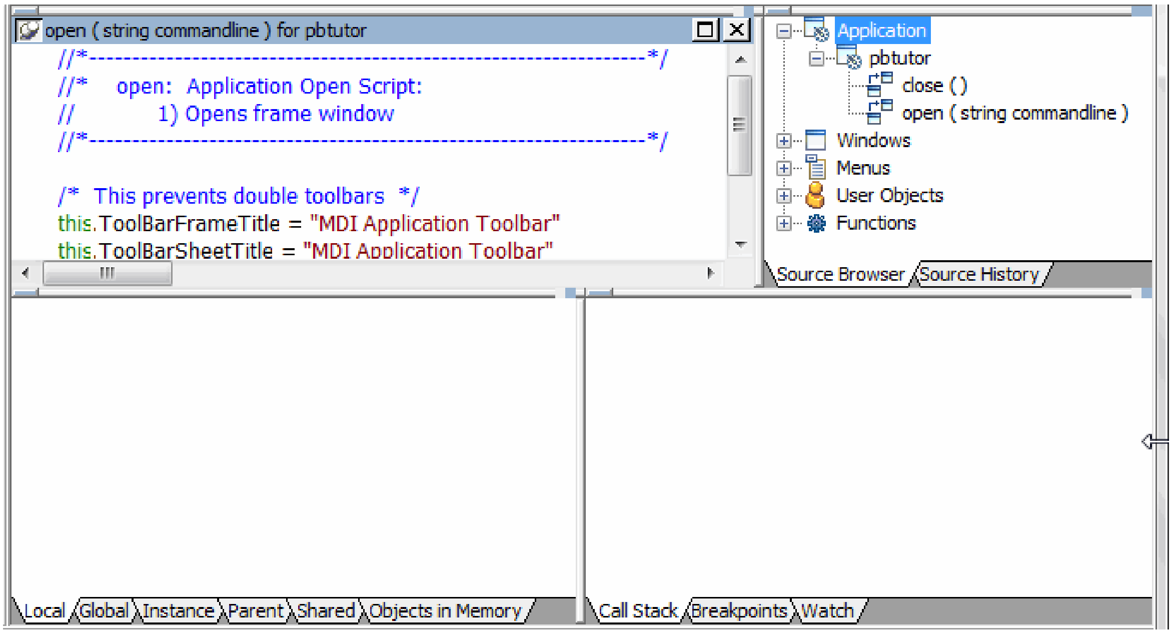
When you insert breakpoints in a script, you should select lines that contain executable statements. If you try to set a breakpoint in variable-declaration lines, comment lines, or blank lines, PowerBuilder sets the breakpoint at the next executable line.

1. Click the Debug button (Debug) in the PowerBar.

   PowerBuilder opens the Debugger. There are three stacks of tabbed panes in the default view layout scheme. The Source view is visible in a single pane at the top left of the Debug window. The Source Browser view is open in the pane at the top right.

If the Debug window looks different

If you have opened the Debug window before and opened, moved, or closed any views, your display may look different. To restore the default view layout scheme, select View>Layouts>Default from the menu bar.



open (string commandline) for pbtutor

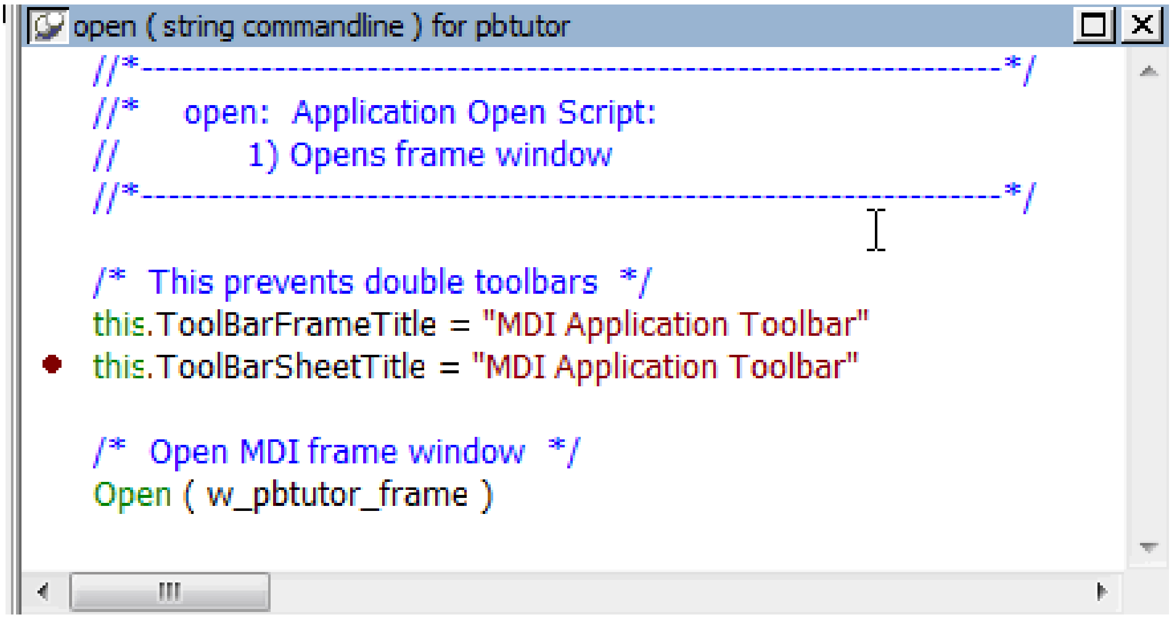
```
//*-----------------------------------------------------------------*/
//*    open:  Application Open Script:
//           1) Opens frame window
//*-----------------------------------------------------------------*/

/*  This prevents double toolbars  */
this.ToolBarFrameTitle = "MDI Application Toolbar"
this.ToolBarSheetTitle = "MDI Application Toolbar"
```

The source code for the application Open event displays in the Source view at top left. If it does not display, expand the Application node in the Source Browser view's tree view and double-click the Open event under the pbtutor application.

2. In the Source view, double-click the line containing the following assignment statement:

```
this.ToolBarSheetTitle = "MDI Application Toolbar"
```

A black breakpoint symbol displays at the start of the line to show that a breakpoint has been set on the statement.



open (string commandline) for pbtutor

```
//*-----------------------------------------------------------*/
//*    open:  Application Open Script:
//          1) Opens frame window
//*-----------------------------------------------------------*/

/*  This prevents double toolbars  */
this.ToolBarFrameTitle = "MDI Application Toolbar"
● this.ToolBarSheetTitle = "MDI Application Toolbar"

/*  Open MDI frame window  */
Open ( w_pbtutor_frame )
```

3. Expand the following node in the Source Browser view: Windows>w_welcome>cb_ok

The Source Browser view lists only events that have been coded. The only event for the login window OK button is the Clicked event.

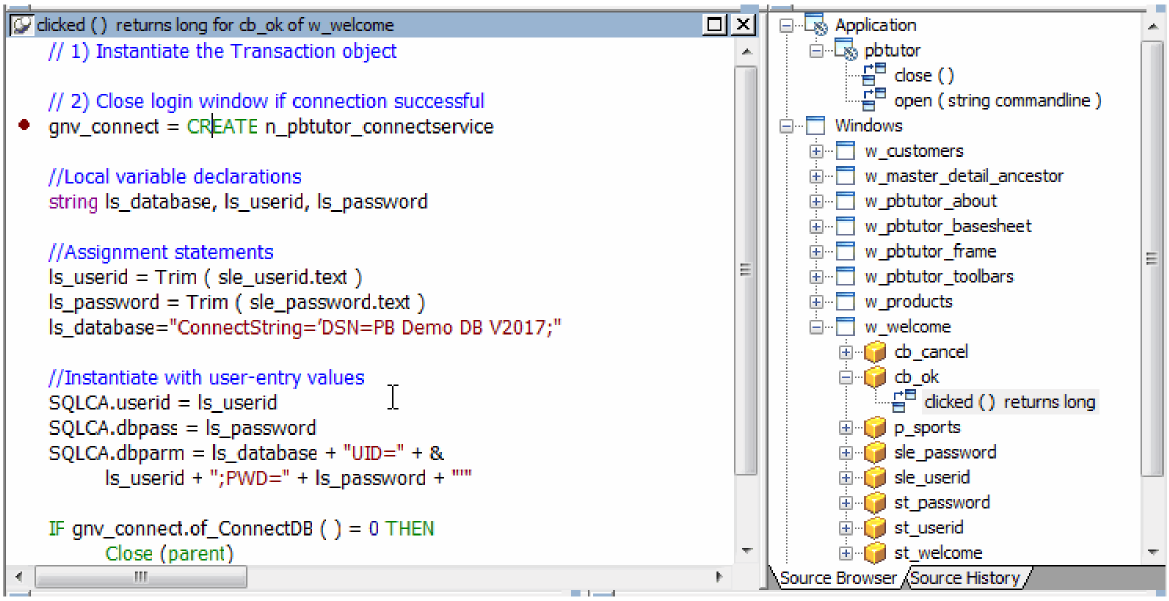
4. Double-click the Clicked event for the cb_ok button in the Source Browser view.

The code for the Clicked event displays in the Source view.

5. Double-click the following line:

```
gnv_connect = CREATE &
         n_pbtutor_connectservice
```

A breakpoint symbol displays at the start of the line.



clicked ( ) returns long for cb_ok of w_welcome

```
// 1) Instantiate the Transaction object

// 2) Close login window if connection successful
gnv_connect = CREATE n_pbtutor_connectservice

//Local variable declarations
string ls_database, ls_userid, ls_password

//Assignment statements
ls_userid = Trim ( sle_userid.text )
ls_password = Trim ( sle_password.text )
ls_database="ConnectString='DSN=PB Demo DB V2017;"

//Instantiate with user-entry values
SQLCA.userid = ls_userid
SQLCA.dbpass = ls_password
SQLCA.dbparm = ls_database + "UID=" + &
        ls_userid + ";PWD=" + ls_password + "'"

IF gnv_connect.of_ConnectDB ( ) = 0 THEN
        Close (parent)
```

6. Double-click w_master_detail_ancestor in the Source Browser view.

7. Double-click dw_master, then rowfocuschanged.

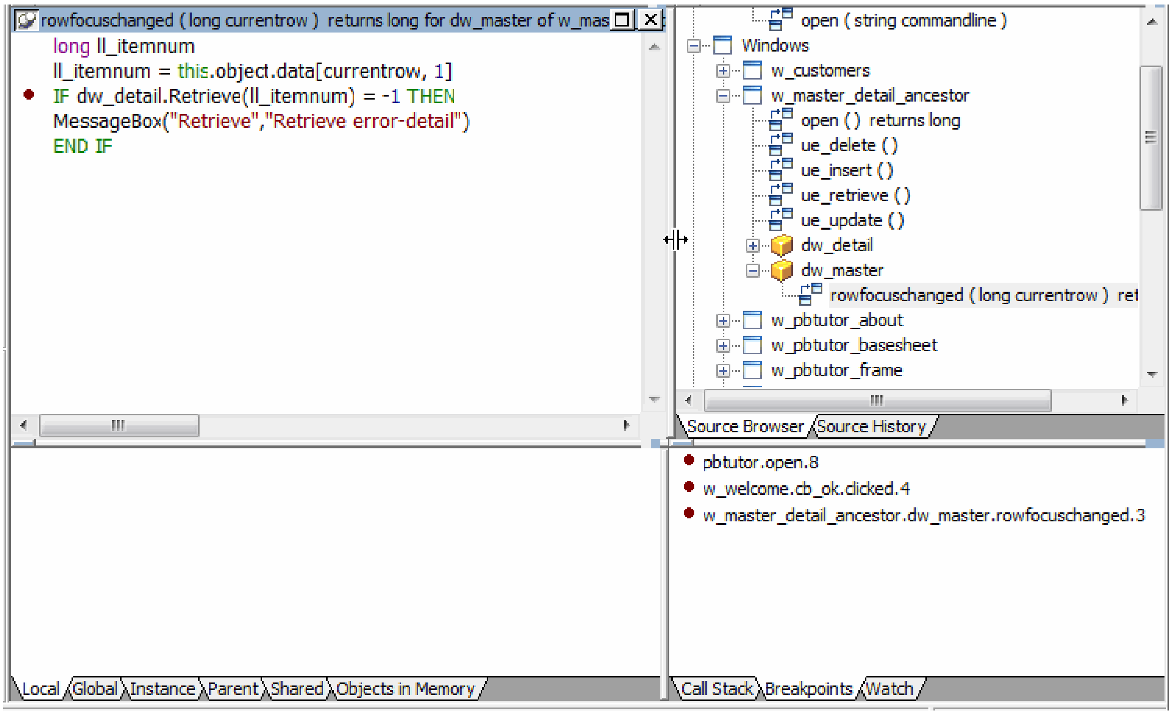
   PowerBuilder displays the script for the RowFocusChanged event of the dw_master DataWindow control in the Source view.

8. Double-click this line:

   ```
   IF dw_detail.Retrieve(ll_itemnum) = -1 THEN
   ```

   A breakpoint symbol displays at the start of the line.

9. Select the Breakpoints tab in the lower-right stack.



rowfocuschanged (long currentrow) returns long for dw_master of w_mas

```
long ll_itemnum
ll_itemnum = this.object.data[currentrow, 1]
IF dw_detail.Retrieve(ll_itemnum) = -1 THEN
MessageBox("Retrieve","Retrieve error-detail")
END IF
```

● pbtutor.open.8
● w_welcome.cb_ok.clicked.4
● w_master_detail_ancestor.dw_master.rowfocuschanged.3

You should see the breakpoints you set in the Breakpoints view. To complete this lesson, you need to have these breakpoints set correctly.

If you have additional breakpoints

You can clear any excess breakpoints using the pop-up menu in the Breakpoints view.

2.9.2 Run in debug mode

Where you are

Add breakpoints in application scripts

> Run in debug mode

Set a watch and a conditional breakpoint

Now you run the application in debug mode. You step through the code line by line.

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 Over to execute the function as a single statement. Use Step In if you want to step into a function and examine the effects of each statement in the function.

If you have stepped into a function, you can use Step Out to execute the rest of the function as a single step and return to the next statement in the script that called the function.

1. Click the Start button (Start) in PainterBar1

   or

   Select Debug>Start pbtutor from the menu bar.

   The application starts and runs until it hits a breakpoint (in this case, the call to the assignment statement for the toolbar title for sheet windows).

   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.

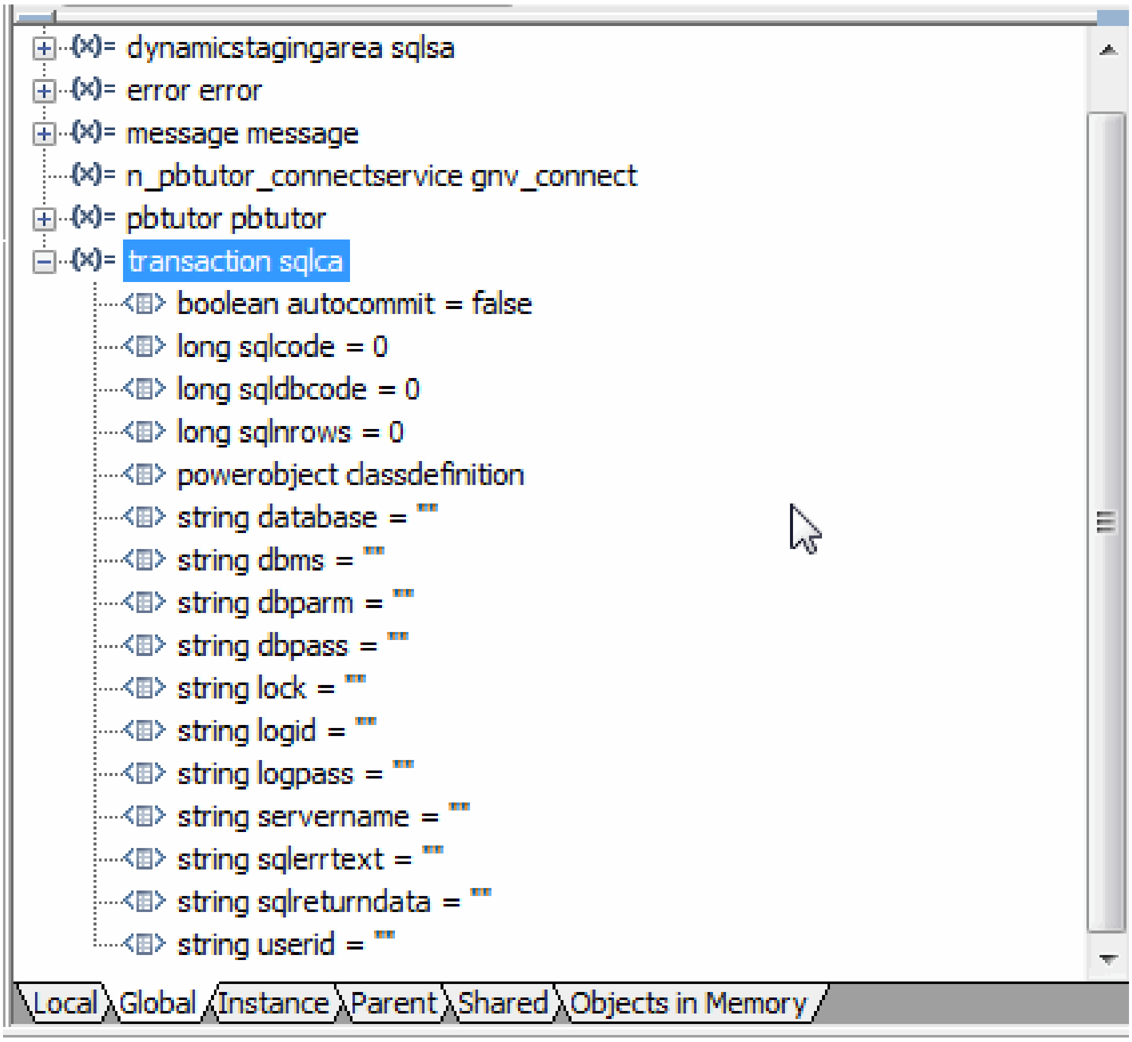
2. Click the Global tab in the lower-left stack.

   The Global Variables view displays.

3. Double-click transaction sqlca.

   Find the DBMS property, which has a String datatype.

   Notice that this property does not yet have a value associated with it because the Debugger interrupted execution before the ProfileString function executed.



```
(x)= dynamicstagingarea sqlsa
(x)= error error
(x)= message message
(x)= n_pbtutor_connectservice gnv_connect
(x)= pbtutor pbtutor
(x)= transaction sqlca
    boolean autocommit = false
    long sqlcode = 0
    long sqldbcode = 0
    long sqlnrows = 0
    powerobject classdefinition
    string database = ""
    string dbms = ""
    string dbparm = ""
    string dbpass = ""
    string lock = ""
    string logid = ""
    string logpass = ""
    string servername = ""
    string sqlerrtext = ""
    string sqlreturndata = ""
    string userid = ""
```

4. To execute the next statement, click the Step In button ( ) in PainterBar1

   or

   Select Debug>Step In from the menu bar.

   The application starts execution of the Open event for the MDI frame window.

5. Use Step In or Step Over to step through the code until you reach this statement in the script for the frame window Open event:

   ```
   open(w_welcome)
   ```

   After PowerBuilder finishes executing this statement, the login window displays and the Debug window is minimized.

   The Open event for the frame window also has a posted call to the ue_postopen function (that you stepped through without examining). This function in turn includes code that starts the processing of a chain of sheet manager functions. These functions are processed at the end of the script for the Open event, after the login window displays.

6. Click Step Over ( ) until the login window displays and the Debugger is minimized.

Type dba in the User ID box of the login window.

Type sql in the Password box and click OK.

You return to the Debug window. The yellow arrow in the Source view points to the next executable statement, the CREATE statement for the connection service object. This is the first executable line in the script for the Clicked event of the cb_ok command button.

7. Select the Call Stack tab in the lower-right stack.

   The yellow arrow in the Call Stack view indicates the current location in the call stack. If you double-click another line in the stack, the Source and Variables views change to display the context of that line, and a green arrow indicates the line in the Source view. If you then single-click another line in the stack, a green arrow displays in the Call Stack view to indicate the line for which context is displayed. When you continue to step through the code, the Source and Variables views return to the current context.

8. Click the Step In button.

   The Debugger takes you to the script for the Constructor event of the connection service object.

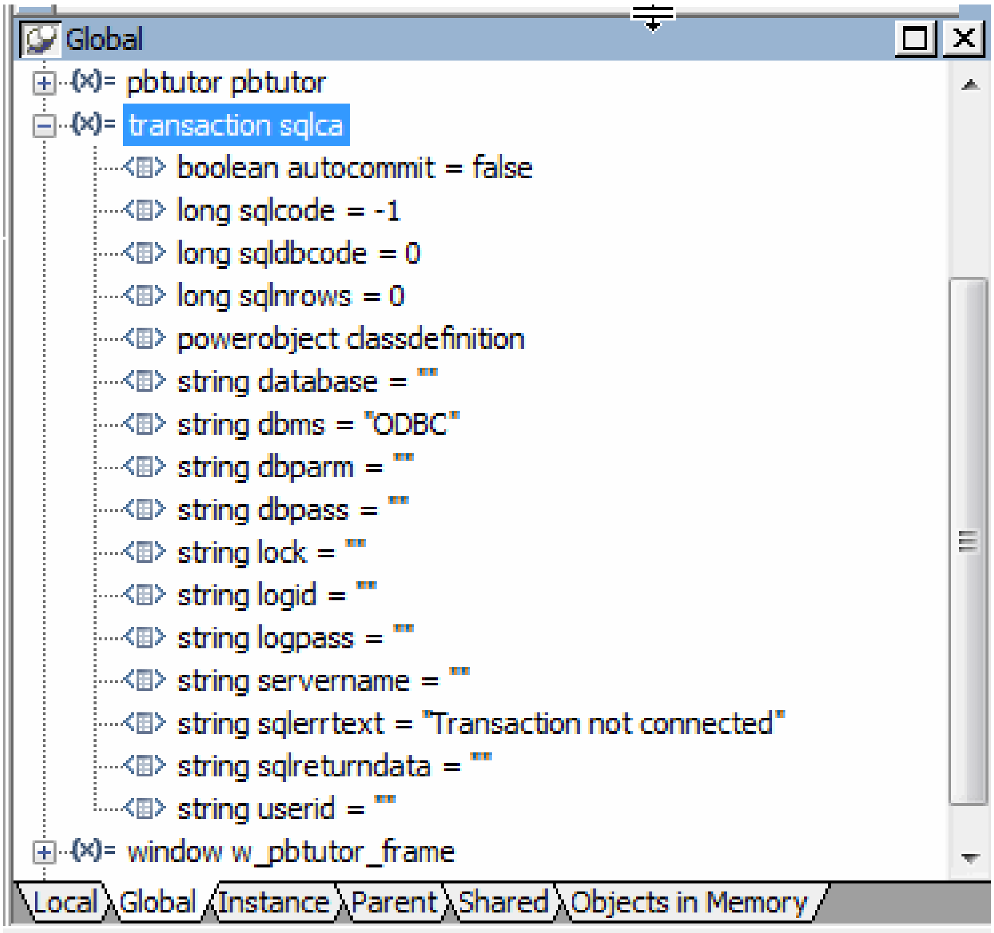
9. Click the Step Out button (Step Out).

10. Click the Global tab in the lower-left stack.

    Look again at the Transaction object properties.

    You step out of the Constructor event in a single step and return to the script for the OK button Clicked event. Now the value of sqlcode has changed, and the sqlerrortext and DBMS property have values, but the UserID, DBPass, and DBParm properties do not.

    The values were assigned during execution of the Constructor event of the connection service object after the of_GetConnectionInfo function returned information from the INI file, but because you commented out the lines in the code for the UserID, DBPass, and DBParm properties, these values were not retrieved.



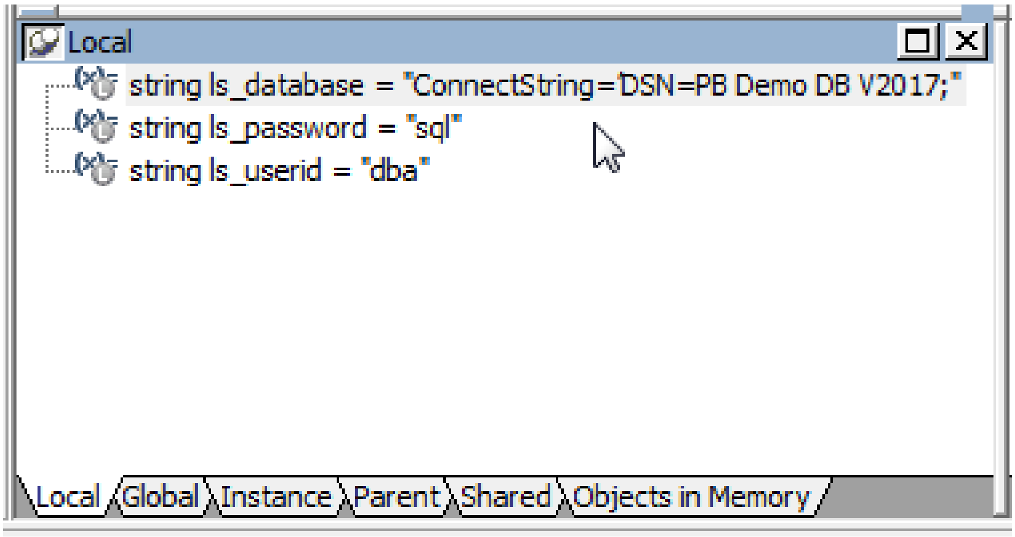
```
Global
+ pbtutor pbtutor
- transaction sqlca
    boolean autocommit = false
    long sqlcode = -1
    long sqldbcode = 0
    long sqlnrows = 0
    powerobject classdefinition
    string database = ""
    string dbms = "ODBC"
    string dbparm = ""
    string dbpass = ""
    string lock = ""
    string logid = ""
    string logpass = ""
    string servername = ""
    string sqlerrtext = "Transaction not connected"
    string sqlreturndata = ""
    string userid = ""
+ window w_pbtutor_frame
```

11.Click on the Local tab in the lower-left stack.

The local variables for the Clicked script have not yet been assigned values.

12.Use the Step In button to step through the three assignment statements for the local variables.

As you step through each statement, you can check that the values assigned to the local variables are what you expected.

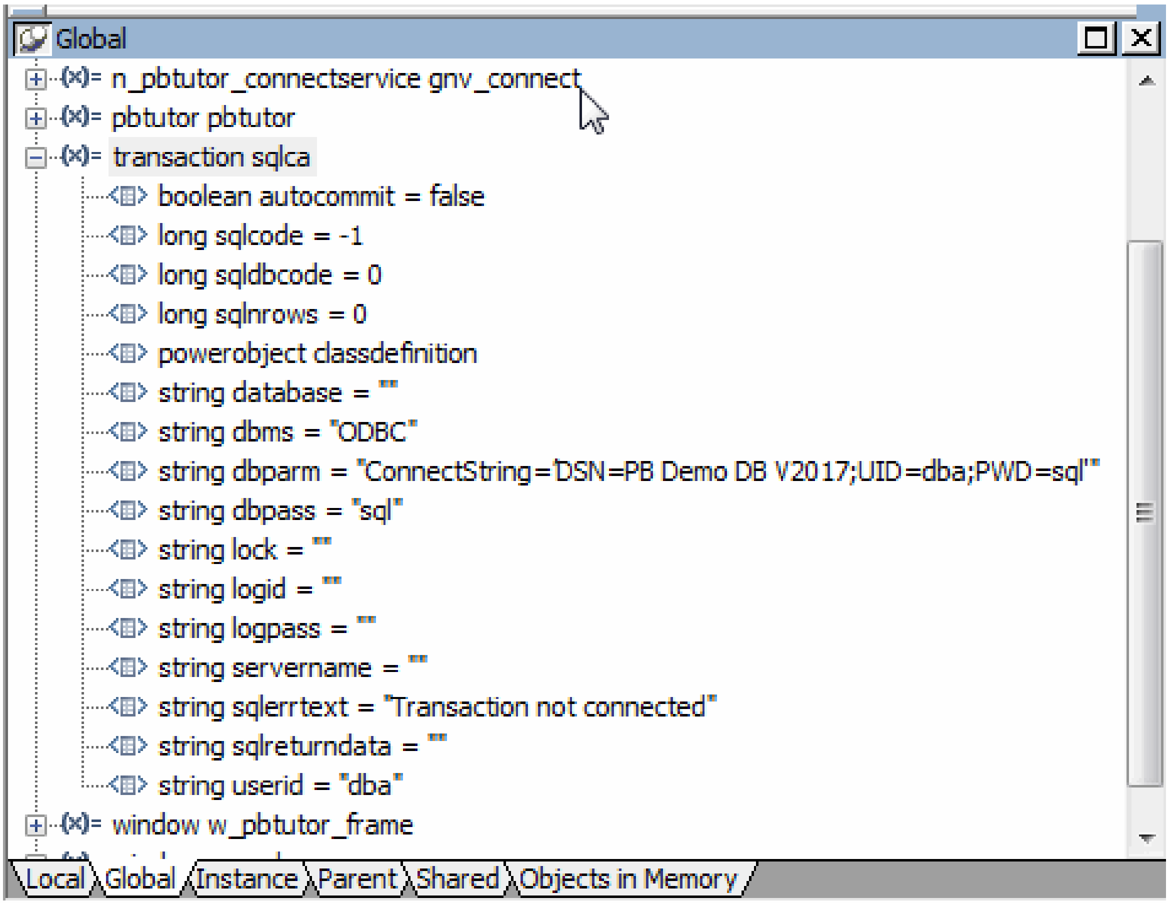


```
string ls_database = "ConnectString='DSN=PB Demo DB V2017;"
string ls_password = "sql"
string ls_userid = "dba"
```

13.Click again on the Global tab in the lower-left stack and expand the Transaction object.

Use the Step In button to step through the three lines that instantiate the Transaction object (SQLCA) with user-entry values for UserID, DBPass, and DBParm.

As you step through each statement, you can check that the values you entered in the login window are being assigned to the Transaction object. You are still not connected to the database until the connection service object of_Connect function is executed.



```
n_pbtutor_connectservice gnv_connect
pbtutor pbtutor
transaction sqlca
    boolean autocommit = false
    long sqlcode = -1
    long sqldbcode = 0
    long sqlnrows = 0
    powerobject classdefinition
    string database = ""
    string dbms = "ODBC"
    string dbparm = "ConnectString='DSN=PB Demo DB V2017;UID=dba;PWD=sql'"
    string dbpass = "sql"
    string lock = ""
    string logid = ""
    string logpass = ""
    string servername = ""
    string sqlerrtext = "Transaction not connected"
    string sqlreturndata = ""
    string userid = "dba"
window w_pbtutor_frame
```

14. Click the Continue button (Continue) in PainterBar1.

15. The Continue button resumes execution until the next breakpoint. The database connection is established, the login window closes, and the MDI frame for your application displays. The application is waiting for user input.

16. Select File>Report>Maintain 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.

2.9.3 Set a watch and a conditional breakpoint

Where you are

Add breakpoints in application scripts

Run in debug mode

> Set a watch and a conditional breakpoint

Next you set a watch on a variable whose value changes when the user selects a row in the Customer window. You then 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. Click the Watch tab in the lower-right stack.

   Click the Local tab in the lower-left stack.

   Select the ll_itemnum variable in the Local 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.

2. Click the Continue button (Continue).

   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.

3. Select a different row in the master DataWindow of the Customer window.

   You return to the Debug window. The new value of ll_itemnum displays in both the Local Variables view and the Watch view.

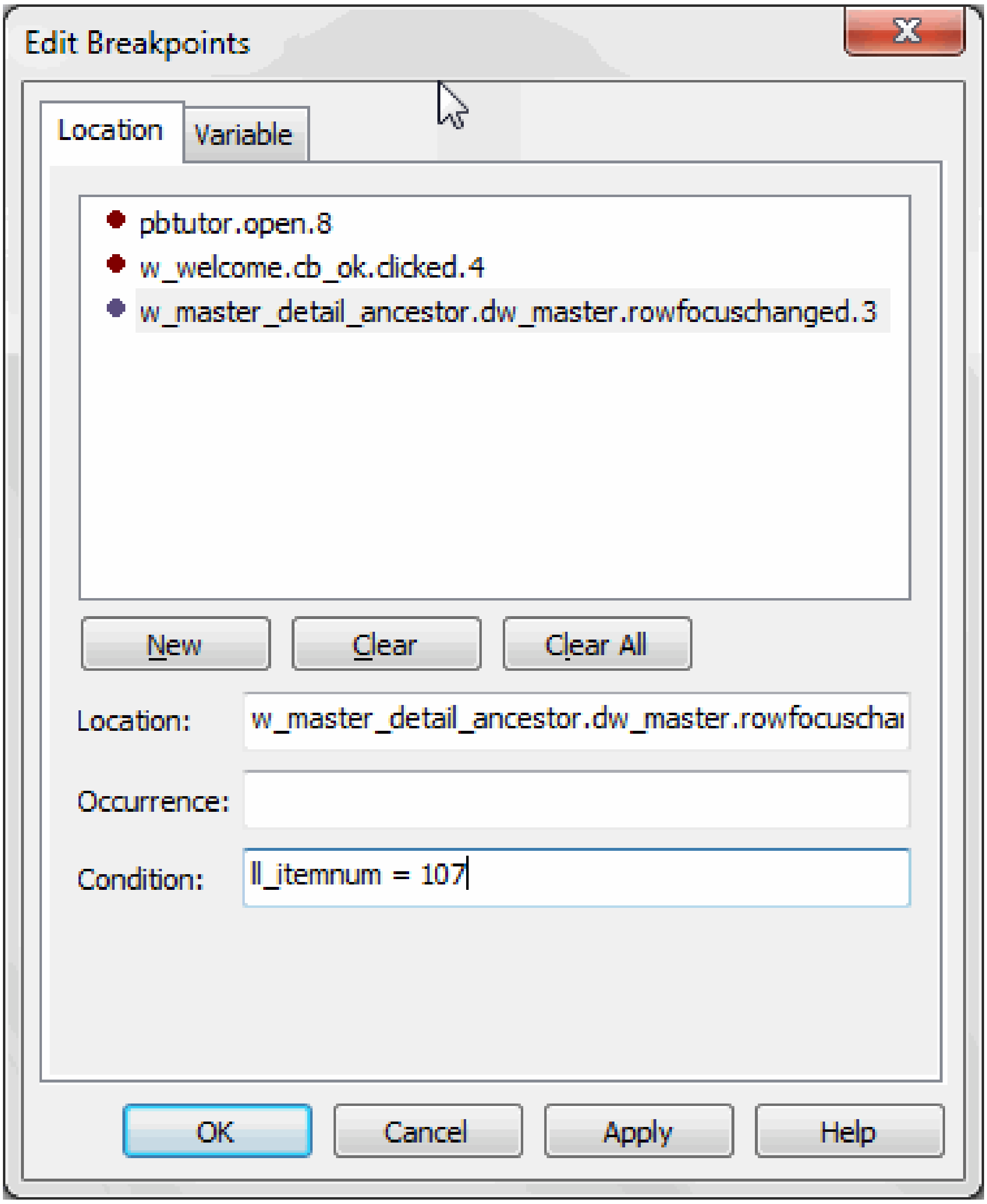
4. Click the Breakpoints tab in the lower-right stack.

   Double-click the rowfocuschanged breakpoint.

   The Edit Breakpoints dialog box opens with the breakpoint in the RowFocusChanged event selected.

5. Type the following line in the Condition text box and click OK:

```
ll_itemnum = 107
```



● pbtutor.open.8
● w_welcome.cb_ok.clicked.4
● w_master_detail_ancestor.dw_master.rowfocuschanged.3

Location: w_master_detail_ancestor.dw_master.rowfocuscha

Occurrence:

Condition: ll_itemnum = 107

The breakpoint in the RowFocusChanged event script is now a conditional breakpoint. PowerBuilder suspends execution only when it reaches this statement and the value of ll_itemnum is 107.

6. Click OK to close the dialog box.

   Click the Continue button.

   The application resumes execution. Now you can select different rows in the Customer window, and the Debug window opens at the breakpoint only if you select the customer whose ID is 107.

   If you select customer 107, click the Continue button again to return to the application.

7. Select File>Exit from the application's menu bar.

   The application terminates and you return to the Debug window.

8. Select File>Close from the menu bar.

   You return to the PowerBuilder development environment.

2.10 LESSON 10 Exception Handling

Exception handling allows you to trap errors that occur during the execution of a program and to provide useful information about those errors to the application user. This lesson describes how to create user-defined exception objects and use them to catch exceptions that you throw from a method in a TRY-CATCH statement.

In this lesson you:

- Add a new sheet window to the existing application
- Create user-defined exception objects
- Create a new user function and user event
- Call the methods and catch the exceptions
- Run the application

How long does it take?

About 45 minutes.

2.10.1 Add a new sheet window to the existing application

Where you are

> Add a new sheet window to the existing application

Create user-defined exception objects

Create a new user function and user event

Call the methods and catch the exceptions

Run the application

In this lesson you add a third sheet window to the main tutorial application. You create and call a function to perform a routine operation (calculate a percentage) on values returned from embedded SQL commands and a value selected by the application user from a drop-down list box control.

The prototype for the function you create throws user-defined exceptions. You call the function in a TRY-CATCH block inside the Clicked event on a command button control. The CATCH clauses in the Clicked event catch user-defined exceptions thrown by the new function as well as a system exception thrown up the application call stack.

You use the new sheet window to calculate the percentage of customers that resides in a selected state. The controls you add to the new sheet window are:

- Two static text boxes that you change programmatically to display read-only results
- A command button to call a function that calculates percentages
- A drop-down list box for a list of states where customers reside
- A text box that displays the percentage of customers residing in the state that application users select from the drop-down list box

To add a sheet window to the existing application, you must:

- Create the sheet window
- Provide access to the sheet window from the main application frame

2.10.1.1 Create the sheet window

You inherit the sheet window from the w_pbtutor_basesheet window. This is the base class for sheet windows that you generated with the Template Application wizard. You do not use the w_master_detail_ancestor extension layer window, since the modifications you made to it are not useful in the new sheet window.

1. Select File>Inherit from the PowerBuilder menu.

   Make sure the Objects of Type box displays Windows.

   Select w_pbtutor_basesheet from the available windows in the pbtutor.pbl library and click OK.

2. Make sure the Layout view displays in the Window painter.

   Select Insert>Control>StaticText and click near the top left corner of the Layout view.

3. In the Properties view, highlight the default text in the Text text box and type the following:

   ```
   1. Select or type a state code in drop-down list:
   ```

4. Lengthen the control width and the width of the sheet window to display the entire text and allow room for a drop-down list control at the top right of the window.

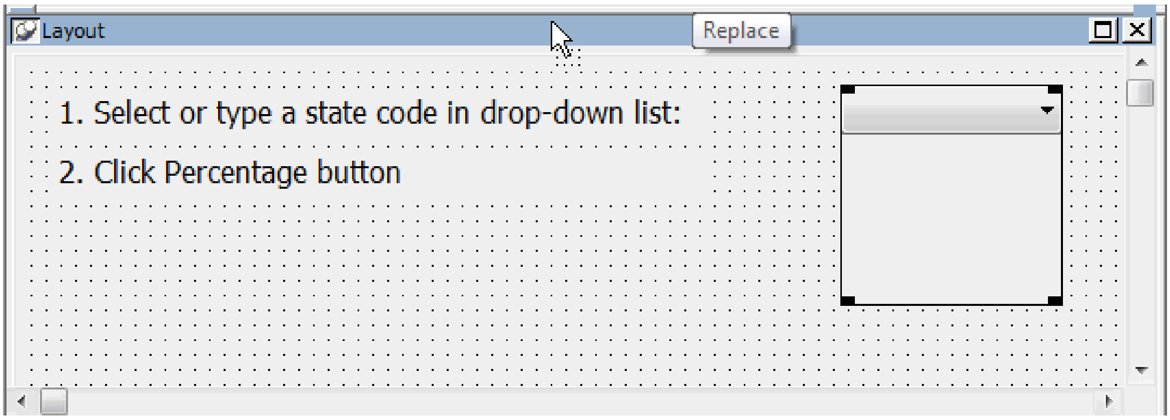
   A length of 2250 should be sufficient for the sheet window width. You can set this on the Other tab of the Properties view for the window, or you can drag the window edge in the Layout view to make room for an additional control.

5. Right-click the static text control in the Layout view and click Duplicate from the pop-up menu.

   In the Properties view, highlight the default text in the Text text box of the new static text control and type the following:

```
2. Click Percentage button
```

6. Select Insert>Control>DropDownListBox and click to the right of the static text boxes near the top right corner of the Layout view.


Layout
Replace
1. Select or type a state code in drop-down list:
2. Click Percentage button

7. In the Properties view for the drop-down list box, type ddlb_state for the control name.

   Select the AllowEdit and the VScrollBar check boxes.

8. Click the CommandButton button in the painter bar and click below the two static text boxes.

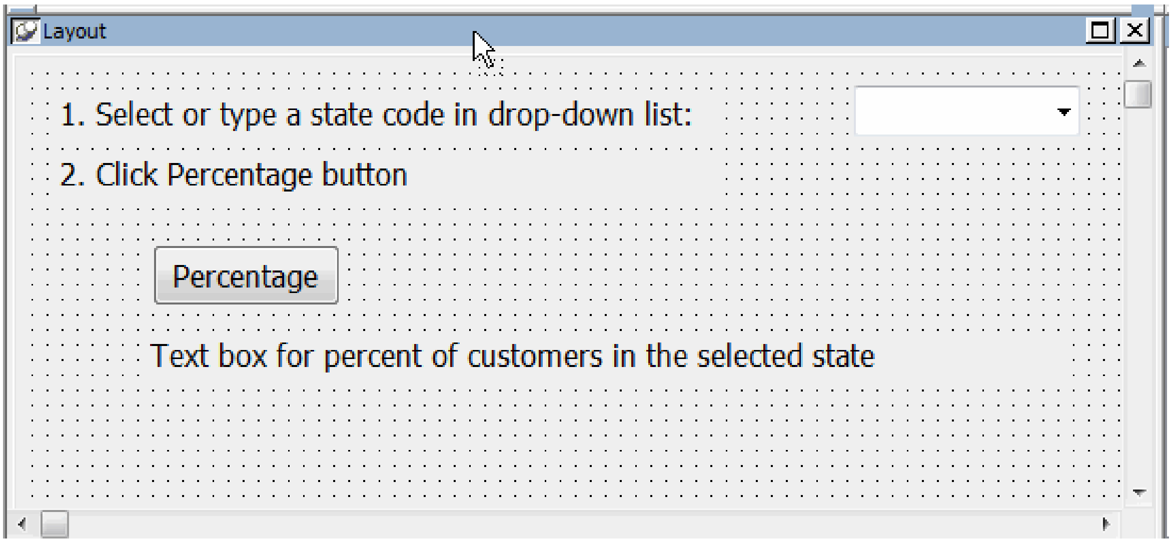
   In the Properties view, type cb_percent for the button name and type Percentage for the button text.

9. Select Insert>Control>SingleLineEdit and click below the command button in the Layout view.

   In the Properties view, type sle_result for the control name and type the following for the control text:

```
Text box for percent of customers in the selected state
```

10.Lengthen the control width to display the entire text.


Layout
1. Select or type a state code in drop-down list:
2. Click Percentage button
Percentage
Text box for percent of customers in the selected state

11. Make sure no control is selected and the sheet window properties are displayed in the Properties view.

    Type Customer Location for the Tag property.

    The text you typed will be visible in the sheet window title at runtime. Code in the basesheet ue_postopen event assigns the Tag text to the sheet window title.

12. Select File>Save from the PowerBuilder menu.

    Select pbtutor.pbl for the application library, type w_cust_pct for the new sheet window name, and click OK.

    This saves the new sheet window with all its controls to the main tutorial library.

2.10.1.2 Provide access to the sheet window from the main application frame

You must register the new sheet with the sheet manager.

1. Double-click w_pbtutor_frame in the System Tree.

2. If the ue_postopen event is not visible in the Script view, click the Event List tab and double-click ue_postopen.

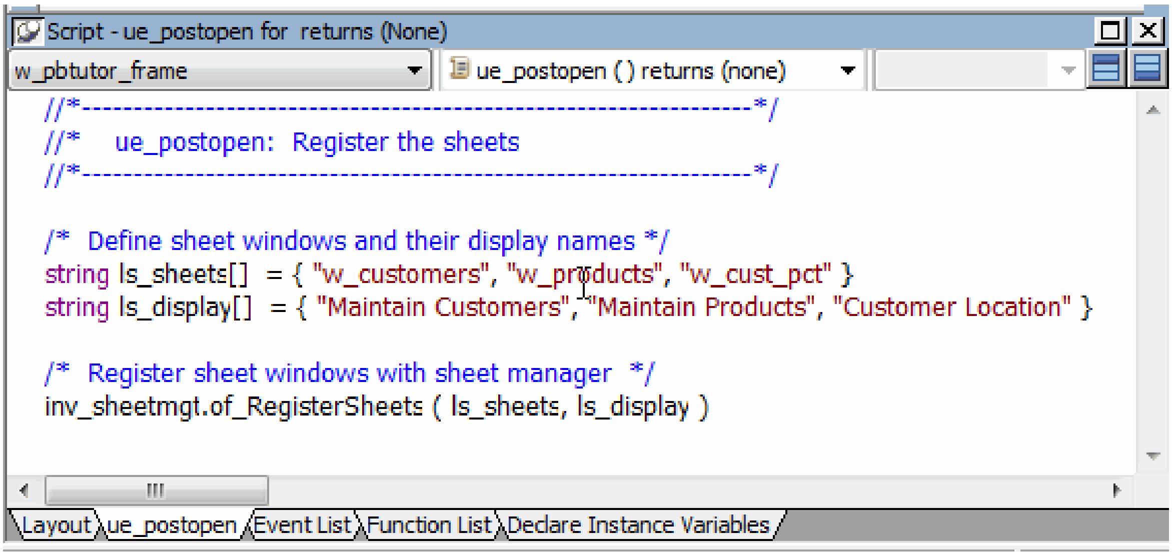
   The ue_postopen event script displays in the Script view.

3. Edit the list of sheets to add your new window. The following entry should be a single line:

```
string ls_sheets[] = { "w_customers", "w_products", "w_cust_pct" }
```

4. Edit the display list to add a label for your new window. The following entry should be a single line:

```
string ls_display[] = { "Maintain Customers", "Maintain Products", "Customer
 Location" }
```


Script - ue_postopen for returns (None)
w_pbtutor_frame
ue_postopen ( ) returns (none)
Layout
ue_postopen
Event List
Function List
Declare Instance Variables

```
//*-----------------------------------------------------------------*/
//*    ue_postopen:  Register the sheets
//*-----------------------------------------------------------------*/

/*  Define sheet windows and their display names */
string ls_sheets[]  = { "w_customers", "w_products", "w_cust_pct" }
string ls_display[]  = { "Maintain Customers", "Maintain Products", "Customer Location" }

/*  Register sheet windows with sheet manager  */
inv_sheetmgt.of_RegisterSheets ( ls_sheets, ls_display )
```

5. Select File>Save and close the w_pbtutor_frame window.

The next time you run the pbtutor application, you should be able to open the new sheet window from the File menu of the main frame window. Although you can now run the new sheet window from the development environment, you must make sure that you can run it from a compiled application as well.

For this purpose, you reference the sheet windows as window objects in the sheet manager of_registersheet script. The reference is necessary for the compiler to know that this object is used in the application so that it will include it in the executable.

You create a compiled application in Preparing the Application for Deployment

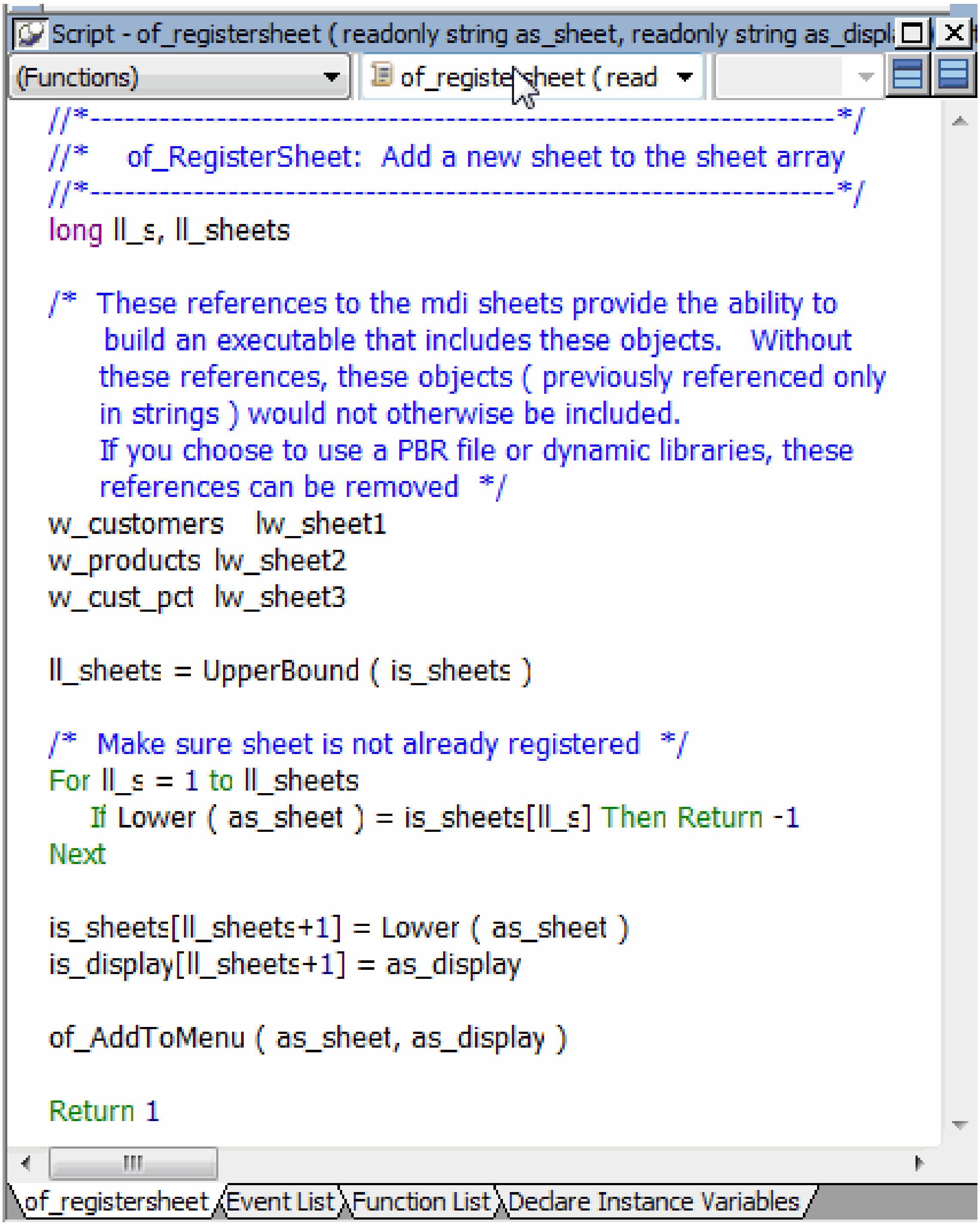
6. Double-click n_pbtutor_sheetmanager in the System Tree.

7. Click the Function List tab and double-click of_registersheet.

   The script for the of_registersheet function displays in the Script view.

8. Enter the following after the lines declaring sheet window variables for the w_customers and w_products windows:

   ```
   w_cust_pct lw_sheet3
   ```



Script - of_registersheet (readonly string as_sheet, readonly string as_displ
(Functions)
of_registersheet (read
of_registersheet
Event List
Function List
Declare Instance Variables

```
//*--------------------------------------------------------------*/
//*   of_RegisterSheet:  Add a new sheet to the sheet array
//*--------------------------------------------------------------*/
long ll_s, ll_sheets

/*  These references to the mdi sheets provide the ability to
     build an executable that includes these objects.   Without
     these references, these objects ( previously referenced only
     in strings ) would not otherwise be included.
     If you choose to use a PBR file or dynamic libraries, these
     references can be removed  */
w_customers   lw_sheet1
w_products  lw_sheet2
w_cust_pct  lw_sheet3

ll_sheets = UpperBound ( is_sheets )

/*  Make sure sheet is not already registered  */
For ll_s = 1 to ll_sheets
    If Lower ( as_sheet ) = is_sheets[ll_s] Then Return -1
Next

is_sheets[ll_sheets+1] = Lower ( as_sheet )
is_display[ll_sheets+1] = as_display

of_AddToMenu ( as_sheet, as_display )

Return 1
```

9. Save and close the n_pbtutor_sheetmanager user object.

2.10.2 Create user-defined exception objects

Where you are

Add a new sheet window to the existing application

> Create user-defined exception objects

Create a new user function and user event

Call the methods and catch the exceptions

Run the application

Now you create two user-defined exception objects that you will throw from a function that is invoked when the user clicks the command button on the w_cust_pct window. You also create a user-defined exception object that you throw from a user event on the drop-down list box control that you added to the w_cust_pct window.

1. Select File>New from the PowerBuilder menu and click the PBObject tab in the New dialog box.

2. Select the Standard Class icon and click OK.

   Select throwable from the Types list box and click OK.

   The new exception object displays in the User Object painter.

3. In the Text box of the Properties view, type the following:

   ```
   No rows were returned from the database. If you typed or selected a state code
    in the drop-down list box and the database connection has not been closed,
    either the state you entered has no customers or you entered the state code
    incorrectly.
   ```

   The exception object has get and set methods for handling the Text property. Here you set the text property directly in the user interface.

4. Click outside the Properties view to enable the Save button.

   Select File>Save, select the pbtutor.pbl application library, and type exc_no_rows in the User Objects box for the new exception object name, and click OK.

5. Select File>Close.

6. Repeat steps 1-5 using the following values for the Text property and the exception object name:

   Table 2.12:

Property	Value
Text	Percentage too low. Only one customer in this state. Notify regional sales manager...
Exception object name	exc_low_number

7. Repeat steps 1-5 using the following values for the Text property and the exception object name:

   Table 2.13:

Property	Value
Text	You must use the two-letter postal code for the state name.
Exception object name	exc_bad_entry

2.10.3 Create a new user function and user event

Where you are

Add a new sheet window to the existing application

Create user-defined exception objects

> Create a new user function and user event

Call the methods and catch the exceptions

Run the application

Now you add a function that you invoke from the Percentage command button's Clicked event and an event that is triggered when the focus is changed from the drop-down list box on the w_cust_pct window. The function calculates the percentage of customers living in a particular state. The event processes the current value of the drop-down list box control to make sure it is two characters in length (for the state code).

1. Open w_cust_pct in the Window painter if it is not already open.

   Select (Functions) in the first list box in the Script view.

   The Script view displays the Prototype window. The first drop-down list box in the Script view displays (Functions) and the second drop-down list box displays (New Function).

2. Select decimal for the Return Type and type uf_percentage for Function Name.

3. Select integer for the Argument Type and type ai_custbystate for the Argument Name.

   You will add a second argument in the next step.

4. Right-click anywhere in the Prototype window and select Add Parameter from the pop-up menu.

5. Select integer for the second Argument Type and type ai_totalcust for the second Argument Name.

6. Type exc_no_rows,exc_low_number in the Throws box.

7. Enter the following script for the new function:

   ```
   Decimal my_result
   exc_no_rows  le_nr
   exc_low_number le_ex
   /* Process two integers passed as parameters. Instantiate and throw exceptions
   if the first integer value is 0 or 1. Otherwise calculate a percentage and
   return a numeric value truncated to a single decimal place. If the second
   integer value is 0, catch and rethrow the runtime dividebyzero error during
   the calculation.
   */
   CHOOSE CASE ai_custbystate
      CASE 0
         le_nr = create exc_no_rows
         throw le_nr
      CASE 1
         le_ex = create exc_low_number
         throw le_ex
      CASE ELSE
   ```

```
      TRY
         my_result=(ai_custbystate/ai_totalcust)*100
      CATCH (dividebyzeroerror le_zero)
         throw le_zero
      End TRY
END CHOOSE
return truncate(my_result,1)
```

Later in this tutorial, you will call the uf_percentage function from the Clicked event on the command button, passing in two integers and processing the return value.

You now add a user event for the drop-down list box that throws the exc_bad_entry exception object when a user-entered state code is not exactly two characters in length.

8. Select ddlb_state in the first drop-down list box of the Script view and select (New Event) in the second drop-down list box.

9. Select integer for Return Type and type ue_modified for Event Name.

   Select string for Argument Type and type as_statecode for Argument Name.

   Type exc_bad_entry in the Throws box or drag it from the System Tree to the Throws box.

   Note that the Event ID is (None). You do not select an Event ID for the ue_modified event. If you selected an Event ID, you could not enter user-defined exception objects in the event Throws clause.

10.Enter the following script for the new ue_modified event:

```
exc_bad_entry  le_ex
//Make sure the current text in the drop-down list
//box is two characters in length. Otherwise,
//instantiate the exc_bad_entry exception object and
//throw the exception.
IF len(this.text)<>2 Then
      le_ex = create exc_bad_entry
      throw le_ex
END IF
Return 1
```

Next you call the ue_modified event and the uf_percentage function, and catch the exceptions thrown by these methods.

2.10.4 Call the methods and catch the exceptions

Where you are

Add a new sheet window to the existing application

Create user-defined exception objects

Create a new user function and user event

> Call the methods and catch the exceptions

Run the application

You now write code to populate the drop-down list box controls with state codes from the customer table in the Demo Database database. Since you made the control editable, an application user can also type in a value for the state code. Before you process a user-entered value, you check to make sure the value conforms to the conditions you set in the ue_modified event, namely that it is two characters in length.

You also add code to the Clicked event of the command button control to process the current state code in the drop-down list box control. In the Clicked event you call the uf_percentage function to calculate the percentage of customers from the selected state and catch all exceptions that can be thrown by the function.

1. Make sure the w_cust_pct is open in the Window painter and that ddlb_state displays in the first drop-down list box of the Script view.

2. Select losefocus ( ) returns long [pbm_cbnkillfocus] in the second drop-down list box.

3. Call the ue_modified event and catch the exception object that it throws by entering the following lines for the losefocus event script:

```
Try
      this.EVENT ue_modified(this.text)
Catch (exc_bad_entry le_be)
      messagebox ("from exc_bad_entry", &
         le_be.getmessage())
End Try

return 1
```

4. Select constructor ( ) returns long [pbm_constructor] from the second drop-down list box in the Script view prototype window for the ddlb_state control.

5. Enter the following lines in the Constructor event to populate the drop-down list box control:

```
int       li_nrows, n
string       ls_state

//Get the distinct count of all states in the
//customer table
SELECT count(distinct state) INTO :li_nrows
FROM customer;

//Declare the SQL cursor to select all states
//in customer table but avoid
//rows with duplicate values for state.
DECLARE custstatecursor CURSOR FOR
SELECT state FROM customer
GROUP BY state HAVING count(state)=1
UNION
SELECT state FROM customer
GROUP BY state
HAVING count(state)>1;
OPEN custstatecursor ;
//Populate the control with a single entry for
//every state in the customer table.
FOR n=1 TO li_nrows
      FETCH NEXT custstatecursor INTO :ls_state;
      this.additem( ls_state)
NEXT
CLOSE custstatecursor ;
//Set first item in list as selected item
this.selectitem (1)
```

6. Select cb_percent from the first drop-down list in the Script view.

   Make sure clicked ( ) returns long [pbm_bnclicked] displays in the second drop-down list box.

7. Enter the following lines for the Clicked event script:

```
Decimal my_result
Double entry_1, entry_2
Int li_int, li_rtn
String sel_state

sel_state=ddlb_state.text
//Get the number of rows with customers from the
//selected states and place in the entry_1 variable.
//Change the first static control to display this
//number.

SELECT count(*) INTO :entry_1 FROM customer
      WHERE customer.state=:sel_state;
st_1.text="Customers in state: " + string(entry_1)

//Get the total number of customers and place in
//the entry_2 variable.
//Change the second static control to display this
//number.
SELECT count(*) INTO :entry_2 FROM customer;
st_2.text="Total number of customers: " &
      + string(entry_2)

//Call uf_percentage and catch its exceptions.
TRY
      my_result = uf_percentage (entry_1, entry_2)
CATCH (exc_no_rows e_nr )
      MessageBox("From exc_no_rows",       &
         e_nr.getmessage())
CATCH (exc_low_number e_ln )
      li_int=1
      MessageBox("From exc_low_number", &
         e_ln.getmessage())
   CATCH (dividebyzeroerror e_zero)
      li_rtn = MessageBox("No Customers", &
         "Terminate Application?", Stopsign!, YesNo!)
      IF li_rtn=1 THEN
         HALT
      END IF
END TRY

//Display the message in the text box. Vary the //message depending on whether
 there is only one
//customer for the selected state or if more than
//one customer resides in selected state.
IF li_int=1 THEN
      sle_result.text ="Value not calculated for " &
         + sel_state + "."    + " Try another state."
ELSE
      sle_result.text = String (my_result) + &
         " % of customers are in " + sel_state + "."
END IF
```

2.10.5 Run the application

Where you are

Add a new sheet window to the existing application

Create user-defined exception objects

Create a new user function and user event

Call the methods and catch the exceptions

> Run the application

You are now ready to run the application and calculate the percentage of customers in a selected state.

You can test the exception conditions you scripted, but to test the divide-by-zero error condition, you need to artificially set the number of customers in the database to zero. You do this by adding a check box to the sheet window, then setting the number of customers to zero if the check box is selected.

In this exercise you:

- Test the new sheet window
- Add a test for the divide-by-zero error

2.10.5.1 Test the new sheet window

1. Click the Run button (Run) in the PowerBar.

   If PowerBuilder prompts you to save changes, click Yes.

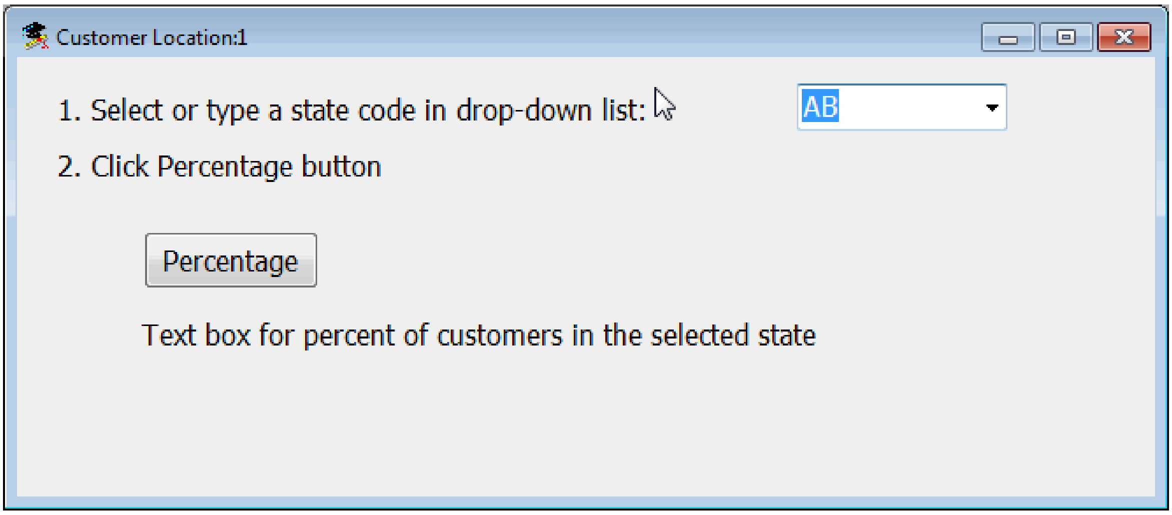
2. Type dba in the User ID box.

   Type sql in the Password box and click OK.

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

3. Select File>Report>Customer Location from the menu bar.

   The Customer Location window displays. The current entry in the drop-down list is AB for Alberta.



Customer Location:1

1. Select or type a state code in drop-down list:

2. Click Percentage button

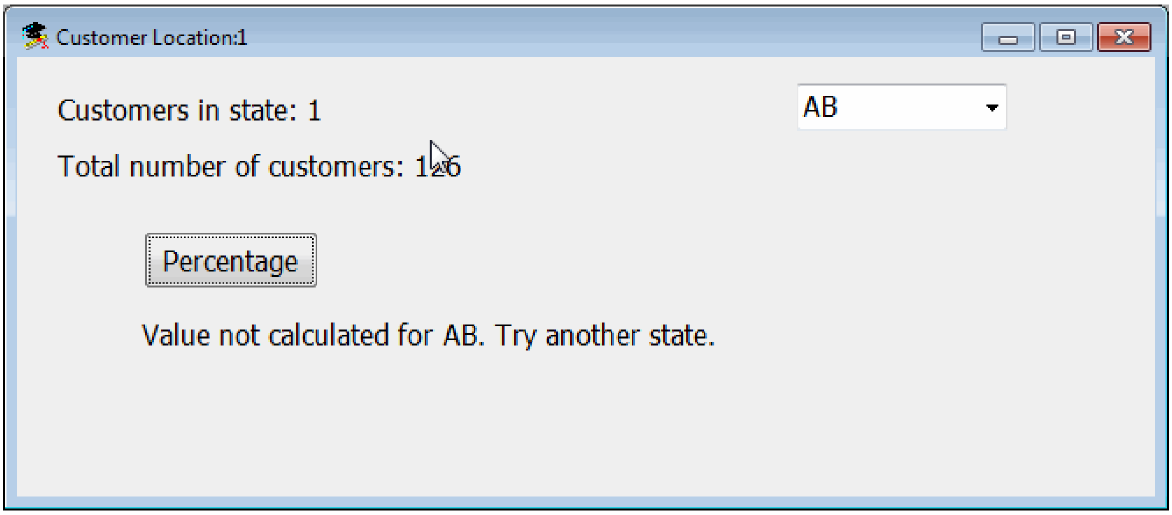
Text box for percent of customers in the selected state

4. Click the Percentage button.

   Because there is only one customer in Alberta, the exc_low_number user-defined exception is thrown. The message from the exception is displayed in a message box that was defined in a CATCH clause in the button Clicked event.

5. Click OK to close the message box.

   The text in the static text boxes now displays the number of customers in Alberta and the total number of customers in the database. The text in the editable text box tells you the value was not calculated and prompts you to select another state.



Customers in state: 1

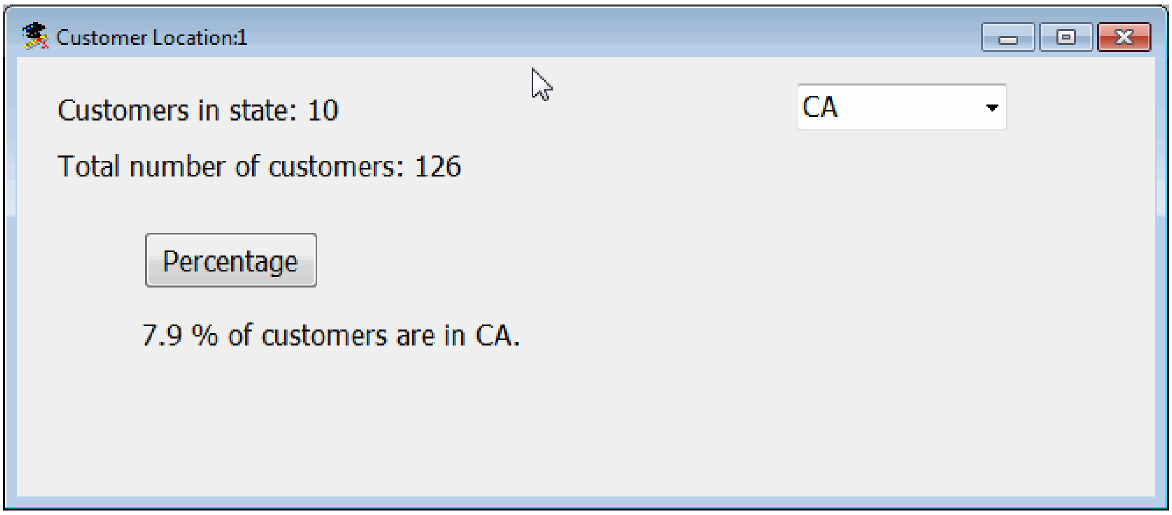
Total number of customers: 126

AB

Value not calculated for AB. Try another state.

6. Select or type CA in the drop-down list box and click the Percentage button.

   The results from the database show 10 customers in California for a total of 7.9% of all customers in the database. The percentage may be different if you have modified the database.



Customers in state: 10

Total number of customers: 126

CA

7.9 % of customers are in CA.

7. Type Ohio into the drop-down list box and click the Percentage button.

   When you lose focus from the drop-down list box by clicking the Percentage button control, the LoseFocus event fires. This event calls the ue_modified event that throws the exc_bad_entry user-defined exception. The exception message tells you to use a two-letter postal code for the state name.

8. Click OK to close the message box, type US in the drop-down list box, and click the Percentage button.

   Because no rows are found in the database with US as the state code, the exc_no_rows exception is thrown. A message displays indicating no rows have been returned and suggests reasons why that might be the case. A more robust application might compare the typed text to a list of state codes and throw the exc_bad_entry exception instead, letting you know that US is not a state code.

9. Click OK to close the message box.

10. Right-click the database icon for the Demo Database, a red and yellow SQL symbol (⚡), in your Windows System Tray.

11. Select Shut down from the pop-up menu, and click Yes in the Warning message box that displays.

    This shuts down the connection to the Demo Database.

12. Select or type AB again in the drop-down list box and click the Percentage button.

    The message from the exc_no_rows exception object displays for Alberta because the connection to the database was closed. To obtain results again, you need to terminate the application and restart it. PowerBuilder reestablishes a connection to the database at runtime when you restart the application.

13. Click OK to close the message box and select File>Exit from the application menu to return to the development environment.

    The Database painter and the Database Profile painter might still list the database connection as being open. In this case you can use either painter to disconnect and reconnect to the database at design time.

2.10.5.2 Add a test for the divide-by-zero error

You now add a check box to the w_cust_pct window. You then write code to force a divide-by-zero error if the check box is selected. Because this test requires an instantiated check box object, you surround the new code in a TRY-CATCH statement that checks for null object errors.

1. Make sure the w_cust_pct window is open in the Layout view.

   Select Insert>Control>CheckBox from the Window painter menu.

2. Click in the window just to the right of the Percentage command button.

3. In the Name box in the Properties view, type cbx_zero.

   In the Text box, type Test divide-by-zero error.

4. Click the Function List tab.

Double-click the uf_percentage function.

5. Type the following text just above the CHOOSE CASE statement:

```
//Set denominator to zero to test error condition
//Numerator unimportant, avoid user exception cases
TRY
IF cbx_zero.checked=TRUE THEN
      ai_totalcust=0
      ai_custbystate=2
END IF
CATCH (nullobjecterror e_no)
      MessageBox ("Null object", "Invalid Test")
END TRY
```

Testing for the null object error

After you finish this lesson, you can test for the null object error by adding the following line above the TRY statement: DESTROY cbx_zero.

6. Click the Run button in the PowerBar.

   If PowerBuilder prompts you to save changes, click Yes.

7. Type dba in the User ID box.

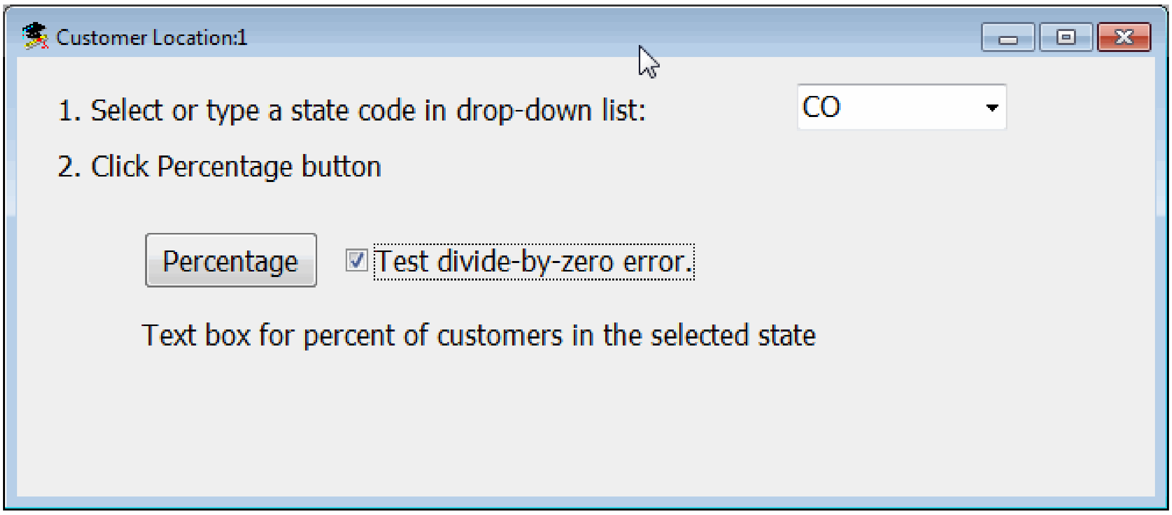
   Type sql in the Password box and click OK.

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

8. Select File>Report>Customer Location from the menu bar.

   Select a state code from the drop-down list box.

9. Select the Test divide-by-zero error check box.



Customer Location:1

1. Select or type a state code in drop-down list: CO

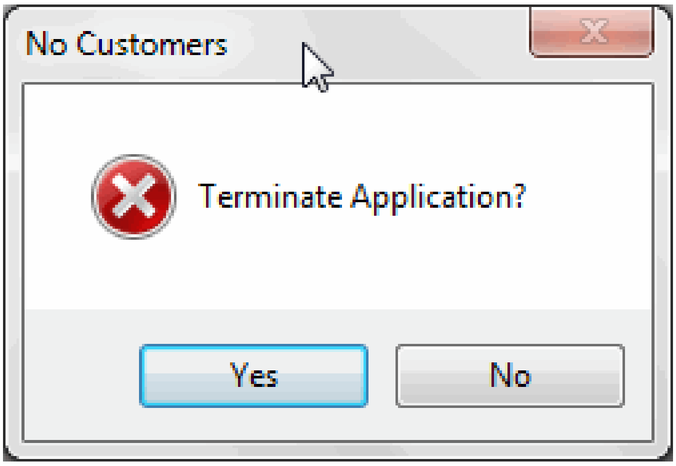
2. Click Percentage button

☑ Test divide-by-zero error.

Text box for percent of customers in the selected state

10. Click the Percentage button.

   The division by zero error is thrown during the percentage calculation and caught by the button Clicked event. The message box that you coded in the CATCH clause for this error displays.



No Customers

Terminate Application?

11.Click No to continue running the application.

Continue to test the application by selecting another state code and optionally clearing the check box.

If the check box is selected when you click the button again and you select Yes in the error message box, the application closes and you return to the development environment.

12.Close the application when you have finished testing it.

2.11 LESSON 11 Preparing the Application for Deployment

In this lesson you 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:

- Create the Project object
- Create the executable file
- Create a shortcut
- Test the executable file

How long does it take?

About 10 minutes.

2.11.1 Create the Project object

Where you are

> Create the Project object

Create the executable file

Create a shortcut

Test the executable file

Now you create the PBTUTOR Project object. You can then use the Project object to create the executable file for the application.

About machine code

When you deploy an application to users, you may want to take advantage of the execution speed of machine code for some computations, such as loops, floating point or integer arithmetic, and function calls. While you are developing the application, you usually use Pcode because it is faster to generate.

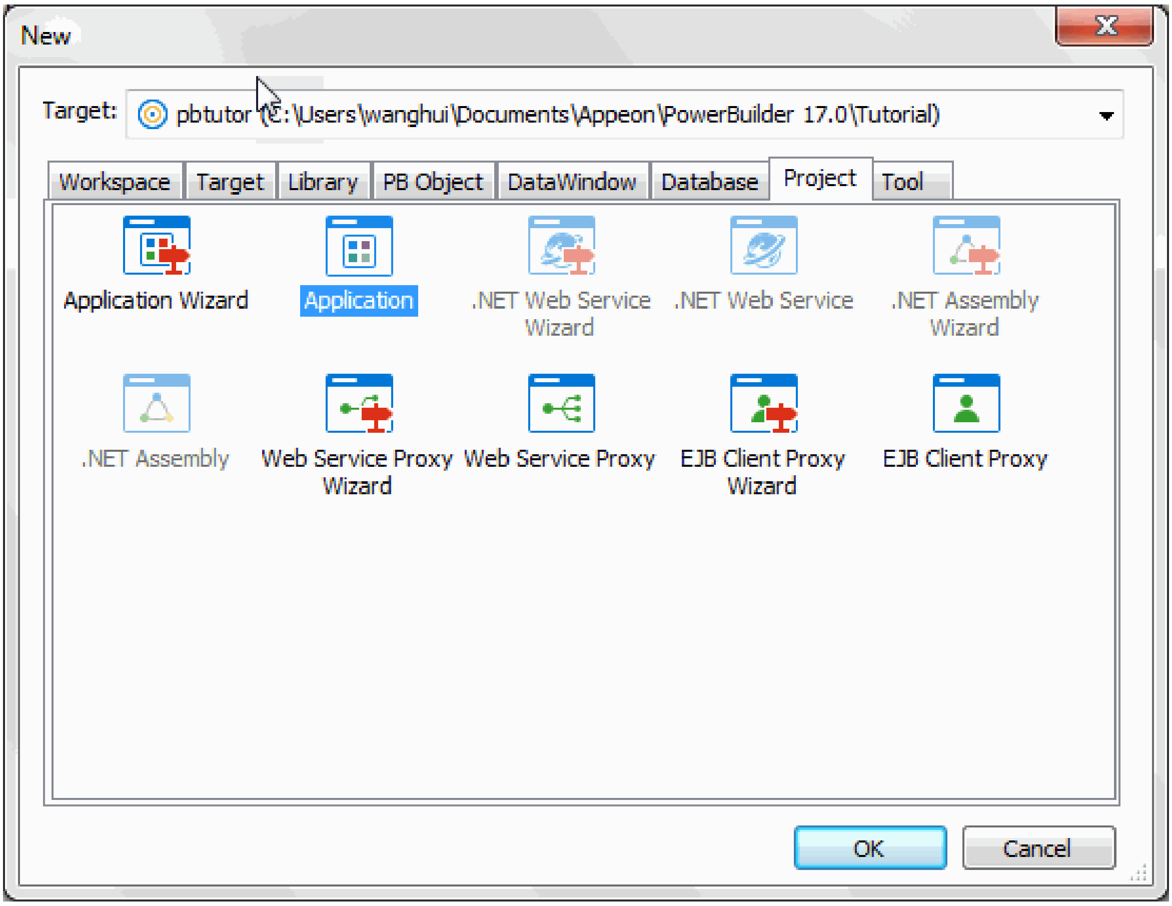
About dynamic libraries

You can also create dynamic libraries for your application. Dynamic libraries can be used to store the objects in the application. 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 that you are working on now, you do not need to use dynamic libraries.

1. Click the New button in the PowerBar.

   Click the Project tab in the New dialog box.



2. Select the Application Wizard icon and click OK.

Using the Project painter

If you clicked the Application icon on the Project page instead of the Application Wizard icon, you open the Project painter. You can make the same selections in the Project painter that you make with the wizard, but the wizard prompts you for this information.

3. Click Next.

   The Specify Destination Library page displays.

4. Select pbtutor.pbl in the Application Libraries list box if it is not already selected.

   Click Next until the Specify Build Options page displays.

   The wizard will generate a project with the following default selections:

Table 2.14:

Wizard property	Default value
Project name	p_pbtutor_exe
Executable filename	pbtutor.exe
Optional resource file	none

5. Select Incremental Build for the Build Option.

   Click Next until the Specify Version Information page displays.

   The wizard will generate a project with the following default selections:

Table 2.15:

Wizard property	Default value
Generate machine code	No
Build dynamic libraries	No

6. If you want to, enter your own version information on the Specify Version Information page.

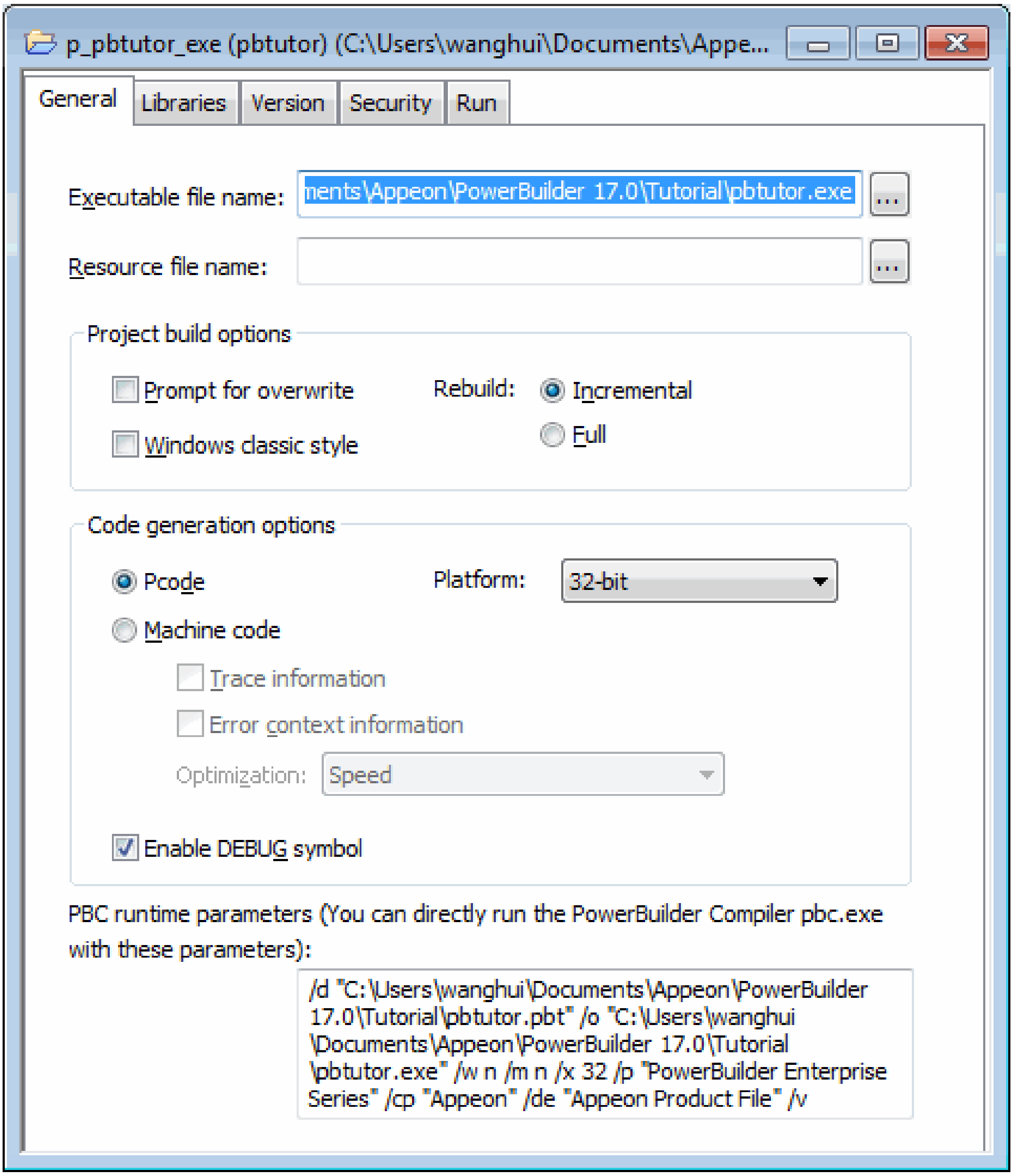
   If you do not change the information on this page, the defaults display in Windows Explorer when you look at the properties of the executable.

7. Click Next.

   Review the information on the Ready to Create Application page.

8. Click the Finish button.

   PowerBuilder creates a Project object for your application and displays it in the Project painter workspace.



Executable file name: ments\Appeon\PowerBuilder 17.0\Tutorial\pbtutor.exe

Resource file name:

Project build options

□ Prompt for overwrite

□ Windows classic style

Rebuild: ● Incremental ○ Full

Code generation options

● Pcode

○ Machine code

□ Trace information

□ Error context information

Optimization: Speed

Platform: 32-bit

☑ Enable DEBUG symbol

PBC runtime parameters (You can directly run the PowerBuilder Compiler pbc.exe with these parameters):

```
/d "C:\Users\wanghui\Documents\Appeon\PowerBuilder
17.0\Tutorial\pbtutor.pbt" /o "C:\Users\wanghui
\Documents\Appeon\PowerBuilder 17.0\Tutorial
\pbtutor.exe" /w n /m n /x 32 /p "PowerBuilder Enterprise
Series" /cp "Appeon" /de "Appeon Product File" /v
```

After a project is defined, you can easily create an executable version of the application. Using a project saves time when you are working on an application that includes dynamic libraries that you expect to rebuild often. Selecting incremental build means that if you make a few changes, you can rebuild your project quickly, rebuilding only files that have changed or files that depend on files that have changed.

2.11.2 Create the executable file

Where you are

Create the Project object

> Create the executable file

Create a shortcut

Test the executable file

Now you create the executable file for your application. The executable file you generate contains definitions for all the objects in the application. For the tutorial application, this includes the bitmap file used in the login window, since you did not include a separate resource file with your project.

You can create the executable in the Project painter, but usually, once you have defined the project, you do not need to open the painter again.

Workspaces and targets in the System Tree have Incremental Build, Full Build, and Deploy items on their pop-up menus that enable you to build and deploy some or all of the projects in a target or in the whole workspace. Incremental Build and Full Build compile your code. Deploy compiles the code and, for applications like the one you built in this tutorial, creates an executable file and optional dynamic libraries.

In this lesson you look at the property sheets where build and deploy options are specified and then create the executable from the PowerBar.

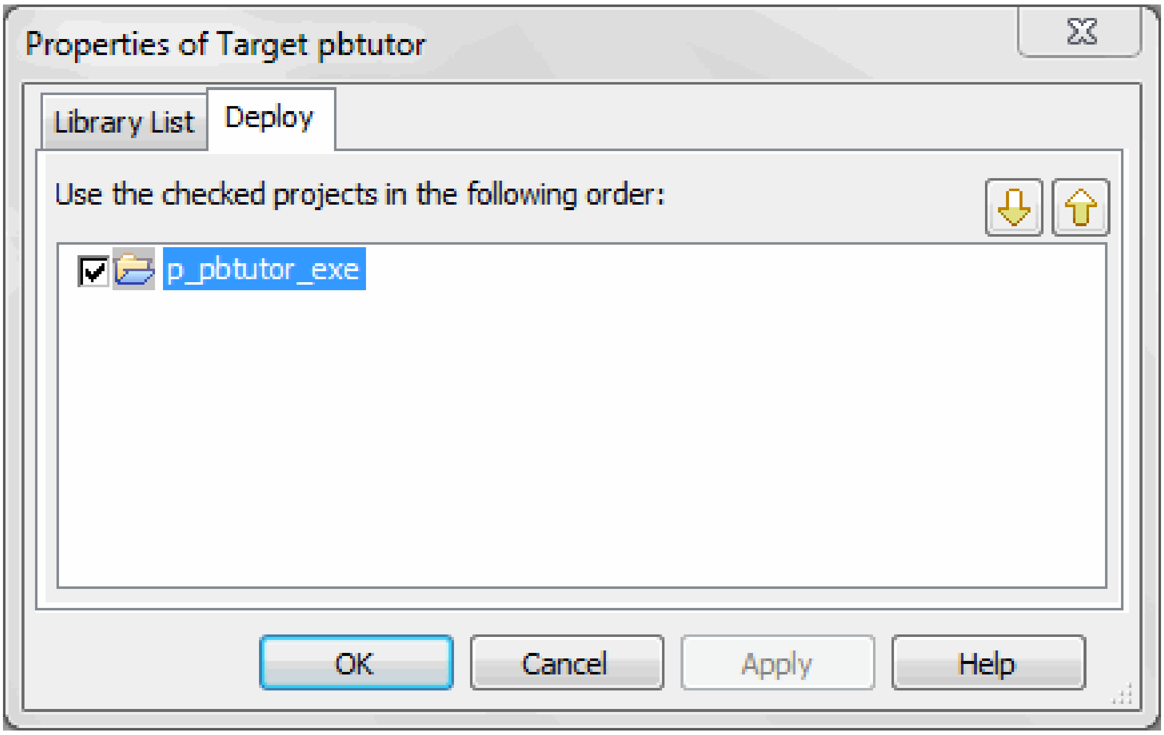
1. Close the Project painter.

   Click Yes if prompted to save changes.

2. Right-click the pbtutor target in the System Tree.

   Select Properties and then select the Deploy tab.

   This page shows all the projects in this target (currently only one).



Properties of Target pbtutor

Use the checked projects in the following order:

☑ p_pbtutor_exe

   If you have more than one project in the target, you can change the order in which they are executed and select which projects you want to build.

3. Leaving p_pbtutor_exe checked, click the Cancel button.

4. Right-click MyWorkspace in the System Tree.

5. Select Properties and select the Deploy Preview tab.

   The Deploy Preview page shows all the targets in your workspace and the projects in each that have been selected for deployment, in the order in which they are to be deployed. You cannot change anything on this page -- you use it to check that you have set up deployment options for your workspace the way you want to. All the projects shown on this page are deployed when you click the Deploy button in the PowerBar.

   This workspace has only one target and only one project, so you can use the Deploy button to create the executable.

6. Click the Cancel button to close the property sheet.

7. Click the Deploy button (Deploy) in the PowerBar.

   The process of creating the executable file might take a few moments. While PowerBuilder is working, you can look at the Output window at the bottom of the screen to see what PowerBuilder is doing.

   If you wanted to stop the deployment process, you could click the Stop button in the PowerBar. When deployment is complete, the Output window displays the following text: Finished Deploy of workspace MyWorkspace.

2.11.3 Create a shortcut

Where you are

Create the Project object

Create the executable file

> Create a shortcut

Test the executable file

Now you create a shortcut for the executable file. The icon serves as a shortcut to open the executable file. You can add the shortcut directly to the desktop or to the program group of your choosing.

1. Minimize PowerBuilder.

   PowerBuilder is minimized to an icon on the taskbar.

2. Right-click on a blank area of the desktop.

   Select New>Shortcut from the pop-up menu.

3. In the Create Shortcut dialog box, click the Browse button and locate pbtutor.exe.

   If you accepted the default installation locations, the file is in %SystemDrive%\Users\[username]\Documents\Appeon\PowerBuilder [version]\Tutorial.

4. Click OK, then click Next.

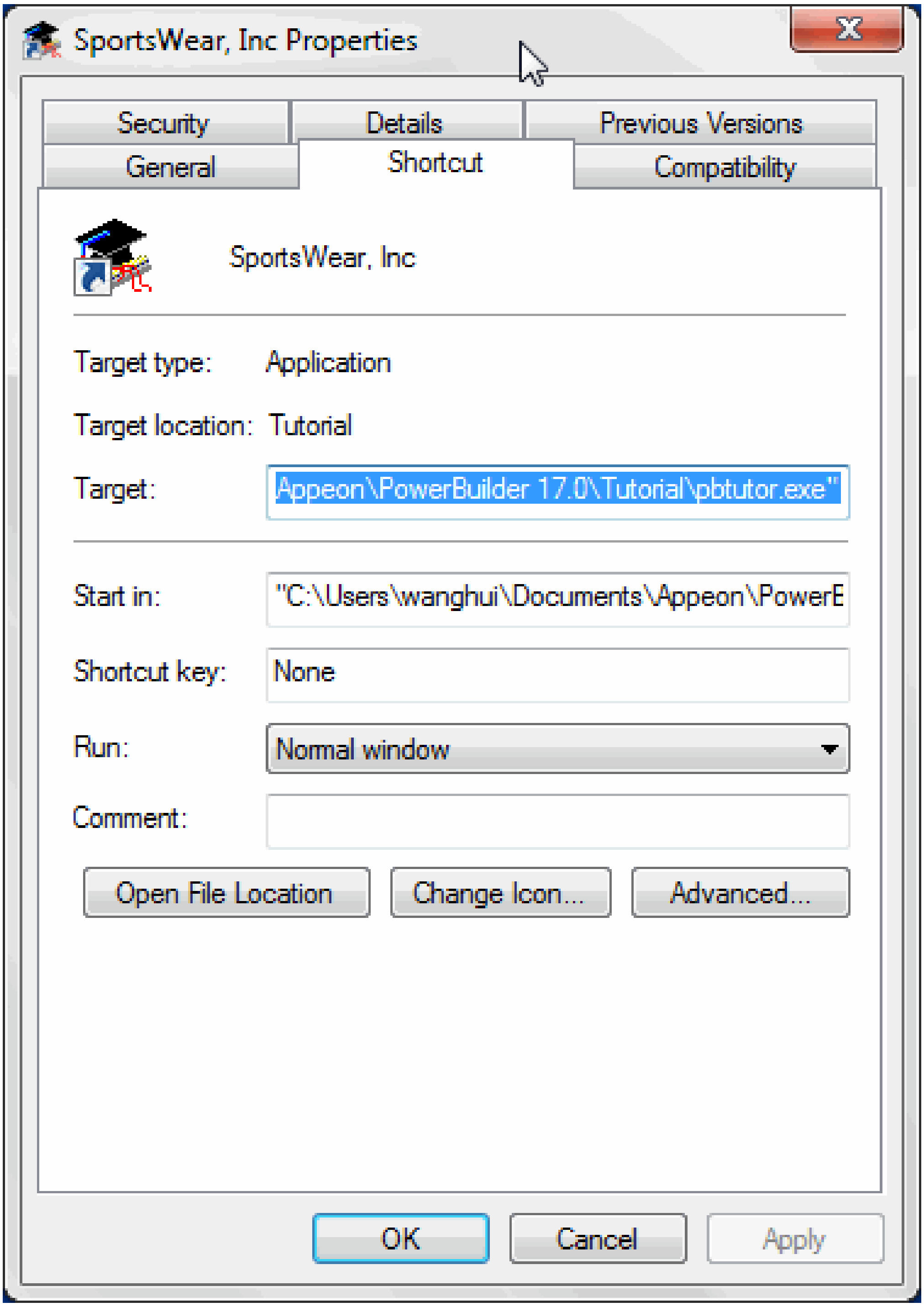
5. Type SportsWear, Inc. as the name of the shortcut.

6. Click Finish to create the shortcut on the desktop.

   Now you must modify a property of the shortcut so that you can run the application. You can also change the icon.

7. Right-click the SportsWear, Inc. icon on the desktop.

   Choose Properties in the pop-up menu.



SportsWear, Inc Properties

SportsWear, Inc

Target type: Application

Target location: Tutorial

Target: Appeon\PowerBuilder 17.0\Tutorial\pbtutor.exe"

Start in: "C:\Users\wanghui\Documents\Appeon\PowerE

Shortcut key: None

Run: Normal window

Comment:

8. Select the Shortcut tab.

Type the path to the PowerBuilder shared modules in the Start In box.

Click OK.

About the location of the shared modules

When you install PowerBuilder Runtime, the installation process puts the DLLs in the %AppeonInstallPath%\Common\PowerBuilder\Runtime [version] directory. The default location is: C:\Program Files\Appeon\Common\PowerBuilder\Runtime [version] (for 32-bit DLLs) and C:\Program Files\Appeon\Common\PowerBuilder \Runtime [version]\x64 (for 64-bit DLLs).

If you want the user to be able to run the application by double-clicking the executable file, you must include this runtime directory in the system environment path.

2.11.4 Test the executable file

Where you are

Create the Project object

Create the executable file

Create a shortcut

> Test the executable file

Now you test the new executable file.

1. Make sure the pbtutor.ini file is in the same directory as the pbtutor.exe executable file.

   The default location of the pbtutor.ini file and the pbtutor.exe file is %SystemDrive% \Users\[username]\Documents\Appeon\PowerBuilder [version]\Tutorial.

2. Double-click the icon for the tutorial application.

   The application begins running.

3. Test the application.

   Notice the changes you made to the customer information.

4. When you have finished testing the application, select File>Exit from the menu bar.

2.11.5 What to do next

Congratulations. You have completed the client-server part of the tutorial. Now you know the basics of application development with PowerBuilder.

The Preface to this book includes a guide to the PowerBuilder documentation. To further your education, you should continue with these books:

Part I, “Users Guide”
Part I, “Application Techniques”
Part I, “DataWindow Programmers Guide”

All the PowerBuilder books are available in the Documentation Center at https://docs.appeon.com.