SYBASE®

User's Guide

PocketBuilder™

2.1

DOCUMENT ID: DC50060-01-0210-01

LAST REVISED: July 2007

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This guide is for programmers building applications with PocketBuilder™. It assumes that:

- You either have read the *Introduction to PocketBuilder* included with the PocketBuilder product or have a basic familiarity with PowerBuilder®. (PocketBuilder uses PowerScript®, the same scripting language as PowerBuilder, with modifications for the Windows Mobile environment.)
- You have a basic familiarity with Windows CE devices. Although your development work in PocketBuilder is done on a desktop machine, you design applications for use on Windows CE and Windows Mobile devices such as the Pocket PC or Smartphone.

  For information about developing applications for Microsoft Windows CE platforms, see the Microsoft Web site at http://msdn2.microsoft.com/en-us/library/ms950422.aspx. You can also find helpful information at the Pocket PC Developer Network Web site at http://www.pocketpcdn.com.

How to use this book

This book gives an overview of the PocketBuilder development environment and explains in detail how to use the PocketBuilder user interface, which allows you to create and deploy applications to a supported Pocket PC device or emulator. It describes basic techniques for building the objects in an application for a handheld device, including windows, menus, DataWindow® objects, and user-defined objects.

Related documents

PocketBuilder documentation The PocketBuilder documentation set also includes the following manuals:

- *Introduction to PocketBuilder* - Provides an overview of PocketBuilder features and the PocketBuilder development environment and a tutorial that leads the new user through the basic process of creating and deploying PocketBuilder applications.
- *Resource Guide* - Presents advanced programming techniques and information about connecting to and synchronizing with a database.

PocketBuilder reference set The PocketBuilder reference set is made up of four manuals that are based on PowerBuilder documentation:

- *Connection Reference* - Describes the database parameters and preferences you use to connect to a database in PocketBuilder.
- *DataWindow Reference* - Lists the DataWindow functions and properties and includes the syntax for accessing properties and data in DataWindow objects.
- *Objects and Controls* - Describes the system-defined objects and their default properties, functions, and events.
- *PowerScript Reference* - Describes syntax and usage for the PowerScript language including variables, expressions, statements, events, and functions.

Online Help Reference information for PowerScript properties, events, and functions is available in the online Help with annotations indicating which objects and methods are applicable to PocketBuilder.

SQL Anywhere® Studio documentation PocketBuilder is tightly integrated with SQL Anywhere (formerly Adaptive Server Anywhere), UltraLite®, MobiLink, and Sybase Central™ which are components of SQL Anywhere Studio. You can install these products from the PocketBuilder setup program. For an introduction to these products, see Chapter 1 in the *Introduction to PocketBuilder*. Documentation for SQL Anywhere Studio is available on the iAnywhere Web site at http://www.ianywhere.com/developer/product_manuals/sqlanywhere/.

Sample applications

The PocketBuilder installation provides a Code Examples workspace with targets that illustrate many of the product's features. Commented text inside events of target objects helps explain the purpose of the sample code. The example workspace is installed in the Code Examples subdirectory under the main PocketBuilder directory.

More applications on the Web

You can find more sample PocketBuilder applications and techniques in the PocketBuilder project on the Sybase® CodeXchange Web site at https://pocketbuilder.codexchange.sybase.com/. There is a link to this page on the Windows Start menu at Program Files>Sybase>PocketBuilder 2.0>Code Samples.

If you have not logged in to MySybase, you must log in to the Sybase Universal Login page to access CodeXchange. If you do not have a MySybase account, you can sign up. MySybase is a free service that provides a personalized portal into the Sybase Web site.

Other sources of information

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

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

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

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase EBFs and software maintenance

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

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.

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

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

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

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
Retrieve and Update	When used in descriptive text, this font indicates: • Command and function names • Datatypes such as integer and char • Database column names such as emp_id and f_name • Keywords such as NULL and TRUE • User-defined objects such as dw_emp or w_main
variable or *file name*	When used in descriptive text and syntax descriptions, oblique font indicates: • Variables, such as *myCounter* • Parts of input text that must be supplied, such as *pklname*.`pkd` • File and path names
File>Save	Menu names and menu items are displayed in plain text. The greater than symbol (>) shows you how to navigate menu selections. For example, File>Save indicates "select Save from the File menu."
`dw_1.Update()`	Monospace font indicates: • Information that you enter in a dialog box or on a command line • Sample script fragments • Sample output fragments

If you need help

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

PART 1

The PocketBuilder Environment

This part describes the basics of using PocketBuilder, understanding and customizing the development environment, and creating workspaces and targets.

CHAPTER 1

Working with PocketBuilder

About this chapter

This chapter describes the basics of working in the PocketBuilder integrated development environment (IDE).

Contents

Before you begin

The PocketBuilder IDE is very similar to the PowerBuilder IDE. Wizards, tools, and application development features in PocketBuilder are based on the same or similar features in PowerBuilder.

For a summary of differences between PowerBuilder and PocketBuilder, see Appendix B, “PowerBuilder and PocketBuilder Product Differences.”

Concepts and terms

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

Workspaces and targets

In PocketBuilder, you work with one or more targets in a workspace. Typically a target is associated with a single application. 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 PocketBuilder target is also referred to as a PowerScript target because it is built using the PowerScript language. PocketBuilder targets have *PKT* extensions to differentiate them from PowerScript targets in PowerBuilder, which have *PBT* extensions. (PowerBuilder Web and JSP targets are not supported in PocketBuilder.)

For more information about creating a workspace and targets, see "Creating and opening workspaces" on page 13 and "Creating a target" on page 15.

Objects

A PocketBuilder application is a collection of objects. PocketBuilder provides many types of objects, including visual objects such as windows, menus, buttons, and graph controls, and nonvisual objects such as the datastore, exception objects, and transaction objects.

As you work to develop a PocketBuilder application, you create new objects and open and modify existing objects.

For more information about creating, opening, and editing objects, see "Working with objects" on page 78.

DataWindow objects

The applications you build are often centered around your organization's data. With PocketBuilder you can define DataWindow objects to retrieve, display, and manipulate data. For more information about DataWindow objects, see Chapter 17, "Defining DataWindow Objects."

PocketBuilder libraries

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

PocketBuilder provides a Library painter for managing your libraries. For information about creating a new library and working with libraries in the Library painter, see Chapter 4, “Working with Libraries.”

Painters and editors

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

PocketBuilder provides painters for windows, menus, DataWindow objects, visual and nonvisual user-defined objects, functions, structures, databases, and the application itself. For each of these object types, there is also a Source editor in which you can modify code directly. See “Working in painters” on page 56 and “Using the Source editor” on page 84.

There is also a file editor you can use to edit any file without leaving the development environment. See “Using the File editor” on page 25.

Events and scripts

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

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

You write scripts in PocketBuilder using PowerScript, an object-oriented scripting language. You can type, drag and drop, or paste script in a Script view of the painter for the object you are working on. Scripts consist of PowerScript functions, expressions, and statements that perform actions or process data and text in response to an event.

The script for a button's Clicked event might retrieve and display information from the database; the script for an edit control's Modified event might evaluate the data and perform processing based on the data.

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

Functions

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

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

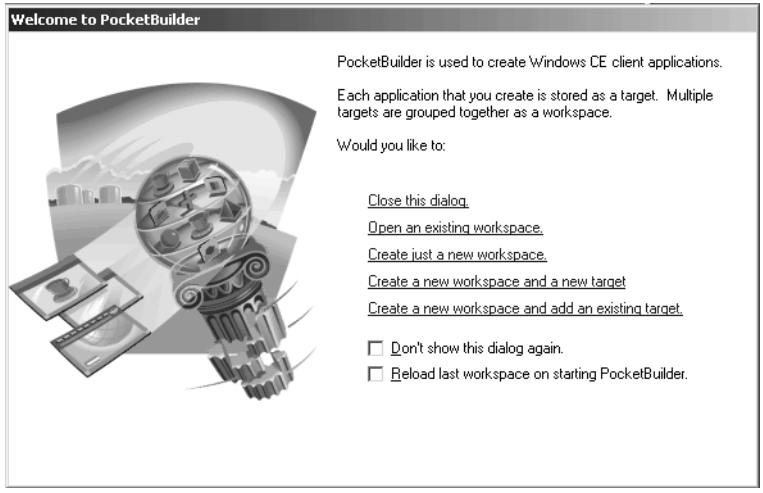
Properties

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

The PocketBuilder environment

When you start PocketBuilder for the first time, the Welcome to PocketBuilder dialog box lets you create a new workspace with or without targets.

Figure 1-1: The Welcome to PocketBuilder dialog box



When PocketBuilder starts, it opens in a window that contains the menu bar and the toolbar (PowerBar) at the top, the System Tree and Clip window on the left, and the Output window at the bottom. The remaining area is for the display of painters and editors that you open when you start working with objects.

The System Tree

The System Tree provides an active resource of programming information that you use while developing targets.

Displaying or hiding the System Tree

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

The System Tree displays the current workspace and all its targets. You can expand each target node to display the library list for the target and all the objects in each target library (PKL). The System Tree works like a tree view in the Library painter, but you can keep it open all the time to serve as the control center of the development environment. It also displays properties, functions, events, structures, and controls, duplicating features in the PocketBuilder Browser.

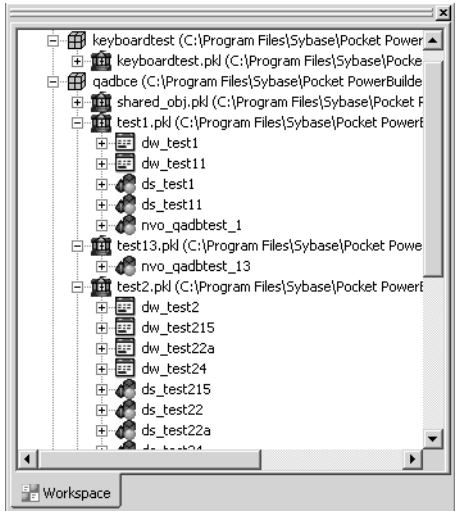
Setting the root in the System Tree

You can set the root of the System Tree to your computer's root directory, the current selection, or any directory or library, as well as to the current workspace. The same capability is available in the Library painter.

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

The following illustration shows two targets, including one that lists multiple libraries in its search path.

Figure 1-2: System Tree showing two targets



You can use the System Tree as the hub of your PocketBuilder session. Pop-up menus let you build and deploy targets, and open and edit any object. The following table lists the actions you can take from the pop-up menu that displays for each item in the System Tree. You can also set properties for each item, choose which object types display in the tree view, change the root of the System Tree, and reset the root to the current workspace.

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

Item	Menu action items
Workspace (PKW)	New (opens New dialog box), Add Target, Open Workspace, Incremental Build, Full Build, Deploy, Debug, Run, Close, Show, Properties.
PowerScript target (PKT)	New, Search, Incremental Build, Full Build, Migrate, Deploy, Debug, Run, Remove Target, Show, Properties.
PocketBuilder library (PKL)	Delete, Search, Optimize, Print Directory, Build Runtime Library, Import, Show, Properties.
PocketBuilder object	Edit, Edit Source, Copy, Move, Delete, Export, Regenerate, Search, Print, Properties. Edit Source is not available for project objects. Inherit and Run/Preview are available only for some object types.

The PowerBar

Default PowerBar buttons

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

Figure 1-3: Default PowerBar buttons

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

Table 1-2: PowerBar buttons and their uses

This PowerBar button	Lets you do this
New	Create new objects.
Inherit	Create new windows, user objects, and menus by inheriting from an existing object.

This PowerBar button	Lets you do this
Open	Open existing objects.
Run/Preview	Run windows or preview DataWindows.
System Tree	Work in the System Tree window, which can serve as the hub of your development session.
Output Window	Examine the output of a variety of operations (migration, builds, deployment, project execution, object saves, and searches). See "The Output window" on page 12.
Next Error, Previous Error	Navigate through the Output window.
To-Do List	Keep track of development tasks you need to do for the current application and use links to get you quickly to the place where you complete the tasks.
Browser	View information about system objects and objects in your application, such as their properties, events, functions, and global variables, and copy, export, or print the information.
Clip Window	Store objects or code you use frequently. You can drag or copy items to the Clip window to be saved and then drag or copy these items to the appropriate painter view when you want to use them. See "The Clip window" on page 11.
Library	Manage your libraries using the Library painter.
DB Profile	Define and use named sets of parameters to connect to a particular database.
Database	Maintain databases and database tables, control user access to databases, and manipulate data in databases using the Database painter.
Edit	Edit text files (such as source, resource, and initialization files) in the File editor.
Launch Emulator	Open a dialog box that lets you select the emulator that you want to start up. You can also start the emulator manager from this dialog box.
Incremental Build Workspace	Update all the targets and objects in the workspace that have changed since the last build.
Full Build Workspace	Update all the targets and objects in the workspace.
Deploy Workspace	Deploy all the targets in the workspace.
Skip	Interrupt a build, deploy, or search operation. When a series of operations is in progress, such as a full deploy of the workspace, the Skip button lets you jump to the next operation.
Stop	Cancel all build, deploy, or search operations.

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

Customizing the PowerBar

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

For more information, see “Using toolbars” on page 39.

About PowerTips

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

The Clip window

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

Using the Clip window

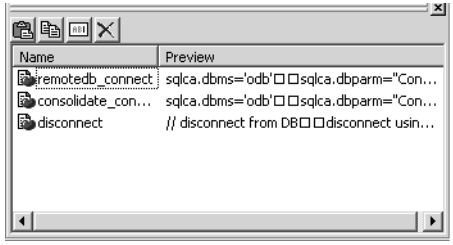
The Clip window displays a list of named clips and a preview of the information contained in each clip. It provides buttons to move Clip window contents to the clipboard, copy clipboard contents to the Clip window, rename a clip, and delete a clip. Clips you save in one workspace are available in all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

For example, you might have standard error-checking code that you want to reuse on a regular basis. You can copy it to the clipboard by highlighting the code in Script view and selecting Copy from the pop-up menu. In the Clip window, click the Paste icon, and name the clip.

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

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

Figure 1-4: Clip window with three clip entries



The Output window

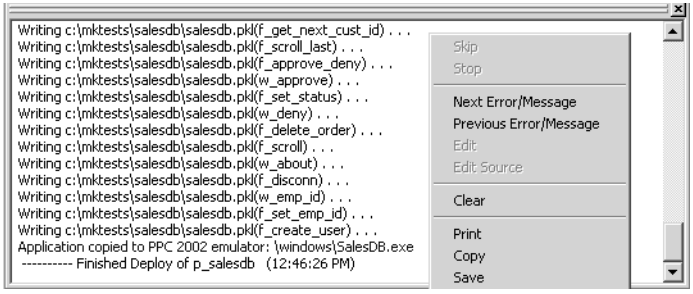
The output of a variety of operations (migration, builds, deployment, project execution, object saves, and searches) displays in the Output window. You control operations in the window using the Skip, Stop, Next Error, and Previous Error buttons or menu options.

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

Using the Output window

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

Figure 1-5: Output window with deployment output



Creating and opening workspaces

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

Creating a workspace

❖ **To create a new workspace:**

1 Do one of the following:

- Click the New button in the PowerBar
- Select File>New from the menu bar
- In the System Tree, right-click the workspace name and select New from the pop-up menu

The New dialog box displays.

2 In the Workspace page of the New dialog box, select Workspace.

The New Workspace dialog box displays.

3 Enter a name for the workspace you want to create and click Save.

The workspace is created and the name of the new workspace displays in the PocketBuilder title bar. Workspaces have the extension *PKW*.

Opening a workspace

By default, the last workspace you open in a PocketBuilder session is opened automatically in the next session. You can change this behavior by modifying options on the Workspaces tab of the System Options dialog box or on the Welcome to PocketBuilder screen. For example, you can have PocketBuilder open not only the workspace, but also the objects and scripts you worked on last. See “Starting PocketBuilder with an open workspace” on page 31.

When PocketBuilder opens with an existing workspace, it displays the name of the current workspace in the title bar. The current workspace is also displayed in the System Tree. Although you can create multiple workspaces in a single instance of PocketBuilder, you can have only one workspace open at a time. You can change workspaces at any time. You can also run multiple instances of PocketBuilder simultaneously.

❖ **To change workspaces:**

1 Do one of the following:

- Select File>Open Workspace from the menu bar
- In the System Tree, right-click on the workspace name and select Open Workspace from the pop-up menu

The Open Workspace dialog box displays.

2 From the list, select the workspace you want to open.

The workspace is changed and the name of the new workspace displays in the PocketBuilder title bar.

❖ **To change the workspace to a recent workspace:**

- Select File>Recent Workspaces from the menu bar and select the workspace

The workspace list includes the eight most recently accessed workspaces. You can change this by selecting Tools>System Options and modifying the number of items.

Using wizards

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

About wizards

Wizards simplify the creation of applications and objects. Using your specifications, wizards can create multiple objects and in some cases automatically generate complex code that you can modify as needed. In most wizards, the first page explains what the wizard builds. If you need help with a field on a wizard page, place the cursor in that field and press F1.

You start wizards from the New dialog box, but not all the icons in the New dialog box represent wizards. For example, the Application icon on the Project page of the New dialog box opens the Project painter, where you can enter required values without being prompted by the Application Project wizard.

Related wizards

The PocketPC and SmartPhone Application Creation wizards on the Target page of the New dialog box can launch the Connection Object wizard and the Application project wizard. If you make selections to create a connection object and a project in one of the Application Creation wizards, the Connection Object and Application project wizards are started automatically.

You can also complete the Application Creation wizards without launching secondary wizards. If your application requires a SQL connection object, you can create one later with the Connection Object wizard on the PBObjects page of the New dialog box. You can create a project later with the Application wizard on the Project page of the New dialog box.

Creating a target

When you create a target, you are prompted for the name and location of a target (PKT) file and one or more other objects. A target file is an ANSI text file that contains information about the target.

❖ **To create a new target:**

1 Do one of the following:

- Click the New button in the PowerBar
- Select File>New from the menu bar
- In the System Tree, highlight the workspace name and select New from the pop-up menu

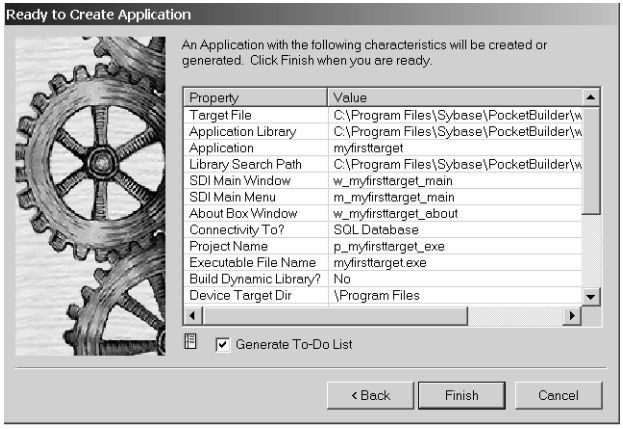
The New dialog box opens.

2 On the Target tab page, select either the Pocket PC Application Creation target wizard or the SmartPhone Application Creation target wizard.

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

You can review your choices on a summary page that displays when you have finished entering information.

Figure 1-6: Summary page of a target creation wizard



Be sure the Generate To-Do List check box is selected if you want the wizard to add items to the To-Do List to guide and facilitate your development work.

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

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

As you develop the application, you can use linked items on the To-Do list to open an object in the specific painter and view where you need to work. See “Using the To-Do List” on page 23.

PowerScript application targets

You can create only Single Document Interface (SDI) applications for Windows CE. Although you can import a PowerBuilder application target into PocketBuilder, you must remove Multiple Document Interface (MDI) windows from a PowerBuilder application before you import it to PocketBuilder for use on Windows CE devices.

For information on importing PowerBuilder targets, see “Import Desktop to Pocket” on page 17 and “Converting a PowerBuilder application” on page 778.

The Target page of the New dialog box in PocketBuilder has wizards that you can use to create or convert PowerScript targets and to add the targets to a PocketBuilder workspace:

- Application Target wizard
- Application Creation Target wizards
- Existing Application Target wizard
- Import Desktop to Pocket

Application Target wizard

The Application Target wizard creates an Application object and the library (PKL) and target (PKT) containing it.

Application Creation Target wizards

The PocketPC and SmartPhone Application Creation Target wizards create an Application object, a PKL, and a PKT, as well as a set of basic objects and scripts. If the application requires a connection to a SQL database, the wizard automatically creates a connection object and integrates it into the application.

Existing Application Target wizard

The Existing Application Target wizard adds a target to your workspace that uses an existing application. In the wizard, you must select an Application object in a PKL file.

After you complete the wizard, the Migrate Application dialog box opens if you need to migrate the application for compatibility to the current PocketBuilder version.

Before you migrate
Always make a backup copy of all the PKL files used in an application before you migrate it or convert it to the current version of PocketBuilder.

You should check the PocketBuilder *Release Bulletin* for the version you are using to find out if there are migration issues that might affect you.

Import Desktop to Pocket

You can select a PowerScript target that was built in PowerBuilder 7, 8, 9, or 10 and convert it to a PocketBuilder target. The Import Desktop to Pocket wizard copies and converts into PKL files all the PBL files in the target you select on the first page of the wizard. The wizard also references the PKL files in a new target with a PKT extension.

The original target and libraries are left in their original directories, which is also where the new target and migrated libraries are saved. The conversion wizard gives you the option of adding the new target to the current PocketBuilder workspace.

Files converted by the wizard keep the same names as the original files except for their extensions. The PowerBuilder VM for your source files (PBT and PBL files) must be in your system's PATH variable. The wizard uses the *PBVMxx.DLL* and the compiler *PBCMPxx.DLL* to convert the source files, where *xx* is the version of the PowerBuilder VM.

Generating an application that is not in the current workspace can cause compile errors during the conversion of PBL files to PKL files. However, once the target has been added to a workspace, a full rebuild fixes everything unless there are true compile errors in the library you are converting.

For additional considerations in converting a PowerBuilder application to a PocketBuilder application, see "Converting a PowerBuilder application" on page 778.

A wizard is available on the Tool page of the New dialog box for converting PocketBuilder targets to PowerBuilder targets. For information, see "Converting a PocketBuilder target to PowerBuilder" on page 26.

Managing workspaces

The tasks you might need to perform to manage a workspace include adding and removing targets, and specifying workspace properties.

Adding an existing target to a workspace

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

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

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

The Add Target to Workspace dialog box displays.

2 Navigate to the directory containing the target you want to add and select the target file.

3 Click Open.

The target is added to your current workspace.

Removing a target from a workspace

When you remove a target from a workspace, the target file is not deleted from the hard drive.

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

1 Right-click on the target displayed in the System Tree.

2 Select Remove Target from the pop-up menu.

Specifying workspace properties

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

❖ **To specify workspace properties:**

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

2 Select the Targets or Deploy Preview tab page.

3 Specify the properties as described in the following sections.

Specifying target order

You can specify the targets and the order in which those targets are built or deployed on the Targets tab page. All the targets identified with the workspace are listed. Check the targets you want to include in the workspace build or deployment. Use the arrows to change a target's position in the target order list.

Previewing deployment

You can verify the targets and the order in which those targets are built or deployed on the Deploy Preview tab page. To make changes, you need to use:

- The Targets page in the Properties of Workspace dialog box to change the targets selected for deployment
- The Deploy page on the target's properties dialog box to specify which projects in a PowerScript target are built

Building workspaces

You can build and deploy workspaces from within PocketBuilder or from a command line. For information on working from a command line, see the *Resource Guide*.

Build and deploy options

In the development environment, you can specify how you want the targets in your workspace to be built and deployed. You can then build individual targets or all the targets in the workspace.

Table 1-3: Building and deploying targets and workspaces

To do this	Do this
Set deployment options for a target	Select Properties from the pop-up menu for the target and select the Deploy tab. Select the check box next to a project to build it when you deploy the target. Use the arrows to set the order in which projects are built. Set options for each project in the target in the Project painter.
Set build and deployment options for the workspace	Select Properties from the pop-up menu for the workspace and select the order in which targets should be built. On the Deploy Preview page, you can check which projects and deploy configurations are currently selected.
Build, migrate, or deploy a selected target	Select Incremental Build, Full Build, Migrate, or Deploy from the pop-up menu for the target. Deploy builds the projects in the target in the order listed on the Deploy page of the target properties dialog box.
Build or deploy all the targets in the workspace	Select Incremental Build, Full Build, or Deploy from the pop-up menu for the workspace, from the Run menu, or from the PowerBar.

Managing databases

PocketBuilder installs with SQL Anywhere Studio tools that you can use to manage your application databases. SQL Anywhere Studio includes the SQL Anywhere (formerly Adaptive Server Anywhere) and UltraLite relational database systems, and the MobiLink synchronization server.

PocketBuilder supports SQL Anywhere databases through ODBC. With MobiLink you can synchronize data between a SQL Anywhere or UltraLite database and an enterprise database that uses SQL Anywhere or a different database management system, such as Oracle, Microsoft SQL Server, IBM DB2, or Sybase Adaptive Server Enterprise.

MobiLink can use a variety of protocols including TCP/IP, HTTP, HTTPS, and Microsoft ActiveSync to synchronize the consolidated database with the remote database on the handheld device. Synchronization can be bidirectional.

For information about MobiLink support in PocketBuilder, see Chapter 16, "Managing the Database," and the chapter on MobiLink synchronization in the *Resource Guide*.

What you can do

At design time, you use the Database painter in PocketBuilder to access SQL Anywhere or UltraLite databases and perform the following operations:

- Modify local table and column properties
- Retrieve, change, and insert data
- Create new local tables or modify existing tables
- Create a new database or delete an existing database

For information about using the Database painter, see Chapter 16, "Managing the Database."

Setting the database connection

When you open a painter that communicates with the database (such as the Database painter or DataWindow painter), PocketBuilder connects you to the database you used last, if you are not already connected. If the connection to the default database fails, the painter still opens.

If you do not want to connect to the database you used last, you can deselect the Connect to Default Profile option in the Database Preferences dialog box.

Changing the database connection

You can change to a different database at any time. You can have several database connections open at a time, although only one connection can be active. The database components for each open connection are listed in the Objects view of the Database painter.

The Database painter title bar displays the number of open connections and indicates which is active. The title bar for each view displays the connection with which it is currently associated. You can change the connection with which a view is associated by dragging the profile name for a different connection onto the view.

Working with tools

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

- Click a button for the tool you want in the PowerBar
- Select the tool from the Tools menu
- Open the New dialog box and select the tool you want on the Tool tab page

Tools in the PowerBar

Table 1-4 lists the tools available in the PowerBar. Some of these tools are also listed on the Tools menu.

Table 1-4: Tools available in the PowerBar

Tool	What you use the tool for
To-Do List	Keep track of development tasks you need to do for the current target and create links to get quickly to the place where you need to complete the tasks. For information, see "Using the To-Do List" on page 23.
Browser	View information about system objects and objects in your PowerScript target, such as properties, events, functions, and global variables, and copy, export, or print the information. For information, see "Browsing the class hierarchy" on page 277.
Library painter	Manage libraries, create a new library, and build dynamic libraries.
Database profiles	Define and use named sets of parameters to connect to a particular database. For information, see the PocketBuilder *Resource Guide*.
Database painter	View and manage database connections for your applications. For information, see Chapter 16, "Managing the Database."
File editor	Edit text files such as source, resource, and initialization files. For information, see "Using the File editor" on page 25.
Launch emulator	Open a dialog box from which you can launch an emulator from a cold boot or with a saved state file.
Debugger	Set breakpoints and watch expressions, step through your application, examine and change variables during execution, and view the call stack and objects in memory. For information, see Chapter 25, "Testing and Debugging Applications."

Tools in the New dialog box

Table 1-5 lists the tools you can launch from the Tool tab page in the New dialog box. The Library painter and File editor can also be launched from the Tools menu and the toolbar.

Table 1-5: Tools available in the New dialog box

Tool	What you use the tool for
File Editor	Edit text files such as source, resource, and initialization files. For information, see “Using the File editor” on page 25.
Library Painter	Manage libraries, create a new library, and build dynamic libraries.
DataWindow Syntax	Report on properties of DataWindow objects and the controls within DataWindow objects. For information, see DataWindow Syntax online Help.
Profiling Class View, Profiling Routine View, and Profiling Trace View	Use trace information to create a profile of your application. For information, see Chapter 27, “Packaging and Distributing an Application.”
Export Pocket to Desktop	Convert PocketBuilder libraries to PowerBuilder libraries. For information, see “Converting a PocketBuilder target to PowerBuilder” on page 26.
Manage Certificates and Device Certificate Management Tool	Manage certificates that you can use to digitally sign PocketBuilder applications and CAB files. For information, see “Managing certificates on the desktop” on page 732 and “Managing certificates on a connected device” on page 734.
Code Signing Wizard	Attach a digital signature to a file that you want to deploy.
Enhanced CAB Generation Tool	Use to build CAB files containing all required application components. For infomation, see “Building CAB files with the Enhanced CAB Generation tool” on page 740.

Using the To-Do List

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

To-Do List entries

The entries on the To-Do List are created in two ways:

- Automatically by some PocketBuilder wizards to guide you through the continued development of objects of different types that you will need to build the application or component specified by the wizard
- By you at any time when you are working in a painter and want to create a link to a task you want to remember to complete

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

Exporting and importing lists

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

Linked entries
If you import a list from another workspace or target, linked entries display in the list, but the links are not active.

Working with entries on the To-Do List

Table 1-6 tells you how to work with entries on the To-Do List.

Table 1-6: Using the To-Do List

To do this	Do this
See linked entries	Move the pointer over the entries. A hand displays when the entry you are over is linked.
Use a linked entry to get to a painter or wizard	Double-click the linked entry, or select it and then select Go To Link from the pop-up menu.
Add an entry with no link	Select Add from the pop-up menu.
Add a linked entry to a painter that edits objects	With the painter open, select Add Linked from the pop-up menu.
Add an entry for a specific target	If the To-Do List is open, select the target from the drop-down list at the top of the To-Do List and add the entry. If the To-Do List is closed, select a target in the System Tree, open the To-Do List, and add the entry.

To do this	Do this
Add an entry for the workspace	Select Current Workspace from the drop-down list at the top of the To-Do List and add the entry.
Change the list that displays	Select a specific target or Current Workspace from the drop-down list at the top of the To-Do List. To display tasks for all targets and the workspace, select All Items.
Change an entry's position on the list	Drag the entry to the position you want.
Edit or delete an entry	Select Edit or Delete from the pop-up menu.
Delete checked entries or all entries	Select Delete Checked or Delete All from the pop-up menu.
Check or uncheck an entry	Click in the margin to the left of the entry or select an entry and then select Check/Uncheck from the pop-up menu.
Export a To-Do List	Select Export from the pop-up menu, name the To-Do List text file, and click Save.
Import a To-Do List	Select Import from the pop-up menu, navigate to an exported To-Do List text file, and click Open.

Using the File editor

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

Setting file editing properties

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

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

Dragging and dropping text

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

Converting a PocketBuilder target to PowerBuilder

Export Pocket to Desktop wizard

You can convert a PocketBuilder target to a PowerBuilder target with the Export Pocket to Desktop wizard. The wizard determines which version of PowerBuilder you are using from your system's PATH variable, then copies and converts into PBL files all the PKL files in the target you select on the first page of the wizard. The wizard also references the PBL files in a new target with a PBT extension.

Creating ANSI or Unicode libraries
If you are converting a PocketBuilder target for use in PowerBuilder 10 or higher, you must select the Create Unicode Libraries check box on the main page of the wizard. You must not select this check box if you want to convert the target to a PowerBuilder 8 or 9 target.

Files converted by the wizard keep the same names as the original files except for their extensions. The PowerBuilder VM must be in your machine's PATH variable in order for the conversion to proceed. The wizard uses the *PBVMxx.DLL* and the compiler *PBCMPxx.DLL* to convert the PocketBuilder source files, where *xx* is the version of the PowerBuilder VM.

The original target and libraries are left in their original directories, which is also where the new target and migrated libraries are saved. It is good practice to do a full rebuild after you add a target converted by the Export Pocket to Desktop wizard to a PowerBuilder workspace.

Using online Help

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

About HTML Help

HTML Help includes the PocketBuilder books: the *Introduction to PocketBuilder*, this book (the *User's Guide*), and the *Resource Guide*.

About Windows Help

Windows Help contains context-sensitive reference information from the following books: *Objects and Controls*, the *DataWindow Reference*, the *PowerScript Reference*, and the *Connection Reference*. It also provides context-sensitive help for dialog boxes and the wizards and tools in the New dialog box.

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

Accessing Help

Table 1-7 lists the ways you can access Help.

Table 1-7: Accessing online Help

Action	What it does
Use the Help menu on the menu bar	Displays the Help contents or Help for the current painter
In a wizard, press F1	Displays Help for the field in the wizard that has focus
In the Properties view in a painter, select Help from the pop-up menu on any tab page	Displays a Help topic from which you can get Help on the properties, events, and functions for the object or control whose properties are displayed in the Properties view
Add a Help button to the PowerBar and use it	Displays the Help contents
Press F1	Displays the Help contents
Press Shift+F1 in the Script view or Function painter	Displays context-sensitive Help about the function, event, or keyword under the cursor
Select Help from the pop-up menu in the Browser	Displays Help for the Browser or for the selected object, control, or function
Click the Help button in a dialog box	Displays information about that dialog box
Click the link to HTML Help icon in a Windows Help topic	Opens the HTML Help at the linked topic

Technical Library CD and Web site

PocketBuilder books are also provided on a Sybase Technical Library CD and the Sybase Product Manuals Web site. For more information, see “Other sources of information” on page xxiii.

Building an application

This section describes the basic steps you follow when building an application for deployment to a Windows CE device. After completing step 1, you can work in any order, defining the objects used in your application as you need them.

❖ **To build an application for deployment to a Windows CE device:**

1 Create the application using a target wizard from the New dialog box and specify the library list for the application.

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

See Chapter 3, “Working with PowerScript Targets,” and Part 3, “Coding Fundamentals.”

2 Create windows.

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

See Chapter 10, “Working with Windows.”

3 Create menus.

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

See Chapter 13, “Working with Menus.”

4 Create user objects.

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

See Chapter 14, “Working with User Objects.”

5 Create functions, structures, and events.

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

See Chapter 7, “Working with User-Defined Functions,” Chapter 8, “Working with User Events,” and Chapter 9, “Working with Structures.”

6 Create DataWindow objects.

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

See Part 5, “Working with Databases and DataWindows.”

7 Test and debug your application.

You can run your application on the desktop at any time. If you discover problems, you can debug your application on the desktop by setting breakpoints, stepping through your code, and looking at the values of variables during execution.

See Chapter 25, “Testing and Debugging Applications.”

You can also create a trace file when you run your application and use PocketBuilder’s profiling tools to analyze the application’s performance and logical flow.

See Chapter 26, “Tracing and Profiling Applications.”

8 Prepare an executable.

When your application is complete, you prepare an executable version to deploy to handheld devices and distribute to your users.

See Chapter 27, “Packaging and Distributing an Application.”

CHAPTER 2

Customizing PocketBuilder

About this chapter

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

Contents

Starting PocketBuilder with an open workspace

When you start a new PocketBuilder session, you can automatically display the last workspace from the previous session. You can even display the painters you had open in the previous session, and open a Script view to the code you were last working on.

Changing start-up options

Options on the Workspaces page of the System Options dialog box allow you to determine what displays when you start PocketBuilder.

❖ **To change start-up options for the development environment:**

1 Select Tools>System Options from the menu bar.

2 Click the Workspaces tab and select start-up display options as described in Table 2-1.

Table 2-1: Start-up options for PocketBuilder

To open PocketBuilder with	Do this
The last workspace you were working on	Select the Reopen Workspace on Startup check box.
The last workspace, and the painters and editors you were using	Select the Reopen Workspace on Startup and the Reload Painters When Opening Workspace check boxes.
No workspace	Clear the Reopen Workspace on Startup check box. If you also select the Reload Painters When Opening Workspace check box, any painters and editors that were open when you last closed a workspace display when you open that workspace. However, the workspace does not automatically display on start-up.
The Welcome dialog box	Select the Show Start Dialog at Startup with no Workspace check box. You must also clear the Reopen Workspace on Startup check box.

Opening PocketBuilder from Windows Explorer
You can double-click a workspace file in Windows Explorer to open PocketBuilder with that workspace displayed. PocketBuilder workspaces have a *PKW* extension.

Changing the design-time layout

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

- Using the Windows XP style for visual controls
- Showing or hiding the System Tree, Output, and Clip windows and changing their locations
- Showing or hiding views in painters and changing their locations

You can also show or hide toolbars, change their locations, and add custom buttons. See “Using toolbars” on page 39.

Using the Windows XP style for visual controls

The PocketBuilder development environment uses the Windows XP style for visual controls when it runs on a Windows XP system and you have chosen the Windows XP visual style for the display of controls. Visual controls using this style include windows, scroll bars, and title bars. The controls in a window or visual user object display with the Windows XP style in the Layout view of the object painter, and when you preview the window or run the application in the development environment.

Arranging the System Tree, Output, and Clip windows

Hiding and moving windows

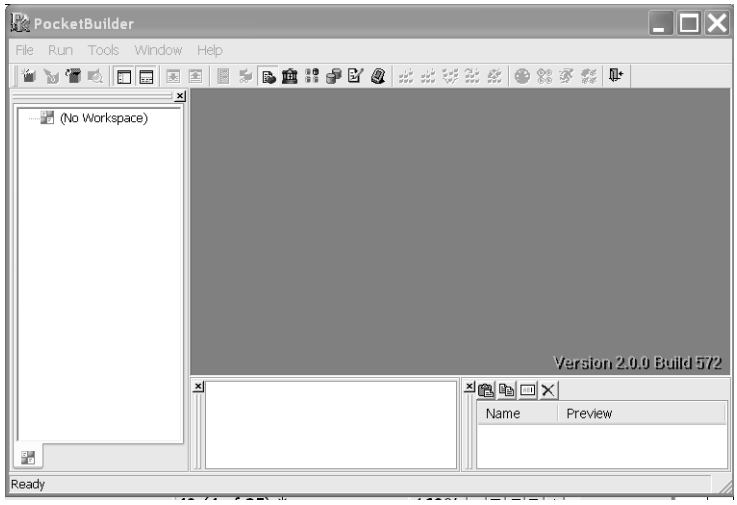
You can hide the System Tree, Output, and Clip windows at any time by clicking their buttons on the PowerBar.

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

Using the full width or height of the main window

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

Figure 2-1: Result of clearing Horizontal Dock Windows Dominate



Using views in painters

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

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

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

Each pane has:

- A title bar you can display temporarily or permanently
- A handle in the top-left corner you can use to drag the pane to a new location
- Splitter bars between the pane and each adjacent pane

Displaying the title bar

Because it is not often necessary and takes up valuable screen real estate, a title bar does not permanently display at the top of a pane for most views, but you can display a title bar for any pane either temporarily or permanently.

❖ **To display a title bar:**

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

The title bar displays.

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

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

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

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

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

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

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

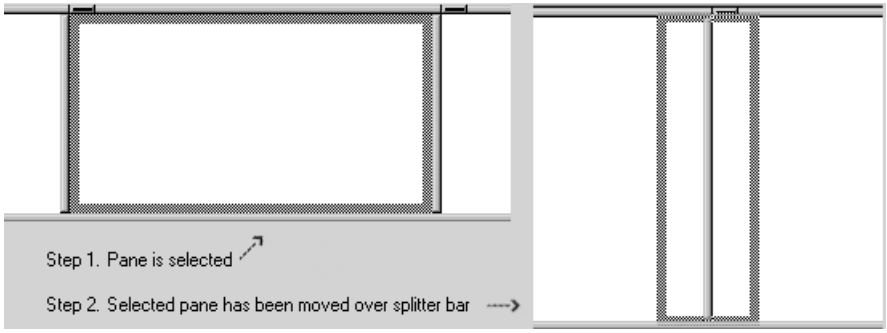
Moving and resizing panes and views

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

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

The outline changes size as you drag it. When the pointer is over the middle of a pane, the outline fills the pane. As you drag the pointer toward any border, the outline becomes a narrow rectangle adjacent to that border. When the pointer is over a splitter bar that separates two panes, rows, or columns, the outline straddles the splitter bar.

Figure 2-2: Moving a pane in the workspace area



❖ **To move a pane:**

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

A gray outline appears around the pane.

2 Drag the outline to the new location.

When you move the pointer to a corner
When you move the pointer to a corner, you have many places where you can drop the outline. To see your options, move the pointer around in all directions in the corner and see where the outline displays as you move it.

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

Table 2-2: Where you can move a painter view (pane)

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

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

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

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

❖ **To resize a pane:**

- Drag the splitter bars between panes

Floating and docking views

By default, panes are docked within a painter window, but some tasks might be easier if you float a pane. A floating pane can be moved outside the painter's window or even outside the PocketBuilder window.

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

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

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

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

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

❖ **To dock a floating view:**

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

Adding and removing views

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

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

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

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

2 Move the pane where you want it.

For how to move panes, see "Moving and resizing panes and views" on page 35.

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

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

2 Click the Close button on the title bar.

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

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

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

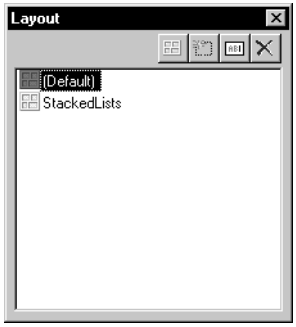
1 If the title bar of the top view in the stack is not displayed, display it by placing the pointer on the splitter bar at the top of the pane.

2 Click the Close button on the title bar.

Saving a layout

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

Figure 2-3: Layout dialog box with a customized layout defined



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

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

2 Click the New Layout button (second from the left at the top of the dialog box).

3 Type an appropriate name in the text box and click OK.

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

Using toolbars

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

You can also create toolbars for use with application menus that you deploy. For information on creating toolbars for applications, see Chapter 15, “Working with Native Objects and Controls for Windows CE Devices.”

Toolbar basics

PocketBuilder uses three basic toolbar types: the PowerBar, PainterBar, and StyleBar.

Table 2-3: Toolbars in the IDE and when they are displayed by default

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

Drop-down toolbars

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

For example, the down arrow next to the Text button in the DataWindow painter displays the Controls drop-down toolbar, which has a button for each control you can place on a DataWindow object.

Figure 2-4: Buttons in the Controls drop-down toolbar

Default button replaced
The button you select from a drop-down toolbar replaces the default button on the main toolbar. For example, if you select the Picture button from the Controls drop-down toolbar, it replaces the Text button in the PainterBar.

Controlling the display of toolbars

You can control:

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

Choosing to display text and PowerTips affects all toolbars.

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

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

2 Click the items you want.

A check mark means the item is currently selected.

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

1 Select Tools>Toolbars from the menu bar.

The Toolbars dialog box displays.

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

PocketBuilder saves your toolbar preferences in the registry.

Moving toolbars using the mouse

You can use the mouse to move a toolbar.

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

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

2 Press and hold the left mouse button.

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

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

Docking toolbars

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

- At the top or bottom of the workspace, at any point from the left edge to the right edge
- At the left or right of the workspace, at any point from the top edge to the bottom edge
- To the left or right of, or above or below, another toolbar

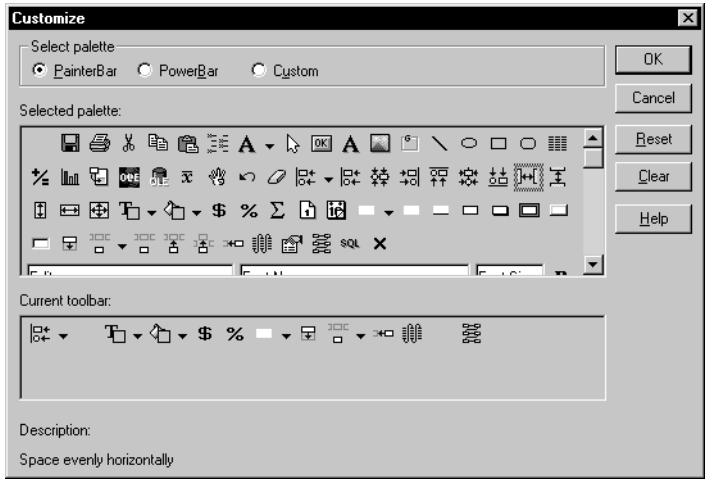
Customizing toolbars

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

Adding, moving, and deleting buttons

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

Figure 2-5: Customize dialog box for IDE toolbar buttons



❖ **To add a button to a toolbar:**

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

2 Select Customize.

The Customize dialog box displays.

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

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

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

For more information, see "Adding a custom button" on page 44.

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

❖ **To move a button on a toolbar:**

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

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

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

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

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

Resetting a toolbar

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

❖ **To reset a toolbar:**

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

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

Clearing or deleting a toolbar

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

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

For information about creating new toolbars and about the meaning of PowerBar1, see "Creating new toolbars" on page 46.

❖ **To clear or delete a toolbar:**

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

2 Click the Clear button, then Yes to confirm.

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

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

4 Click OK.

If you added new buttons, the toolbar is saved and contains the new buttons. If you did not add new buttons, the toolbar is deleted.

Adding a custom button

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

- Invoke a PocketBuilder menu item
- Run an executable (application) outside PocketBuilder
- Run a query or preview a DataWindow object
- Place a user object in a window or in a custom user object
- Assign a display format or create a computed field in a DataWindow object

❖ **To add a custom button:**

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

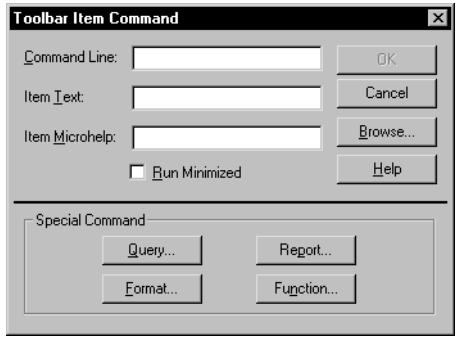
2 Select Custom in the Select Palette group box.

The custom buttons display in the Selected Palette list box.

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

The Toolbar Item Command dialog box displays. Different buttons display in the dialog box, depending on which toolbar you are customizing.

Figure 2-6: Toolbar Item Command dialog box



4 Fill in the Command Line box using Table 2-4 on page 45.

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

ButtonText, PowerTip

For example:

```
Save, Save File
```

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

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

Table 2-4: Defining custom buttons

Button action	Toolbar Item Command dialog box entry

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

Button action	Toolbar Item Command dialog box entry
Run an executable file outside PocketBuilder	Type the name of the executable file in the Command Line box. Specify the full path name if the executable is not in the current search path. To search for the file name, click the Browse button.
Run a query	Click the Query button and select the query from the displayed list.
Preview a DataWindow object	Click the Report button and select a DataWindow object from the displayed list. You can then modify the command-line arguments in the Command Line box.
Select a user object for placement in a window or custom user object	Window and User Object painters only. Click the UserObject button and select the user object from the displayed list.
Assign a display format to a column in a DataWindow object	DataWindow painter only. Click the Format button to display the Display Formats dialog box. Select a data type, then choose an existing display format from the list or define your own in the Format box. For more about specifying display formats, see Chapter 21, "Displaying and Validating Data."
Create a computed field in a DataWindow object	DataWindow painter only. Click the Function button to display the Function for Toolbar dialog box. Select the function from the list.

Modifying a custom button

❖ **To modify a custom button:**

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

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

The Toolbar Item Command dialog box displays.

3 Make your changes, as described in "Adding a custom button" on page 44.

Creating new toolbars

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

PowerBars and PainterBars

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

Table 2-5: PowerBars and PainterBars

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

About the StyleBar

A StyleBar is another type of toolbar that is available by default in certain painters. You can have only one StyleBar per painter. The default StyleBar contains drop-down lists for text fonts and sizes, as well as buttons for text presentation styles and alignment settings. For painters that do not have the default StyleBar, you can define a toolbar with the name StyleBar, but you can add only painter-specific buttons, not style buttons.

Where you create them

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

❖ **To create a new toolbar:**

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

   The New Toolbar dialog box displays a list of PowerBar and PainterBar names. StyleBar is also listed for painters that do not already have a StyleBar.

2. Select a toolbar name and click OK.

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

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

Customizing keyboard shortcuts

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

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

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

1 Select Tools>Keyboard Shortcuts from the menu bar.

The keyboard shortcuts for the current menu bar display.

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

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

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

Figure 2-7: Message indicating a shortcut key is already assigned

❖ **To remove a keyboard shortcut associated with a menu item:**

1 Select Tools>Keyboard Shortcuts from the menu bar.

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

3 Click Remove.

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

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

- Click the Reset button and respond to the prompt

Changing fonts

Table 2-6 summarizes the various ways you can change the fonts used in PocketBuilder.

Table 2-6: Changing the fonts used in PocketBuilder

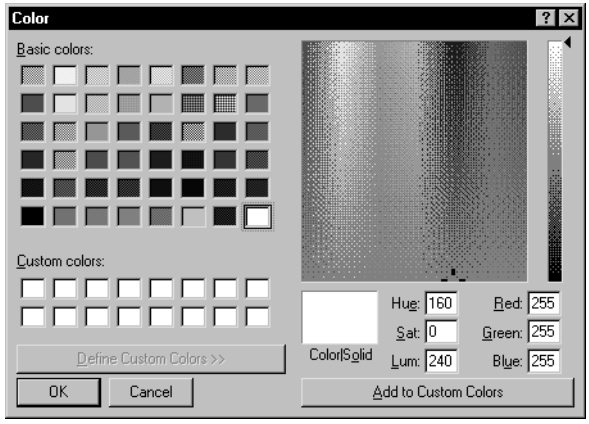
Object, painter, or tool	How to change fonts
A table's data, headings, and labels.	In the Database painter, display the Properties view for the table, and change the font properties on the Data, Heading, and Label Font tabs.
Objects in the User Object, Window, and DataWindow painters.	Select objects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

Changes you make in the Tools>System Options dialog box and from the Design>Options menu selection are used the next time you open PocketBuilder.

Defining colors

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

Figure 2-8: The Color dialog box for defining custom colors



❖ **To define custom colors:**

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

The Color dialog box displays.

2 Define your custom colors using the Color dialog box and click OK.

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

Managing the PocketBuilder IDE

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

About the registry

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

Information related to your preferences, such as the applications you have created, the way you have arranged your views in the painters, and the shortcut keys you have defined for PocketBuilder menu items, is stored in *HKEY_CURRENT_USER/Software/Sybase/PocketBuilder/2.0*.

Installation-related information is stored in *HKEY_LOCAL_MACHINE/Software/Sybase/PocketBuilder/2.0*.

About the initialization file

PK.INI is a text file containing variables that specify your PocketBuilder preferences. These preferences include information such as the last workspace you used and your startup preferences. When you perform certain actions in PocketBuilder, your preferences are written to *PK.INI* automatically.

Format of INI files

PK.INI uses the Windows INI file format. This file has three types of elements:

- Section names, which are enclosed in square brackets
- Keywords, which are the names of preference settings
- Values, which are numeric or text strings assigned as the value of the associated keyword

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

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

Specifying preferences

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

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

Where the initialization file is kept

PK.INI is installed in the same directory as the PocketBuilder executable file.

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

❖ **To record your initialization path:**

1 Select Tools>System Options from the menu bar.

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

PocketBuilder records the path in the Windows registry.

How PocketBuilder finds the initialization file

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

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

PART 2

Working with Targets and Libraries

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

CHAPTER 3

Working with PowerScript Targets

About this chapter

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

Contents

About PowerScript targets

PowerScript is a scripting language used by PowerBuilder and PocketBuilder. To create an executable PocketBuilder application, a PowerScript target is required.

The first step in creating a new application or component is to use a target wizard, described in "Creating a target" on page 15.

The Application object

Every PowerScript target includes an Application object, which is a discrete object that is saved in a PocketBuilder library (*PKL* file). When a user runs the application, the scripts you write for events are triggered in the Application object.

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

Working in painters

In PocketBuilder, you use particular painters to edit objects such as applications, windows, menus, DataWindow objects, and user objects. Other painters such as the Library painter and the Database painter provide you with the ability to work with libraries and databases.

PocketBuilder painters

Table 3-1 describes the painters available in PocketBuilder.

Table 3-1: Painters in PocketBuilder

Use this painter	To do this
Application painter	Specify application-level properties and scripts.
Database painter	Maintain databases, control user access to databases, manipulate data in databases, and create tables.
DataWindow painter	Build intelligent objects called DataWindow objects that present information from the database.
Function painter	Build global functions to perform processing specific to your application.
Library painter	Manage libraries, create a new library, and build dynamic libraries.
Menu painter	Build menus to be used in windows.
Project painter	Create an executable file for your application, select a build directory, specify a resource file, specify a CAB file for distribution, and select certificates for signing the application and CAB files.
Query painter	Graphically define and save SQL SELECT statements for reuse with DataWindow objects.
SQL Select painter	Graphically define SQL SELECT statements for DataWindow objects. You can open this painter from the Design>Data Source menu of the DataWindow painter.

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

Opening painters

Painters that edit objects

There are several ways to open painters that edit objects.

Table 3-2: How to open painters that edit objects

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

Other painters

Painters that do not edit objects are accessible on the Database tab or the Tool tab of the New dialog box. Some are also available on the PowerBar and from the Tools menu.

Select Target for Open

Suppose that you use the same PKL in more than one target in the current workspace. If you double-click an object from that PKL in the Library painter or on the Workspace page of the System Tree when the root is not set to the current workspace, the Select Target for Open dialog box displays. The same dialog box displays if you select Inherit, Run/Preview, Regenerate, Print, or Search for this object under the above conditions.

Painter features

Painters that edit objects

Table 3-3 describes the features included in most painters used to edit PocketBuilder objects.

Table 3-3: Common features of PocketBuilder painters

Feature	Notes
Painter window with views	See “Views in painters that edit objects” next.
Unlimited undo/redo	Undo and redo apply to all changes.
Drag-and-drop operations	Most drag-and-drop operations change context or copy objects.
To-Do List support	When you are working in a painter, a linked item you add to the To-Do list can take you to the specific location. See “Using the To-Do List” on page 23.
Save needed indicator	When you make a change, PocketBuilder displays an asterisk after the object’s name in the painter’s Title bar to remind you that the object needs to be saved.

Other painters

Most of the painters that are not used to edit PocketBuilder objects have views and some drag-and-drop operations.

Views in painters that edit objects

Each painter has a View menu that you use for opening views. Which views you can open depends on the painter you are working in. Every painter has a default arrangement of views. You can rearrange these views, choose to show or hide views, and save arrangements that suit your working style.

For more information about arranging views, see Chapter 2, “Customizing PocketBuilder.”

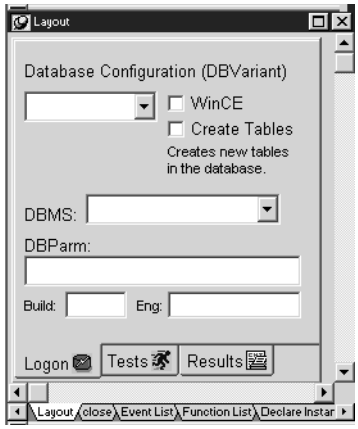
Many views are shared by some painters, but some views are specific to a single painter. For example, the Layout, Properties, and Control List views are used by the Window, Visual User Object, and Application painters, but the Design, Column Specifications, Data, and Preview views are specific to the DataWindow painter. The WYSIWYG Menu and Tree Menu views are specific to the Menu painter.

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

Layout view

The Layout view shows a representation of the object and its controls. It is where you place controls on an object and design the layout and appearance of the object. Figure 3-1 displays an example.

Figure 3-1: Example of a window in Layout view

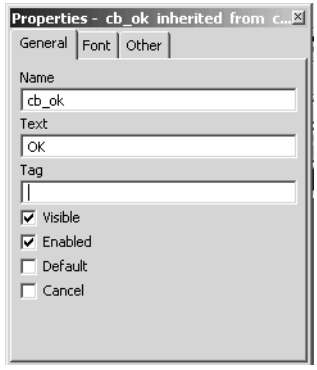


If the Properties view is displayed and you select a control in the Layout view or the Control List view, the properties for that control display in the Properties view. If you select several controls in the Layout view or the Control List view, the properties common to the selected controls display in the Properties view.

Properties view

The Properties view displays properties for the object itself or for the currently selected controls or nonvisual objects in the object. You can see and change the values of properties in this view. Figure 3-2 shows the Properties view for a button control.

Figure 3-2: Properties view for a button control



The Properties view dynamically changes when you change selected objects or controls in the Layout, Control List, and Non-Visual Object List views.

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

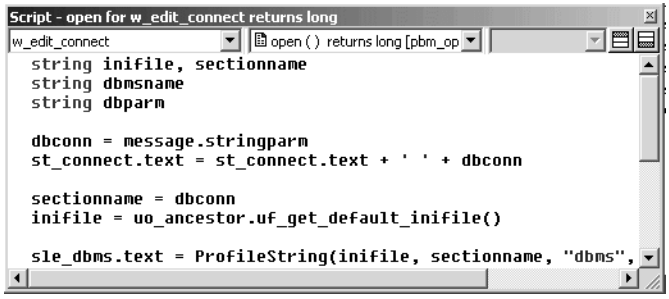
In the Properties view pop-up menu, you can select Labels On Top or Labels On Left to specify where the labels for the properties display. For help on properties, select Help from the pop-up menu.

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

Script view

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

Figure 3-3: Script view for a window Open event



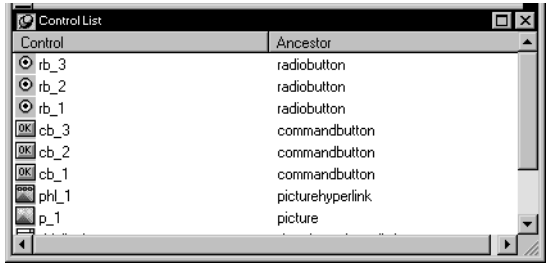
You can open the default script for an object or control by double-clicking it in the System Tree or the Layout, Control List, or Non-Visual Object List views, and you can insert the name of an object, control, property, or function in a script by dragging it from the System Tree.

For information about the Script view, see Chapter 6, "Writing Scripts."

Control List view

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

Figure 3-4: Control List view for a window

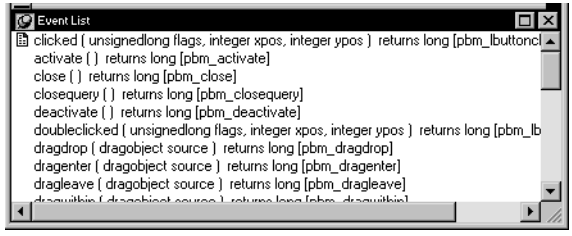


If you select one or more controls in the Control List view, the controls are also selected in the Layout view. Selecting a control changes the Properties view, and double-clicking a control changes the Script view.

Event List view

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

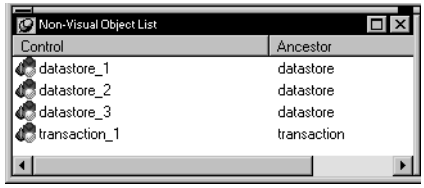
Figure 3-5: Event List view for a window



Non-Visual Object List view

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

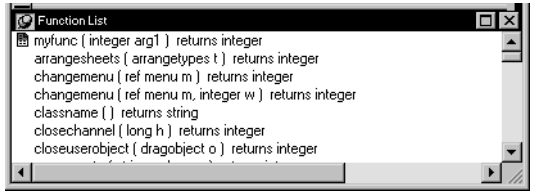
Figure 3-6: Non-Visual Object List view



Function List view

The Function List view lists the system-defined functions and the object-level functions you defined for the object. Icons identify whether a function has a script, is a descendant of a function with a script, or is a descendant of a function with an ancestor script as well as a script of its own.

Figure 3-7: Function List view

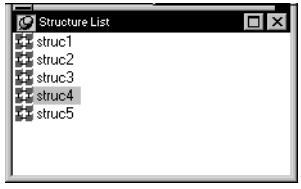


Note that although the half-colored icon identifies the myfunc user-defined function as having both an ancestor script and a script of its own, for a function this means that the function is overridden. This is different from the meaning of a half-colored icon in the Event List view.

Structure List view

The Structure List view lists the object-level structures defined for the object.

Figure 3-8: Structure List view

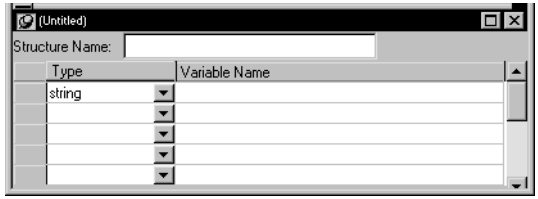


If you double-click a structure in the Structure List view, the structure's definition displays in the Structure view.

Structure view

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

Figure 3-9: Structure view



About the Application painter

Views in the Application painter

The Application painter has several views where you specify properties for your application and how it behaves at start-up. Because the Application painter is an environment for editing a nonvisual object of type application, the Application painter looks like the User Object painter for nonvisual user objects, and it has the same views. For details about the views, how you use them, and how they are related, see “Views in painters that edit objects” on page 58.

Most of your work in the Application painter is done in the Properties view and the Script view to set application-level properties and code application-level scripts. For information about specifying properties, see “Specifying application and Today item properties” on page 64. For information about coding in the Script view, see Chapter 6, “Writing Scripts.”

Inserting nonvisual objects

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

You insert a nonvisual object in an Application object in the same way you insert one in a user object. For more information, see “Using class user objects” on page 325.

Adding a custom Today item

The Properties view for an Application object includes a Today Item page for adding a custom item to the Today screen for a Pocket PC. The Today screen acts as the main screen or home page in the Pocket PC environment. Default Today items show the status of an application and launch it when you tap the item. For example, the default Calendar item on the Today screen shows whether you have any upcoming appointments. When you tap the item, the Calendar application starts (or displays if it is running in the background).

Specifying application and Today item properties

You specify Application object and Today item properties in the Application painter's Properties view. You can assign additional application properties in the Additional Properties dialog box that you open by clicking a button on the General tab page of the painter's Properties view.

Table 3-4: Specifying properties of an application object

To specify this	Use this
DWMessageTitle property	General tab page
Properties for a custom Today item	Today Item tab page
Default font for static text as it appears in windows, user objects, and DataWindow objects	Text Font tab page of the Additional Properties dialog box
Default font for data retrieved in a DataWindow object	Column Font tab page of the Additional Properties dialog box
Default font for column headers in tabular and grid DataWindow objects	Header Font tab page of the Additional Properties dialog box
Default font for column labels in freeform DataWindow objects	Label Font tab page of the Additional Properties dialog box
Application icon	Icon tab page of the Additional Properties dialog box
Global objects for the application	Variable Types tab page of the Additional Properties dialog box

❖ **To specify application properties:**

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

The only modifiable property on the General tab page is DWMessageTitle, which specifies the title of the message box for any runtime DataWindow errors encountered in the application. If you change the value of this property in script, the new value will be recognized only for DataWindows created (or painted) after the new value is set.

2 (Optional) Specify properties of a custom Today item on the Today Item tab page of the Properties view.

For information about Today item properties, see "Application object properties for a custom Today item" on page 66.

3 On the General tab page, click the Additional Properties button, and specify additional application properties in the Application properties dialog box.

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

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

- "Specifying default text properties" on page 69
- "Specifying an icon" on page 70
- "Specifying default global objects" on page 71

Application object properties for a custom Today item

Today item properties

PocketBuilder lets you add a custom item to the Today screen that can launch a PocketBuilder application. You set the properties for a custom Today item in the Application painter:

Table 3-5: Properties for a custom Today item in the Application painter

Property	Description
DisplayText	Text that displays when the custom item first displays. If you do not specify a display application, this text is always used.
DisplayApplication	An application that changes the appearance of the custom item dynamically. For example, the application might have a timer event that replaces the display text with a reminder of when to run the application the item launches.
Name	The name used to identify the custom item in the Windows CE registry.
RunApplication	The name of the application to be launched when the user taps the custom item. This can be the same application you specified as a display application or a different application. If you specify separate display and run applications, set the Today item properties on the display application. If you do not specify a run application, nothing will happen when the user taps the custom item.
Order	The position in the list of Today items where the custom item displays.
BackColor	The background color of the custom item. By default, this is set to display the standard background used by the Today screen.
TextColor	The color of the text for custom item label. By default, this is set to display the standard text color used by the Today screen.

Custom item icon

The icon used in the Today screen is the same as the icon specified for the run application. You specify this icon in the Additional Properties dialog box launched from the General property page in the Application painter. The icon from the display application is used if you do not set a run application. No icon is used if you do not set a run or display application.

Changing the display text

If you specify a display application, you can use it to change the text on the Today screen. For example, suppose you have a PocketBuilder application that initiates MobiLink synchronization. You could write a display application called SyncDisplay that counts the number of updates that have been performed on the local database since the last time it was synchronized. This statement changes the text in the custom item to show the latest count:

```
SyncDisplay.TodayDisplayText="Sync Update Count is " &
   + string(counter)
```

The user could monitor the count and use it to decide when to tap the custom item to launch the synchronization application.

Changing the displayed text is a useful feature. However, when planning to add a custom Today item, you should consider its memory requirements.

Memory usage

If you set both a display application and a run application, the PocketBuilder VM (PKVM) is loaded twice, in two separate processes, even if you use the same application as the display and run application. One is loaded with the Today screen and remains in memory at all times, or until the custom Today item is disabled or removed. The second PKVM is loaded when the custom item is tapped and remains in memory until the launched application terminates.

If you are concerned about memory requirements, specify a run application only. The PKVM is not loaded until the custom item is tapped. It remains in memory only while the application is running. (If you set neither a display nor a run application, the PKVM is never loaded.)

Deploying a custom item

To deploy a Today item with an application, you must select the Deploy Today Item check box in the Project painter.

It is possible to set up different configurations for the Today item that you deploy to a device or emulator. The configuration you choose depends on the amount of memory you want to consume and the functionality that you want to make available to the end user.

Table 3-6: Configurations for a PocketBuilder Today item

Configuration	Effects on memory consumption
Valid values provided for Display Application and Run Application properties	PocketBuilder VM is loaded at all times (remains in memory)
Valid value provided for Run Application property; no Display Application is set	PocketBuilder VM is loaded only after a user taps the custom Today item. It resides in memory until the application is shut down.
No values provided for Display Application and Run Application properties	PocketBuilder VM is not loaded, even after a user taps the custom Today item.

When you deploy an application with a Today item to a Pocket PC device, a new key with the name you specified on the Today Item property page is added to the *HKEY_LOCAL_MACHINE\Software\Microsoft\Today\Items* registry key. The other properties are added to the new key as string or DWORD values. These registry settings are also added to the CAB file if you choose to build one.

Using the TodaySave function

If you use a display application to change the appearance or other characteristics of the custom item, use the Application object's TodaySave function to update the current state of the custom item in the registry as well as on the Today screen. This ensures that the changes display if the device is reset or rebooted.

Disabling or removing custom items

A user can enable or disable a custom item by selecting Settings>Today from the Start menu, and selecting or clearing the check box on the Items page. If the custom item is disabled, it does not display on the Today screen.

You can remove all custom PocketBuilder Today items by selecting Design>Delete Custom Today Items>Default Device from the Project painter menu. Making this selection does not immediately delete an active Today item, but it does make sure the item information is removed from the registry. The next time the device is reset or rebooted, the previously active custom item is also removed.

For more information

The properties associated with the Today item are described in the *PowerScript Reference* and in the online Help.

Specifying default text properties

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

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

Properties set in the Database painter override application properties
If extended attributes have been set for a database column in the Database painter or Table painter, those font specifications override the fonts specified in the Application painter.

❖ **To change the text defaults for an application:**

1 In the Properties view, click Additional Properties and select one of the following:

Text Font tab
Header Font tab
Column Font tab
Label Font tab

The tab you choose displays the current settings for the font, size, style, and color. The text in the Sample box illustrates text with the current settings.

2 Review the settings and make any necessary changes:

- To change the font, select a font from the list in the Font list.
- To change the size, select a size from the list in the Size list or type a valid size in the list.
- To change the style, select a style (Regular, Italic, Bold, or Bold Italic) from the Font styles list.
- To change font effects, select one or more from the Effects group box (Strikeout and Underline).

- To change the text color, select a color from the Text Color list. (You do not specify colors for data, headings, and labels here. You do that in the DataWindow painter.)
- To change the background color, select a color from the Background list.

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

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

Specifying an icon

An application icon can display in the File Explorer next to the application file name, and in the Pocket PC Switcher bar and Start menu.

The Switcher bar drop-down list on a Pocket PC displays the icons of active applications that are running in the background. If you specify an icon for an Application object, the icon you select displays in the Switcher bar drop-down list and in the row of icons under the Start menu that represent the most recently used applications.

The application icon must have a resolution appropriate to the device where you deploy it. Different platforms might require 16x16 pixel or 32x32 pixel icons with either a 4- or 8-bit color depth (16 or 256 colors). Some VGA screens require 48x48 pixel icons with 256 colors. To ensure that the icon you want is used for the deployed application, the icon file you select should include an image in each of the supported resolutions and color depths.

If you specify an icon in the PocketBuilder IDE rather than in script, the icon is automatically included in the application executable when you build and deploy the application project. In this case you do not need to list the icon in a PocketBuilder Resource (PKR) file.

Pocket PC devices can cache a copy of the application icon when you run a program. Therefore, if you are redeploying an application with a changed or modified application icon, a soft reset might be required on the Pocket PC device before the new icon is displayed.

❖ **To associate an icon with an application:**

1 In the Properties view, click Additional Properties and select the Icon tab.

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

The button displays below the Browse button.

3 Click OK to associate the icon with the application.

Specifying default global objects

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

Table 3-7: Built-in global objects in PocketBuilder applications

Global object	Description
SQLCA	Transaction object, used to communicate with your database
Error	Used to report errors during execution
Message	Used to process messages that are not defined events, and to pass parameters between windows

You can create your own versions of these objects by creating a standard class user object inherited from one of the built-in global objects. You can add instance variables and functions to enhance the behavior of the global objects.

For more information, see Chapter 14, “Working with User Objects.”

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

❖ **To specify the default global objects:**

1 In the Properties view, click Additional Properties and select the Variable Types tab.

The Variable Types property page displays.

2 Specify the standard class user object you defined in the corresponding field.

For example, if you defined a user object named mytrans that is inherited from the built-in Transaction object, type `mytrans` in the box corresponding to SQLCA.

3 Click OK.

When you run your application, it will use the specified standard class user objects instead of the built-in global objects as the default objects.

Writing application-level scripts

When a user runs an application, an Open event is triggered in the Application object. The script you write for the Open event initiates the activity in the application. Typically it sets up the environment and opens the initial window.

When a user ends an application, a Close event is triggered in the Application object. The script you write for the Close event usually does all the cleanup required, such as closing a database or writing a preferences file.

If there are serious errors during execution, a SystemError event is triggered in the Application object.

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

Table 3-8 lists all events that can occur in the Application object. The only event that requires a script is Open.

Table 3-8: Events in the Application object

Event	Occurs when
Open	The user starts the application.
Close	The user closes the application. Typically, you write a script for this event that shuts everything down (such as closing the database connection and writing out a preferences file).
SystemError	A serious error occurs during execution (such as trying to open a nonexistent window). If there is no script for this event, PocketBuilder displays a message box with the PocketBuilder error number and message text. If there is a script, PocketBuilder executes the script. For more about error handling, see “Handling errors during execution” on page 670.
Idle	The Idle PowerScript function has been called and the specified number of seconds has elapsed with no mouse or keyboard activity.

Setting application properties in scripts

The Application object has several properties that specify application-level properties.

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

AppName.property

If the script is in the Application object itself, you do not need to qualify the property name with the application name.

Application name cannot be changed
The name of an application is one of the Application object's properties, but you cannot change it.

For a complete list of Application object properties, see *Objects and Controls* in the online Help. However, most of the Application object properties listed do not have any meaning for PocketBuilder.

Specifying the target's library search path

The objects you create in painters are stored in PocketBuilder libraries. You can use objects from one or more libraries in a PowerScript target. You define each library the target uses in the library search path.

PocketBuilder uses the search path to find referenced objects during execution. When a new object is referenced, PocketBuilder looks through the libraries in the order in which they are specified in the library search path until it finds the object.

On the Library List tab page of the Target Properties dialog box, you can modify the libraries associated with the current target.

❖ **To specify the target's library search path:**

1 In the System Tree, right-click on the target containing your application and select Properties from the pop-up menu.

The Target Properties dialog box displays.

2 Select the Library List tab page.

The libraries currently included in the library search path are displayed in the list.

3 Do one of the following:

- Enter the name of all the libraries you want to include in the Library Search Path list, separating them with semicolons
- Use the Browse button to include other libraries in your search path

You must specify libraries using an absolute path. To change the order of libraries in the search path, use the pop-up menu to copy, cut, and paste libraries.

To delete a library from the search path, select the library in the list and use the pop-up menu or press Delete.

Make sure the order is correct
When you select multiple libraries from the Select Library dialog box using Shift+Click or Ctrl+Click, the first library you select appears last in the Library Search Path list and will be the last library searched.

4 Click OK.

PocketBuilder updates the search path for the target.

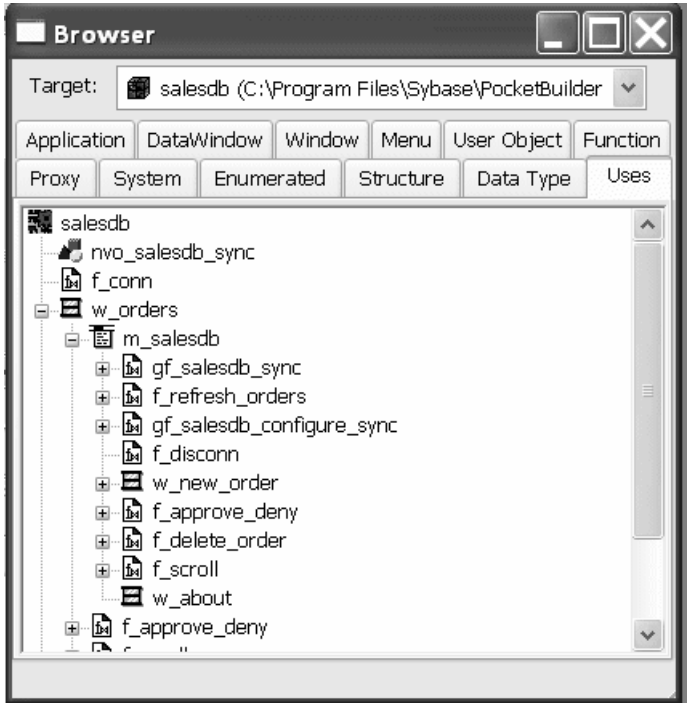
Where PocketBuilder maintains the library search path
PocketBuilder stores your target's library search path in the target (PKT) file in a line beginning with `liblist`; for example:

```
liblist "pbtutor.pkl;tutor_pb.pkl";
```

Looking at an application's structure

If you are working with an application that references one or more objects in an application-level script, you can look at the application's structure in the Browser.

Figure 3-10: Displaying an application's structure in the Browser



❖ **To display the application's structure:**

1 Click the Browser button on the PowerBar.

2 In the Browser, select the Uses tab page and select Expand All from the Application object's pop-up menu.

PocketBuilder expands the display to show all the global objects that are referenced in a script for the Application object. You can expand the display more as needed.

Which objects are displayed

The Uses tab page of the Browser shows global objects that are referenced in your application. It shows the same types of objects that you can see in the Library painter. It does not show entities that are defined within other objects, such as controls and object-level functions.

Which references are displayed

The Browser displays the following types of references when the Application object is expanded.

Objects referenced in painters

These are examples of objects referenced in painters:

- If a menu is associated with a window in the Window painter, the menu displays when the window is expanded
- If a DataWindow object is associated with a DataWindow control in the Window painter, the DataWindow object displays when the window is expanded
- If a window contains a custom user object that includes another user object, the custom user object displays when the window is expanded, and the other user object displays when the custom user object is expanded

Objects directly referenced in scripts

These are examples of objects referenced in scripts:

- If a window script contains the following statement, w_continue displays when the window is expanded:

```
Open(w_continue)
```

Which referenced windows display in the Browser
Windows are considered referenced only when they are opened from within a script. A use of another window's property or instance variable does not cause the Browser to display the other window as a reference of the window containing the script.

- If a menu item script refers to the global function f_calc, f_calc displays when the menu is expanded:

```
f_calc(EnteredValue)
```

- If a window uses a pop-up menu through the following statements, m_new displays when the window is expanded:

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

Which references are not displayed

The Browser does not display objects referenced through instance variables or properties, or objects referenced dynamically through string variables.

Objects referenced through instance variables or properties

These are examples of objects referenced through instance variables or properties:

If w_go has this statement (and no other statement referencing w_emp), w_emp does not display as a reference for w_go:

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

Objects referenced dynamically through string variables

These are examples of objects referenced dynamically through string variables:

- If a window script has the following statements, the window w_go does not display when the window is expanded. The window w_go is named only in a string:

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

- If the DataWindow object d_emp is associated with a DataWindow control dynamically through the following statement, d_emp does not display when the window containing the DataWindow control is expanded:

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

Working with objects

In PowerScript targets, you can:

- Create new objects
- Create new objects using inheritance
- Open existing objects
- Run or preview objects

After you create or open an object, the object displays in its painter and you work on it there.

Creating new objects

To create new objects, you use the New dialog box.

❖ **To create a new object:**

1 Do one of the following:

- Click the New button in the PowerBar
- Select File>New from the menu bar
- In the System Tree, right-click on a workspace or target name and select New from the pop-up menu

2 In the New dialog box, select the appropriate tab page for the object you want to create.

You use icons on the PB Object tab page for creating new user objects, windows, menus, structures, and functions.

3 Select an icon and click OK.

Creating new objects using inheritance

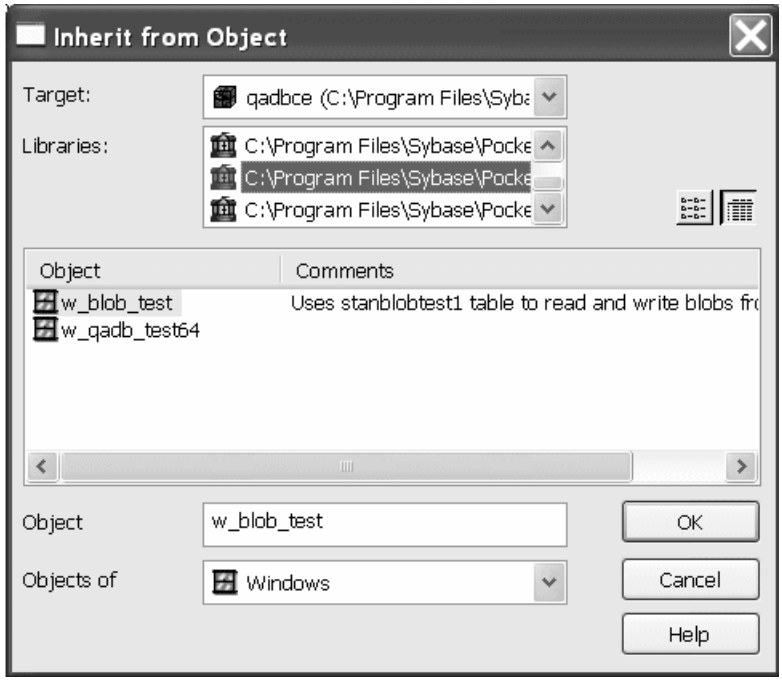
One of the most powerful features of PocketBuilder is inheritance. With inheritance, you can create a new window, user object, or menu (a descendent object) from an existing object (the ancestor object).

❖ **To create a new object by inheriting it from an existing object:**

1 Click the Inherit button in the PowerBar, or Select File>Inherit from the menu bar.

2 In the Inherit From Object dialog box, select the object type (menu, user object, or window) from the Object Type drop-down list. Then select the target as well as the library or libraries you want to look in. Finally, select the object from which you want to inherit the new object.

Figure 3-11: Inheriting from a window object



Displaying objects from many libraries
To find an object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to select additional libraries and Shift+Click to select a range.

3 Click OK.

The new object, which is a descendant of the object you chose to inherit from, opens in the appropriate painter.

For more information about inheritance, see Chapter 12, “Understanding Inheritance.”

Naming conventions

As you use PocketBuilder to develop your application, you create many different components for which you need to provide names. These components include objects such as windows and menus, controls that go into your windows, and variables for your event and function scripts.

You should devise a set of naming conventions and follow them throughout your project. This is critical when you are working in a team to enforce consistency and enable others to understand your code. This section provides tables of common naming conventions. PocketBuilder does not require you to use these conventions, but they are followed in many PocketBuilder books and examples.

All identifiers in PocketBuilder can be up to 40 characters long. You typically use the first few characters to specify a prefix that identifies the kind of object or variable, followed by an underscore character, followed by a string of characters that uniquely describes this particular object or variable.

Object naming conventions

Table 3-9 shows common prefixes for objects that you create in PocketBuilder.

Table 3-9: Common prefixes for objects

Prefix	Description
w_	Window
m_	Menu
d_	DataWindow
q_	Query
n_ or n_*standardobject*_	Standard class user object, where *standardobject* represents the type of object; for example, n_trans
n_ or n_cst	Custom class user object
u_ or u_*standardobject*_	Standard visual user object, where *standardobject* represents the type of object; for example, u_cb
u_	Custom visual user object
f_	Global function
of_	Object-level function
s_	Global structure
str_	Object-level structure
ue_	User event

Variable naming conventions

The prefix for variables typically combines a letter that represents the scope of the variable and a letter or letters that represent its datatype. Table 3-10 lists the prefixes used to indicate a variable's scope. Table 3-11 lists the prefixes for standard datatypes, such as integer or string.

The variable might also be a PocketBuilder object or control.
Table 3-12 lists prefixes for some common PocketBuilder system objects. For controls, you can use the standard prefix that PocketBuilder uses when you add a control to a window or visual user object. To see these prefixes, open the Window painter, select Design>Options, and look at the Prefixes 1 and Prefixes 2 pages.

Table 3-10: Prefixes that indicate the scope of variables

Prefix	Description
a	Argument to an event or function
g	Global variable
i	Instance variable
l	Local variable
s	Shared variable

Table 3-11: Prefixes for standard datatypes

Prefix	Description
a	Any
blb	Blob
b	Boolean
ch	Character
d	Date
dtm	DateTime
dc	Decimal
dbl	Double
e	Enumerated
i	Integer
l	Long
r	Real
s	String
tm	Time
ui	UnsignedInteger
ul	UnsignedLong

Table 3-12: Prefixes for selected PocketBuilder system objects

Prefix	Description
ds	DataStore
dw	DataWindow
dwc	DataWindowChild
dwo	DWobject

Prefix	Description
env	Environment
err	Error
gr	Graph
inet	Inet
ir	InternetResult
lvi	ListViewItem
mfd	MailFileDescription
mm	MailMessage
mr	MailRecipient
ms	MailSession
msg	Message
nvo	NonVisualObject
tr	Transaction
tvi	TreeViewItem

Opening existing objects

You can open existing objects from the Open dialog box or directly from the System Tree.

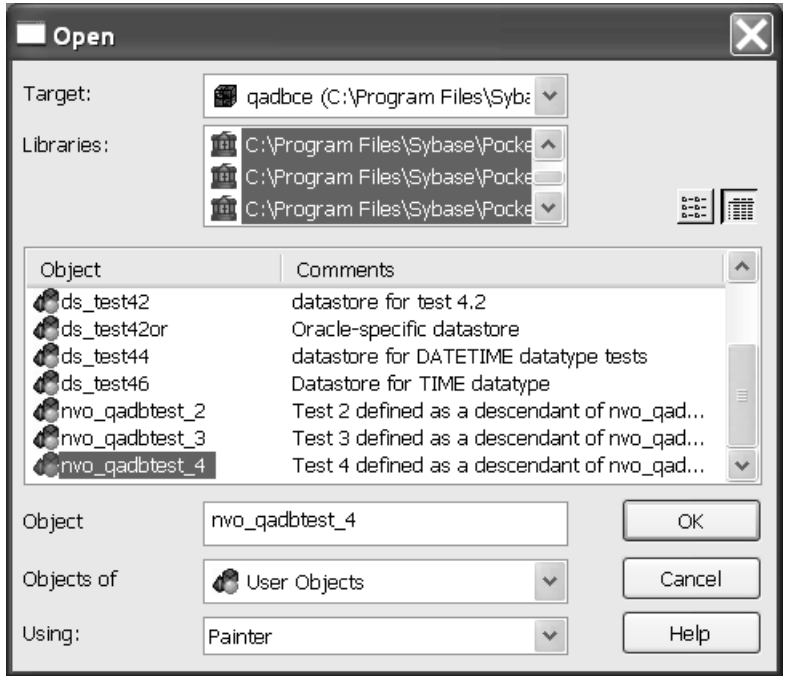
❖ **To open existing objects:**

1 Click the Open button in the PowerBar or Select File>Open from the menu bar.

When using the System Tree
You can open an existing object directly from the System Tree. Either double-click on the object name or select Edit from the pop-up menu.

2 In the Open dialog box, select the object type from the Object Type drop-down list. Then select the target as well as the library or libraries you want to look in. Finally, select the object you want to open.

Figure 3-12: Opening a User Object from the Open dialog box



Displaying objects from many libraries
To find an object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to select additional libraries and Shift+Click to select a range.

3 Click OK.

The object opens in the appropriate painter.

Accessing recently opened objects

You can quickly open recently opened objects by selecting File>Recent Objects from the menu bar. The Recent Objects list includes the eight most recently opened objects, but you can include up to 36 objects on the list.

❖ **To modify the number of recent objects:**

1 Select Tools>System Options from the menu bar.

2 On the General page of the System Options dialog box, modify the number for the recent objects list.

Running or previewing objects

To run a window or preview a DataWindow object, you can use a button on the PowerBar or the Run/Preview menu item in the File menu.

Using the System Tree
You can right-click an object in the System Tree and select Run/Preview from the pop-up menu.

❖ **To run or preview an object:**

1 Do one of the following:

- Click the Run/Preview Object button in the PowerBar
- Select File>Run/Preview from the menu bar

2 In the Run dialog box, select the object type from the Object Type drop-down list.

3 Select the target as well as the library or libraries you want to look in.

4 Select the object you want to run or preview and click OK.

The object runs or is previewed.

For more specific information on running a window, see “Running a window” on page 221. For information on using the DataWindow painter’s Preview view, see Chapter 17, “Defining DataWindow Objects.”

Using the Source editor

You can use the Source editor to edit the source of most PowerScript objects directly instead of making changes to an object in a painter. You cannot edit the source of project or proxy objects. The Source editor makes it unnecessary to export an object in order to edit it and then import it, as you do with the File editor.

For more information on the PocketBuilder File editor, see “Using the File editor” on page 25.

Caution—back up your objects
Although the Source editor provides a quick way to make global changes, you should use it with caution, and you must be familiar with the syntax and semantics of PowerScript source code before changing it in the Source editor.

Changes you make to an object's source code using the Source editor take effect *immediately* when you save the object, *before the code is validated.* If an error message displays in the Output window, you must fix the problem in the Source editor before you close the editor. If you do not, you will not be able to open the object in a painter.

Technical Support is not able to provide support if changes you make in the Source editor render an object unusable. For this reason, Sybase strongly recommends that you make backup copies of your PKLs or objects before you edit objects in the Source editor.

You can open an object in the Source editor in one of several ways:

- Use the Open dialog box
- Select the Edit Source menu item in the System Tree or Library painter
- Select the Edit Source menu item in the Output window for a line that contains an error

Unlike the File editor, the Source editor cannot be opened independently. It can be used only in conjunction with an object defined within a PowerScript target in the current workspace. You cannot open an object in the Source editor that is already open in a painter.

In exported objects that you view with the File editor, a PBExportHeader line is always generated for the object. If you saved the object with a comment from the object's painter, a PBExportComment line is also viewable in the exported file. The Source editor display is identical to the display in the File editor except that the PBExport lines that are visible at the beginning of the code in the File editor are not visible in the Source editor.

For more information on exporting objects, see "Exporting and importing entries" on page 106.

CHAPTER 4

Working with Libraries

About this chapter

PocketBuilder stores all the PowerScript objects you create in libraries. When you work with a PowerScript target, you specify which libraries it will use. This chapter describes how to work with your libraries.

Contents

About libraries

Whenever you save an object such as a window or menu in a painter, PocketBuilder stores the object in a library (a *PKL* file). Similarly, whenever you open an object in a painter, PocketBuilder retrieves the object from the library.

Assigning libraries

PowerScript targets can use as many libraries as you want. When you create a target, you specify which libraries it uses. You can also change the library search path for a target at any time during development.

For information about specifying the library search path, see “Specifying the target’s library search path” on page 73.

How the information is saved

Every object is saved in two parts in a library:

- **Source form** This is a syntactic representation of the object, including the script code.
- **Object form** This is a binary representation of the object, similar to an object file in the C and C++ languages. PocketBuilder compiles an object automatically every time you save it.

Using libraries

It is hard to predict the needs of a particular application, so the organization of a target's libraries generally evolves over the development cycle. PocketBuilder lets you reorganize your libraries easily at any time.

For small applications, you might use only one library, but for larger applications, you should split the application into different libraries.

About library size

There are no limits to how large libraries can be, but for performance and convenience, you should follow these guidelines:

- **Number of objects** It is a good idea not to save more than 50 or 60 objects in a library. This is strictly for your convenience; the number of objects does not affect performance. If you have many objects in a library, list boxes that list library objects become unmanageable and the System Tree and Library painter become more difficult to use.
- **Balance** Managing a large number of libraries with only a few objects makes the library search path too long and can slow performance by forcing PocketBuilder to look through many libraries to find an object. Try to maintain a balance between the size and number of libraries.

Organizing libraries

You can organize your libraries any way you want. For example, you might put all objects of one type in their own library, or divide your target into subsystems and place each subsystem in its own library.

Opening the Library painter

❖ **To open the Library painter:**

- Click the Library button in the PowerBar or select Tools>Library Painter

What you can do in the Library painter

In the Library painter, you can:

- Create a new library
- Create new objects in targets in your current workspace
- Copy, move, and delete objects in any library
- Open objects in libraries that are on a library list in the current Workspace to edit them in the appropriate painters
- Migrate, rebuild, and regenerate libraries in the current Workspace
- Create a dynamic runtime library (*PKD*) that includes objects in the current library and related resource objects

What you cannot do in the Library painter

You cannot migrate or open objects in PocketBuilder libraries that are not on the library list. You also cannot rename a library.

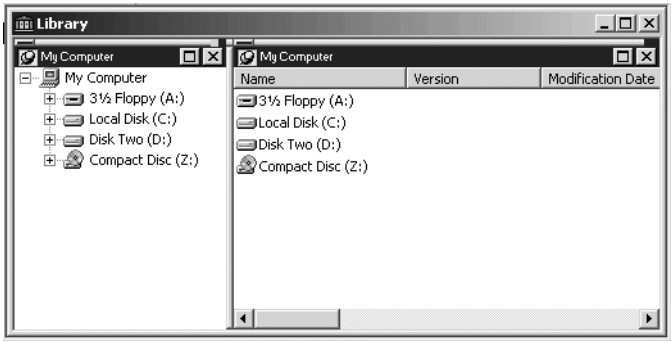
About the Library painter

Views in the Library painter

The Library painter has two views, the Tree view and the List view, that can display all the files in your file system, not just PocketBuilder objects. You use the painter primarily for displaying and working with workspaces, targets, library files, and the objects they contain.

The Tree and List views are available from the View menu. By default, the Library painter displays one Tree view (on the left) and one List view (on the right). When the Library painter opens, both the Tree view and the List view display all the drives on your computer, including mapped network drives.

Figure 4-1: Default view layout for Library painter



About the Tree view

The Tree view in the Library painter displays the drives and folders on the computer and the workspaces, targets, libraries, objects, and files they contain. You can expand drives, folders, and libraries to display their contents.

About the List view

The List view in the Library painter displays the contents of a selected drive, folder, or library and has columns with headers that provide extra information. For libraries, the comment column displays any comment associated with the library. For objects in libraries, the columns display the object name, modification date, size, and any comment associated with the object. You can resize columns by moving the splitter bar between columns. You can sort a column's contents by clicking the column header.

About sorting the Name column

When you click the Name column header repeatedly to sort, the sort happens in four ways: by object type and then name, in both ascending and descending order, and by object name only, in both ascending and descending order. You might not easily observe the four ways of sorting if all objects of the same type have names that begin with the same character or set of characters.

Displaying items in the Tree view and the List view

Typically, you select a library in the Tree view and display the objects in that library in the List view, but at any time, you can set a new root or move back and forward in the history of your actions to display libraries or other items. For more information, see "Setting the root" on page 99 and "Moving back, forward, and up one level" on page 100.

Using custom layouts

You might find that having more than one Tree view or List view makes your work easier. Using the View menu, you can display as many Tree views and List views as you need.

You can filter the objects in each of the List views so that one List view shows menus, another windows, and another user objects. For information about filtering objects in a view, see "Filtering the display of objects" on page 94.

For information about opening and closing views, manipulating views, returning to the default view layout, or saving your favorite layouts, see Chapter 2, "Customizing PocketBuilder."

View synchronization

Tree and List views are synchronized with each other. When you are using more than one Tree view or List view, changes you make in one type of view are reflected in the last view you touched of the other type. For example, when an item is selected in a Tree view, the contents of that item display in the List view that you last touched. When you display new contents in a List view by double-clicking an item, that item is selected in the Tree view you last touched unless this would require resetting the root.

You can also display the contents of different libraries in separate List views by dragging and dropping libraries from a Tree view to different List views. For information about dragging and dropping libraries, see "Displaying libraries and objects" on page 92.

Using the System Tree

The System Tree works like a Tree view in the Library painter. You can perform most tasks in either the System Tree or the Library painter Tree view, using the pop-up menu in the System Tree and the pop-up menu, PainterBar, or menu bar in the Library painter. When you have the System Tree and a Library painter open at the same time, the PainterBar and the menu bar apply to the Library painter only, not to the System Tree.

Each time you click the Library painter button on the PowerBar, PocketBuilder opens a new instance of the Library painter. By contrast, there is only a single instance of the System Tree, which you can display or hide by clicking the System Tree button on the PowerBar.

Working with libraries

The Library painter is designed for working with PocketBuilder libraries. You can use the System Tree instead of the Library painter to manage PocketBuilder libraries, but you can select only one object at a time in the System Tree, whereas the Library painter List view allows you to select multiple objects at the same time.

Displaying libraries and objects

What you see in the views

In the Library painter Tree view, you can expand items and see the folders, libraries, or objects they contain. The List view displays the contents of a selection in the Tree view.

❖ **To expand or collapse an item in the Tree view:**

- Double-click the item

  If the item contains libraries or objects, they display in the List view.

❖ **To display the contents of an item in the List view:**

- Select the item in the Tree view or double-click the item in the List view.

Using drag and drop to expand items

You can drag and drop items to expand them and see the contents.

If you drag an item from a Tree view or List view to a List view, the List view sets the item as the root and displays its contents.

If you drag an item from a Tree view or List view to a Tree view, the Tree view expands to display the dragged item.

For example, you can drag a library from the Tree view and drop it in the List view to quickly display the objects the library contains in the List view. If you are using one Tree view and multiple List views, you can drag a specific library from the Tree view to each List view so each List view contains the contents of a specific library.

For information about using drag and drop to copy or move items, see "Copying, moving, and deleting objects" on page 97.

Controlling columns that display in the List view

You can control whether to display the last modification date, compilation date, size, SCC version number, and comments (if comments were added when an object or library was created) in the List view.

The version number column in the Library painter list view remains blank if the source control system for your workspace does not support the PowerBuilder extension to the SCC API. If your source control system supports this extension and if you are connected to source control, you can override the SCC version number of a PowerScript object in the local copy directory through the property sheet for that object.

For more information about listing the SCC version number and overriding it through the PowerBuilder interface, see "Extension to the SCC API" on page 118.

❖ **To control the display of columns in the List view:**

1 Select Design>Options from the menu bar.

2 On the General tab page, select or clear these display items: Modification Date, Compilation Date, Sizes, SCC Version Number, and Comments.

Using the pop-up menu

Like other painters, the Library painter has a pop-up menu that provides options that apply to the selected item in the Tree view or the List view. For example, from a library's pop-up menu, you can delete, optimize, or search the library, print the directory, specify the objects that display in the library, and import objects into it.

The actions available from an object's pop-up menu depend on the object type. For PocketBuilder objects that you can work with in painters, there are pop-up menu items that allow you to edit the object in a painter or in the Source editor. You can copy, move, or delete the object, export it to a text file, search it, regenerate it, or send it to a printer. You can also preview and inherit from some objects. For most of these actions, the object must be in a library in your current workspace.

Actions available from the pop-up menus are also available on the Entry menu on the menu bar.

Selecting objects

In the List view, you can select one or more libraries or objects to act on.

❖ **To select multiple entries:**

- In the List view, use Ctrl+Click (for individual entries) and Shift+Click (for a group of entries)

❖ **To select all entries:**

- In the List view, select an object and click the Select All button on the PainterBar

Filtering the display of objects

By default, the Library painter displays all the objects in PocketBuilder libraries when you expand the libraries in a painter Tree view or List view. You can restrict what objects display in expanded libraries to specific kinds of objects or to objects whose names match a specific pattern. For example, you can limit the display to DataWindow objects only, or limit the display to windows with names that begin with `w_emp`.

Settings are remembered
PocketBuilder records your preferences in the Library section of the PocketBuilder initialization file so that the next time you open the Library painter, the same information is displayed.

❖ **To restrict which objects are displayed:**

1 Select Design>Options from the menu bar and select the Include tab.

2 Specify the display criteria:

- To limit the display to entries that contain specific text in their names, enter the text in the Name box. You can use the wildcard characters question mark (?) and asterisk (*) in the string. A ? represents one character; a * represents any string of characters. The default is all entries of the selected types.
- To limit the display to specific entry types, clear the check boxes for the entry types that you do not want to display. The default is all entries.

3 Click OK.

The Options dialog box closes.

4 In the Tree view, expand libraries or select a library to display the objects that meet the criteria.

Overriding the choices you made for a specific view
In either the Tree view or the List view, you can override your choice of objects that display in all libraries by selecting a library, displaying the library's pop-up menu, and then clearing or selecting items on the list of objects.

Creating and deleting libraries

A library is created automatically when you create a new PowerScript target, but you can create as many libraries as you need for your project in the Library painter.

❖ **To create a library:**

1 Click the Create button or select Entry>Library>Create from the menu bar.

The Create Library dialog box displays, showing the current directory and listing the libraries it contains.

2 Enter the name of the library you are creating and specify the directory in which you want to store it.

The file is given the extension *PKL*.

3 Click Save.

The library properties dialog box displays.

4 Enter any comments you want to associate with the library.

Adding comments to describe the purpose of a library is important if you are working on a large project with other developers.

5 Click OK.

PocketBuilder creates the library.

❖ **To delete a library:**

1 In either the Tree view or the List view, select the library you want to delete.

2 Select Entry>Delete from the menu bar or select Delete from the pop-up menu.

Restriction
You cannot delete a library that is in the current target's library search path.

The Delete Library dialog box displays, showing the library you selected.

3 Click Yes to delete the library.

The library and all its entries are deleted from the file system.

Creating and deleting libraries during execution
You can use the LibraryCreate and LibraryDelete functions in scripts to create and delete libraries. For information about these functions, see the online Help.

Filtering the display of libraries and folders

In either the Tree view or the List view, you can control what displays when you expand a drive or folder. An expanded drive or folder can display folders, workspaces, targets, files, and libraries.

❖ **To control display of the contents of drives and folders:**

- In either the Tree or List view, select a drive or folder, select Show from the pop-up menu, and select or clear items from the cascading menu.

Working in the current library

In PocketBuilder, the current library is the library that contains the object most recently opened or edited. That library becomes the default for Open and Inherit. If you click the Open or Inherit button in the PowerBar, the current library is the one selected in the Libraries list.

You can display the current library in the Library painter.

❖ **To display objects in the current library:**

1 Click in the Tree view or the List view.

2 Click the Display Most Recent Object button on the PainterBar or select Most Recent Object from the View menu.

The library that contains the object you opened or edited last displays in the view you selected with the object highlighted.

Opening and previewing objects

You can open and preview objects in the current workspace.

Opening PocketBuilder objects

PocketBuilder objects, such as windows and menus, can be opened only if they are in a PKL in the current workspace.

❖ **To open an object:**

- In either the Tree view or the List view, double-click the object, or select Edit from the object's pop-up menu

  PocketBuilder takes you to the painter for that object and opens the object. You can work on the object and save it as you work. When you close it, you return to the Library painter.

Opening other objects

The Library painter allows you to open most of the different file types it displays. When you double-click on an object, PocketBuilder attempts to open it using the following algorithm:

1 PocketBuilder determines if the object can be opened in the File editor. For example, files with the extensions *.txt*, *.ini*, and *.sr** open in the File editor.

2 PocketBuilder determines if the object can be opened in a painter.

3 PocketBuilder checks to see if the object is associated with a program in the *HKEY_CLASSES_ROOT* section of the Windows registry and, if so, launches the application.

Previewing PocketBuilder objects

You can run windows and preview DataWindow objects from the Library painter.

❖ **To preview an object in the Library painter:**

- Select Run/Preview from the object's pop-up menu

Copying, moving, and deleting objects

As the needs of your target change, you can rearrange the objects in libraries. You can copy and move objects between libraries or delete objects that you no longer need.

❖ **To copy objects using drag and drop:**

1 In the Tree view or the List view, select the objects you want to copy.

2 Drag the objects to a library in either view.

  If the contents of a library are displaying in the List view, you can drop the selected objects there.

  PocketBuilder copies the objects. If an object with the same name already exists, PocketBuilder prompts you and if you allow it, replaces it with the copied object.

❖ **To move objects using drag and drop:**

1 In the Tree view or the List view, select the objects you want to move.

2 Press and hold Shift and drag the objects to a library in either view.

If the contents of a library are displaying in the List view, you can drop the selected objects there.

PocketBuilder moves the objects and deletes them from the source library. If an object with the same name already exists, PocketBuilder prompts you and if you allow it, replaces it with the moved object.

❖ **To copy or move objects using a button or menu item:**

1 Select the objects you want to copy or move to another library.

2 Do one of the following:

- Click the Copy button or the Move button
- Select Copy or Move from the pop-up menu
- Select Entry>Library Item>Copy or Entry>Library Item>Move from the menu bar

The Select Library dialog box displays.

3 Select the library to which you want to copy or move the objects and click OK.

❖ **To delete objects:**

1 Select the objects you want to delete.

2 Do one of the following:

- Click the Delete button
- Select Delete from the pop-up menu
- Select Entry>Delete from the menu bar

You are asked to confirm the first deletion.

Being asked for confirmation
By default, PocketBuilder asks you to confirm each deletion. If you do not want to have to confirm deletions, select Design>Options to open the Options dialog box for the Library painter and clear the Confirm on Delete check box in the General tab page.

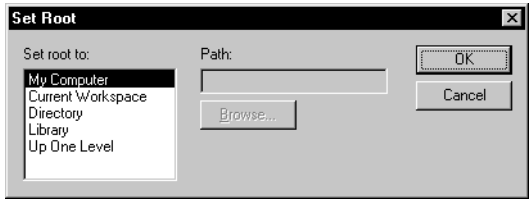
PocketBuilder records this preference as the DeletePrompt variable in the Library section of the PocketBuilder initialization file.

3 Click Yes to delete the entry or Yes To All to delete all entries. Click No to skip the current entry and go on to the next selected entry.

Setting the root

In either the Tree view or the List view, you can set the root location of the view from the Set Root dialog box.

Figure 4-2: The Set Root dialog box



Setting the root to the current workspace
In the System Tree, the default root is the current workspace. If you prefer to work in the Library painter, you might find it convenient to set the root to the current workspace. Using the current workspace as your root is particularly helpful if you are using many libraries in various locations, because they are all displayed in the same tree.

❖ **To set the root of the current view:**

1 In either view, select View>Set Root from the menu bar or select Set Root from the pop-up menu to display the Set Root dialog box.

2 Select the location you want from the Set Root To list box.

3 If you want the root to be a directory or library, type the path or browse to the path.

4 Click OK.

Moving back, forward, and up one level

You can set a new root location for the Library painter by moving back to where you were before, moving forward to where you just were, or, for the List view, moving up a level.

❖ **To move back, forward, or up one level:**

- Do one of the following:
  - Select View>Back, View>Forward, or View>Up One Level from the menu bar
  - Select Back, Forward, or Up One Level from the pop-up menu

  The name of the location you are moving back to or forward to is appended to Back and Forward.

Modifying comments

You can use comments to document your objects and libraries. For example, you might use comments to describe how a window is used, specify the differences between descendent objects, or identify a PocketBuilder library.

You can associate comments with an object or library when you first save it in a painter and add or modify comments in the System Tree or Library painter. If you want to modify comments for a set of objects, you can do so quickly in the List view.

❖ **To modify comments for multiple objects:**

1 In the List view, select the objects you want.

2 Select Entry>Properties from the menu bar or select Properties from the pop-up menu.

PocketBuilder displays the Properties dialog box. The information that displays is for one of the objects you selected. You can change existing comments, or, if there are no comments, you can enter new descriptive text.

3 Click OK when you have finished editing comments for this object.

If you do not want to change the comments for an object, click OK. The next object displays.

4 Enter comments and click OK for each object until you have finished.

If you want to stop working on comments before you finish with the objects you selected, click Cancel. The comments you have entered for other objects are retained. These comments can be seen in the List view.

❖ **To modify comments for a library:**

1 Select the library you want.

2 Click the Properties button or select Library from the pop-up menu.

3 Add or modify the comments, then click Apply or OK.

Searching targets, libraries, and objects

Global search of targets

You can search a target to locate where a specified text string is used. For example, you could search for:

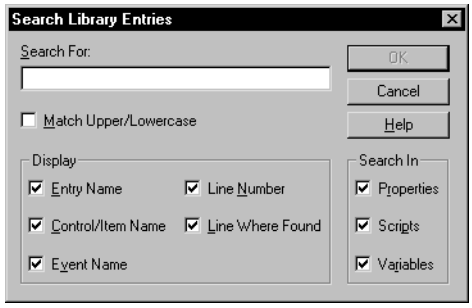
- All scripts that use the SetTransObject function
- All windows that contain the CommandButton cb_exit (all controls contained in a window are listed in the window definition, so they can be searched for as text)
- All DataWindow objects accessing the Employee table in the database

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

Searching selected libraries and objects

You can also select a library or one or more PocketBuilder objects to search using the Search Library Entries dialog box.

Figure 4-3: The Search Library Entries dialog box



The following procedure applies whatever the scope of your search is.

❖ **To search a target, a library, or objects for a text string:**

1 Select the target, library, or objects you want to search.

You can select multiple objects in the List view using Shift+Click and Ctrl+Click.

2 Select Search from the pop-up menu or the PainterBar.

The Search Library Entries dialog box displays.

3 Enter the string you want to locate (the search string) in the Search For box.

The string can be all or part of a word or phrase used in a property, script, or variable. You cannot use wildcards in the search string.

4 In the Display group box, select the information you want to display in the results of the search.

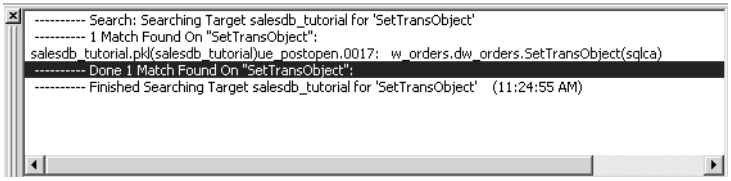
5 In the Search In group box, select the parts of the object that you want PocketBuilder to inspect: properties, scripts, and/or variables.

6 Click OK.

PocketBuilder searches the libraries for matching entries. When the search is complete, PocketBuilder displays the matching entries in the Output window.

For example, the following screen displays the results of a search for the string `SetTransObject`.

Figure 4-4: Results from a library search for SetTransObject



From the Output window, you can:

- Jump to the painter in which an entry was created

  To do this, double-click the entry or select it and then select Edit from the pop-up menu.
- Print the contents of the window
- Copy the search results to a text file

Optimizing libraries

You should optimize your libraries regularly. Optimizing removes gaps in libraries and defragments the storage of objects, thus improving performance.

Optimizing affects only layout on disk; it does not affect the contents of the objects. Objects are not recompiled when you optimize a library.

Once a week
For the best performance, you should optimize libraries you are actively working on about once a week.

❖ **To optimize a library:**

1 In either Tree view or List view, choose the library you want to optimize.

2 Select Entry>Library>Optimize from the menu bar or select Optimize from the library's pop-up menu.

PocketBuilder reorganizes the library structure to optimize object and data storage and index locations. Note that PocketBuilder does not change the modification date for the library entries. PocketBuilder saves the unoptimized version as a backup file in the same directory.

The optimized file is created with the default permissions for the drive where it is stored. On some systems new files are not shareable by default. If you see "save of object failed" messages or "link error" messages after optimizing, check the permissions assigned to the PKL.

If you do not want a backup file
If you do not want to save a backup copy of the library, clear the Save Optimized Backups check box in the Library painter's Design>Options dialog box. If you clear this option, the new setting will remain in effect until you change it.

Regenerating library entries

Why you need to regenerate objects

Occasionally you might need to update library entries. For example:

- When you modify an ancestor object, you can *regenerate* descendants so they pick up the revisions to their ancestor
- When you make extensive changes to a target, you can *rebuild* entire libraries so objects are regenerated sequentially based on interdependence
- When you upgrade to a new version of PocketBuilder, you need to *migrate* your targets

When you regenerate an entry, PocketBuilder recompiles the source form stored in the library and replaces the existing compiled form with the recompiled form.

❖ **To regenerate library entries:**

1 Select the entries you want to regenerate.

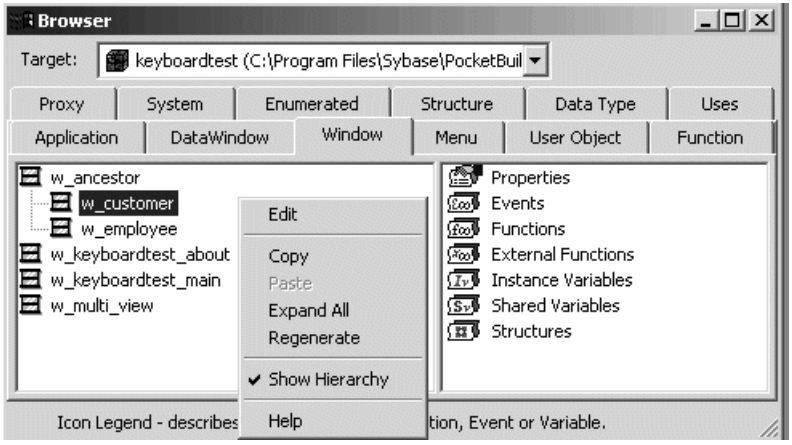
2 Click the Regenerate button or select Entry>Library Item>Regenerate from the menu bar.

PocketBuilder uses the source to regenerate the library entry and replaces the current compiled object with the regenerated object. The compilation date and size are updated.

Regenerating descendants

You can use the Browser to easily regenerate all descendants of a changed ancestor object.

Figure 4-5: Regenerating objects from the Browser



❖ **To regenerate descendants:**

1 Click the Browser button in the PowerBar.

The Browser displays.

2 Select the tab for the object type you want to regenerate.

For example, if you want to regenerate all descendants of window w_ancestor, click the Window tab.

3 Select the ancestor object and choose Show Hierarchy from its pop-up menu.

The Regenerate button displays on the pop-up menu.

4 Click the Regenerate button.

PocketBuilder regenerates all descendants of the selected ancestor.

For more about the Browser, see “Browsing the class hierarchy” on page 277.

Regenerate limitations

If you regenerate a group of objects, PocketBuilder regenerates them in the order in which they appear in the library, which may cause an error if an object is generated before its ancestor. For this reason, you should use a full or incremental build to update more than one object at a time.

Rebuilding workspaces and targets

When you make modifications to a target and need to update one or more libraries, you should use a rebuild option to update all the library objects in the correct sequence.

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

There are two methods to use when you rebuild a workspace or target:

- **Incremental rebuild** Updates all the objects and libraries that reference objects that have been changed since the last time you built the workspace or target
- **Full rebuild** Updates all the objects and libraries in your workspace or target

Table 4-1: Rebuilding a workspace or target

To rebuild	Do one of the following
Workspace	• Select Incremental Build Workspace or Full Build Workspace from the PowerBar • Select the Workspace in the System Tree or Library painter and select Incremental Build or Full Build from the pop-up menu
Target	• Select the target in the Library painter and select Entry>Target>Incremental Build or Entry>Target>Full Build from the menu bar • Select the target in the System Tree or Library painter and select Incremental Build or Full Build from the pop-up menu

Exporting and importing entries

Exporting object definitions to text files

You can export object definitions to Unicode text files. The text files contain all the information that defines the objects. The files are virtually identical syntactically to the source forms that are stored in libraries for all objects.

You might want to export object definitions in the following situations:

- You want to store the objects as text files
- You want to move objects to another computer as text files

Caution
The primary use of the Export feature is exporting source code, not modifying the source. You can use the Source editor to modify the source code of an object directly, but modifying source in a text file is not recommended for most users. See "Using the Source editor" on page 84.

❖ **To export entries to text files:**

1 Select the Library entries you want to export.

You can select multiple entries in the List view.

2 Do one of the following:

- Select Export from the pop-up menu
- Click the Export button on the PainterBar
- Select Entry>Library Item>Export from the menu bar

The Export Library Entry dialog box displays, showing the name of the first entry selected for export in the File Name box and the name of the current directory. The current directory is the target's directory or the last directory you selected for saving exported entries or saving a file using the File editor.

PocketBuilder appends the file extension *.srx*, where *x* represents the object type.

3 Specify the file name and directory for the export file. Do not change the file extension from the one that PocketBuilder appended.

4 Click OK.

PocketBuilder converts the entry to Unicode file format, stores it with the specified name, then displays the next entry you selected for export.

If a file already exists with the same name, PocketBuilder displays a message asking whether you want to replace the file. If you say no, you can change the name of the file and then export it, skip the file, or cancel the export of the current file and any selected files that have not been exported.

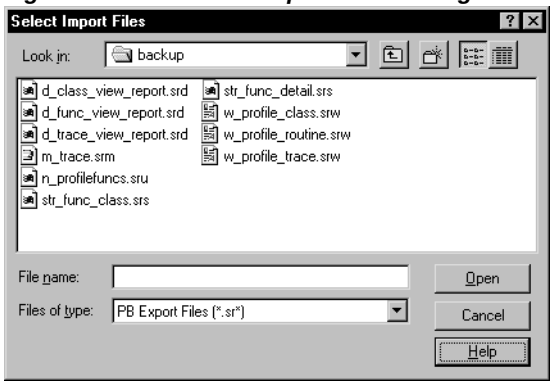
5 Repeat steps 3 and 4 until you have processed all the selected entries.

If the Library painter is set to display files, you can see the saved files and double-click them to open them in the File editor.

Importing text files

You can import source files for PocketBuilder objects to a library in the current workspace. The files you import can be in either ANSI or Unicode format. You select the files you want to import from the Select Import Files dialog box.

Figure 4-6: The Select Import Files dialog box



❖ **To import text files to library entries:**

1 In the System Tree or Library painter, select the library into which you want to import an object.

2 Select Import from the pop-up menu, or, in the Library painter only, click the Import button on the PainterBar.

The Select Import Files dialog box displays, showing the current directory and a list of files with the extension *.sr** in that directory. The current directory is the target's directory or the last directory you selected for saving exported entries or saving a file using the File editor.

3 Select the files you want to import. Use Shift+Click or Ctrl+Click to select multiple files.

4 Click Open.

PocketBuilder converts the specified text files to PocketBuilder format, regenerates (recompiles) the objects, stores the entries in the specified library, and updates the entries' timestamps.

If a library entry with the same name already exists, PocketBuilder replaces it with the imported entry.

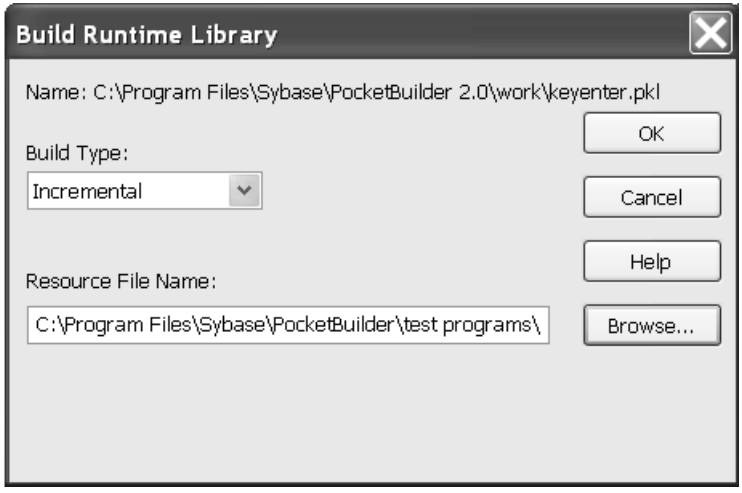
Caution
When you import an entry with the same name as an existing entry, the old entry is deleted before the import takes place. If an import fails, the old object will already have been deleted.

Creating runtime libraries

If you want your deployed target to use dynamic runtime libraries, you can create them from the Build Runtime Library dialog box.

For information about using runtime libraries, see Chapter 27, "Packaging and Distributing an Application." That chapter also describes the Project painter, which you can use to create dynamic runtime libraries automatically.

Figure 4-7: The Build Runtime Library dialog box



❖ **To create a runtime library:**

1 Select the library you want to use to build a runtime library.

2 Select Entry>Library>Build Runtime Library from the menu bar, or select Build Runtime Library from the library's pop-up menu.

The Build Runtime Library dialog box displays, listing the name of the selected library.

3 If any of the objects in the source library use resources, specify a PocketBuilder resource file in the Resource File Name box (see "Including additional resources" next).

4 Click OK.

PocketBuilder closes the dialog box and creates a runtime library with the same name as the selected library and the extension *pkd*.

Including additional resources

When building a runtime library, PocketBuilder does not inspect the objects; it simply removes the source form of the objects. Therefore, if any of the objects in the library use resources (pictures, icons, and pointers)—either specified in a painter or assigned dynamically in a script—and you do not want to provide these resources separately, you must list the resources in a PocketBuilder resource file (PKR file). Doing so enables PocketBuilder to include the resources in the runtime library when it builds it.

For more on resource files, see "Using PocketBuilder resource files" on page 714.

After you have defined the resource file, specify it in the Resource File Name box to include the named resources in the runtime library.

Creating reports on library contents

You can generate three types of reports from the Library painter:

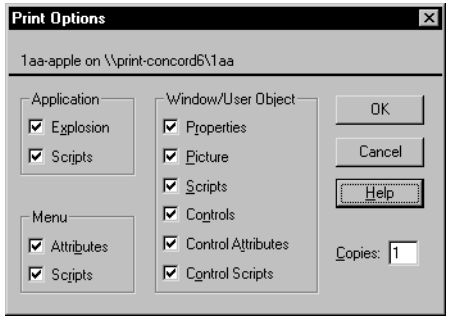
- The search results report
- Library entry reports
- The library directory report

The search results report contains the matching-entries information that PocketBuilder displays after it completes a search, described in "Searching targets, libraries, and objects" on page 101. The other two types of reports are described in this section.

Creating library entry reports

Library entry reports provide information about selected entries in the current target. You can use these reports to get printed documentation about the objects you have created in your target.

Figure 4-8: The Print Options dialog box for library entry reports



❖ **To create library entry reports:**

1 Select the library entries you want information about in the List view.

2 Select Entry>Library Item>Print from the menu bar, or select Print from the pop-up menu.

The Print Options dialog box displays.

3 If you have selected the Application object or one or more menus, windows, or user objects to report on, select the information you want printed for each of these object types.

For example, if you want all properties for selected windows to appear in the report, make sure the Properties box is checked in the Window/User Object group box.

The settings are saved
PocketBuilder records these settings in the Library section of the PocketBuilder initialization file.

4 Click OK.

PocketBuilder generates the selected reports and sends them to the printer specified in Printer Setup in the File menu.

Creating the library directory report

The library directory report lists all entries in a selected library in your workspace, showing the following information for all objects in the library, ordered by object type:

- Name of object
- Modification date and time
- Size (of compiled object)
- Comments

❖ **To create the library directory report:**

1 Select the library that you want the report for.

The library must be in your current workspace.

2 Select Entry>Library>Print Directory from the menu bar, or select Print Directory from the pop-up menu.

PocketBuilder sends the library directory report to the printer specified under File>Printer Setup in the menu bar.

CHAPTER 5

Using Source Control

About this chapter

PocketBuilder provides a direct connection to external SCC-compliant source control systems. This chapter describes how to work with source control.

Contents

About source control systems

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

What source control systems do

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

Why use a source control system

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

Source control interfaces

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

PocketBuilder institutes source control at the object level. This gives you much finer control of the source code than if you copied your PKLs directly to source control outside of the PocketBuilder SCC API.

No other interfaces
PocketBuilder does not support working with source control systems through proprietary interfaces provided by source control vendors. To work with source control systems from your PocketBuilder workspace, you must use the PocketBuilder SCC API. PocketBuilder also uses this API to connect to the PowerBuilder Native check in/check out utility that installs with the product.

Using your source control manager

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

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

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

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

  For information on configuring a source control connection profile, see “Setting up a connection profile” on page 120.

Which tool to use

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

Table 5-1: Where to perform source control operations

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

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

Using PBNative

PocketBuilder enables you to use the PBNative check in/check out utility that ships with PowerBuilder (but not with PocketBuilder). PBNative allows you to lock the current version of PocketBuilder objects and prevents others from checking out these objects while you are working on them. It provides minimal versioning functionality, and does not allow you to add comments or labels to objects that you add or check in to the PBNative project directory.

Connecting to PBNative

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

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

For more information about PKG files, see "Editing the PKG file for a source-controlled target" on page 144.

PRP files

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

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

For more information on the Show History dialog box, see "Displaying the source control version history" on page 141. For information on displaying the version number in the Library painter, see "Displaying libraries and objects" on page 92.

Using Show Differences functionality with PBNative

PBNative has an option that allows you to see differences between an object on the server and an object on the local machine using a 32-bit visual difference utility that you must install separately. For information on setting up a visual difference utility for use with PBNative, see "Comparing local objects with source control versions" on page 139.

Constraints of a multiuser environment

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

Project manager's tasks

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

- Sets up source control projects (and archive databases)
- Assigns each developer permission to access the new project
- Sets up the directory structure for all targets in a project

  Ideally, the project manager should create a subdirectory for each target. Whatever directory structure is used, it should be copied to all machines used to check out source-controlled objects.
- Distributes the initial set of PKLs and target (PKT) files to all developers working on the project or provides a network location from which these files and their directory structure can be copied.

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

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

Getting the latest version of all targets in a workspace

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

Unfortunately, this does not work well for large PowerScript projects with multiple PKLs and complicated inheritance schemes.

Ongoing maintenance tasks of a project manager typically include:

- Distributing any target (PKT) files and PKLs that are added to the workspace during the course of development, or maintaining them on a network directory in an appropriate hierarchical file structure
- Making sure the PKL mapping files (PKGs) do not get out of sync

  For information about the PKG files, see "Editing the PKG file for a source-controlled target" on page 144.

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

Developers' tasks

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

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

For more information about getting the latest version of objects in source control, see "Synchronizing objects with the source control server" on page 136.

Extension to the SCC API

Status determination by version number

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

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

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

For more information on viewing the version number, see "Displaying libraries and objects" on page 92.

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

Overriding the version number

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

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

For more information about the PKC file, see "Working in offline mode" on page 126.

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

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

Using a source control system with PocketBuilder

PocketBuilder provides a direct connection to external SCC-compliant source control systems.

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

The source control connection profile assigns a PocketBuilder workspace to a source control project. Setting up a source control project is usually the job of a project manager or administrator. See “Project manager’s tasks” on page 117.

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

Setting up a connection profile

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

Local connection options

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

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

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

Table 5-2: Source control properties for a workspace

Select this option	To do this
Log All Source Management Activity (not selected by default)	Enable trace logging. By default the log file name is *PKSCCxx.LOG*, where *xx* is the PocketBuilder version number. The log file is saved in your workspace directory, but you can select a different path and file name.
Append To Log File (default selection when logging is enabled)	Append source control activity information to named log file when logging is enabled.
Overwrite Log File (not selected by default)	Overwrite the named log file with source control activity of the current session when logging is enabled.
Require Comments On Check In (not selected by default; not available for PBNative source control)	Disable the OK button on the Check In dialog box until you type a comment.
This Project Requires That I Sometimes Work Offline (not selected by default)	Disable automatic connection to source control when you open the workspace.
Delete PowerBuilder Generated Object Files (not selected by default)	Remove object files (such as SRDs) from the local directory after they are checked in to source control. This may increase the time it takes for PocketBuilder to refresh source control status, but it minimizes the drive space used by temporary files. You cannot select this option for the Perforce, ClearCase, or Continuus source control systems.
Perform Diff On Status Update	Permit display of out-of-sync icons for local objects that are different from objects on the source control server. Selecting this also increases the time it takes to refresh source control status.You cannot select this option for Perforce.

Select this option	To do this
Suppress prompts to overwrite read-only files	Avoid message boxes warning that read-only files exist on your local project directory.
Status Refresh Rate (5 minutes by default)	Specifies the minimum time elapsed before PocketBuilder automatically requests information from the source control server to determine if objects are out of sync. Valid values are between 1 and 59 minutes. Status refresh rate is ignored when you are working offline.

Advanced connection options

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

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

❖ **To set up a connection profile:**

1. Right-click the Workspace object in the System Tree (or in the Tree view of the Library painter) and select Properties from the pop-up menu.
2. Select the Source Control tab from the Workspace Properties dialog box.
3. From the Source Control System drop-down list, select the system you want to use.

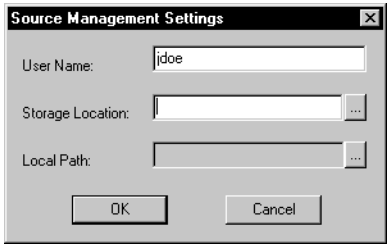
   Only source control systems that are defined in your registry (*HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders*) appear in the drop-down list.
4. Type in your user name for the source control system.

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

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

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

The dialog box displayed for PBNative is shown below:



6 Fill in the information required by your source control system and click OK.

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

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

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

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

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

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

10 Click Apply or click OK.

Viewing the status of source-controlled objects

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

Source control icons

The icons and their meanings are described in Table 5-3 and Table 5-4.

Table 5-3: Source control status icons in PocketBuilder

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

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

Table 5-4: Source control status icons with multiple checkouts enabled

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

For more information on allowing multiple user checkouts, see "Checking objects out from source control" on page 131.

Pop-up menus

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

Library painter Entry menu

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

Library painter selections

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

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

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

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

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

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

For more information, see "Displaying libraries and objects" on page 92.

Working in offline mode

Viewing status information offline

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

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

For more information about setting source control options for your workspace, see "Setting up a connection profile" on page 120.

About the PKC file

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

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

Backing up the PKC file

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

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

For information about refreshing the status cache, see "Refreshing the status of objects" on page 138.

Fine-tuning performance for batched source control requests

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

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

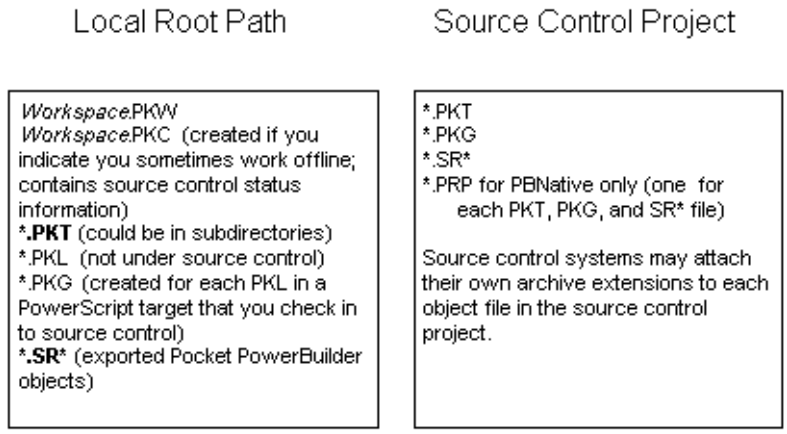
```
[Library]
SccMaxArraySize=nn
```

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

Files available for source control

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

Figure 5-1: Directory structure in local path and source control server



Typically, the source control server files are stored in a database but preserve the file system structure.

Temporary files in local root path

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

Source control operations in PocketBuilder

The following source control operations are described in this section:

- Adding objects to source control
- Checking objects out from source control
- Checking objects in to source control
- Clearing the checked-out status of objects
- Synchronizing objects with the source control server
- Refreshing the status of objects
- Comparing local objects with source control versions
- Displaying the source control version history
- Removing objects from source control

Source control operations on workspace and PKL files are performed on the objects contained in the current workspace or in target PKLs, not on the actual PKW and PKL files. The PKW and PKL files cannot be added to source control through the PocketBuilder interface. Source control operations are not enabled for target PKD files or for any of the objects in target PKD files.

Adding objects to source control

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

What happens when you add objects to source control

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

PocketBuilder creates read-only object files in the local root directory for each PocketBuilder object that you add to source control. These files can be automatically deleted if you selected the Delete PocketBuilder Generated Object Files option as a source control connection property (although you cannot do this for certain SCC systems such as Perforce or ClearCase). Read-only attributes are not changed by PocketBuilder if you later remove a workspace containing these files from source control.

Adding multiple objects to source control

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

- Select multiple files contained in the target
- Register the target file only

If you select the multiple files radio button, another dialog box displays with a list of objects to add to source control. A check box next to each object lets you select which objects you want to add to source control. By default, check boxes are selected for all objects that are not in your source control project. They are not selected for any object already under source control.

For all source control dialog boxes listing multiple files, you can resize the list by placing a cursor over the edge of the dialog box until a two-headed arrow displays, then dragging the edge in the direction of one of the arrow heads.

Selecting multiple files from a PKL

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

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

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

Creating a mapping file for target PKLs

When you add a target, or an object in a target that is not under source control, to source control, PocketBuilder creates a PKG file. The PKG files are used by PocketBuilder to make sure that objects are distributed to the correct PKLs and targets when you check the objects out (or get the latest versions of the objects) from source control.

A PKG file maps objects in a target to a particular PKL in a PowerScript target. One PKG file is created per PKL, so there can be multiple PKG files per PowerScript target. If a PKG file already exists for a target PKL containing the object you are adding to source control, PocketBuilder checks the PKG file out of source control and adds the name of the object to the names of objects already listed in the PKG file. It then checks the PKG file back in to source control.

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

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

For more information on modifying PKG files, see "Editing the PKG file for a source-controlled target" on page 144.

Checking objects out from source control

Enabling multiple user checkout

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

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

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

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

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

What happens

When you check out an object, PocketBuilder:

- Locks the object in the archive so that no one else can modify it (unless you set the source control system to allow shared checkouts)
- Copies the object to the directory for the target to which it belongs
- For a PowerScript object, compiles the object and regenerates it in the target PKL to which it is mapped
- Displays a check mark icon next to the object in your System Tree and in your Library painter to show that the object has been checked out

Checking out multiple objects

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

- Select multiple files contained in the target
- Check out the target file only

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

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

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

Creating a source control branch

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

If you want just a read-only copy of the latest version of an object
Instead of checking out an object and locking it in the source control system, you can choose to get the latest version of the object with a read-only attribute. See "Synchronizing objects with the source control server" on page 136.

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

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

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

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

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

Checking objects in to source control

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

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

Checking in multiple objects

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

- Select multiple files contained in the target
- Check in the target file only

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

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

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

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

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

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

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

Clearing the checked-out status of objects

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

Clearing the status of multiple objects

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

- Select multiple files contained in the target
- Undo the checked-out status for the target file only

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

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

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

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

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

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

Synchronizing objects with the source control server

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

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

- Yes To All, to overwrite all files in your selection.
- No To All, to cancel the synchronization for all objects in the selection that have writable files in the local path.

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

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

For more information, see "Refreshing the status of objects" on page 138.

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

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

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

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

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

Refreshing the status of objects

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

For more information about source control options you can set on your workspace, see "Setting up a connection profile" on page 120.

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

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

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

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

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

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

For information about the meaning of source control icons in PocketBuilder, see "Viewing the status of source-controlled objects" on page 124.

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

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

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

Comparing local objects with source control versions

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

Setting up PBNative for object comparisons

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

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

Option	Select this if
Enclose file names in double quotes	Your visual difference utility does not handle spaces in file names
Refer to local PKL entry as argument #1	You do not want the visual difference utility to use the repository object as the first file in a file comparison
Generate short (8.3) file names	Your visual difference utility does not handle long file names
Generate an extra space prior to file arguments	Your visual difference utility requires an extra space between files that are listed as arguments when you open the utility from a command line

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

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

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

2 Click Connect.

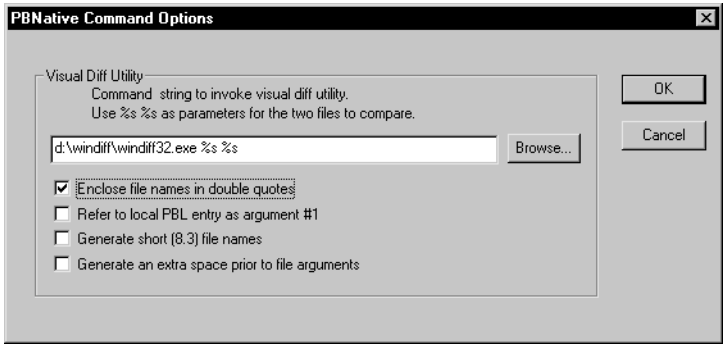
The Connect button is disabled if you are already connected to source control.

3 Click Advanced.

The PBNative Options dialog box displays.

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

Typically, you would add two *%s* parameter markers to indicate where PocketBuilder should perform automatic file name substitution. The following figure shows a setting used to call the Microsoft WinDiff utility:



5 (Optional) Select any or all of the check box options in the PBNative Command Options dialog box for your object comparisons.

6 Click OK twice.

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

Using Show Differences to compare objects

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

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

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

A dialog box from your source control system displays.

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

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

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

Displaying the source control version history

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

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

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

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

A dialog box from your source control system displays.

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

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

Removing objects from source control

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

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

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

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

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

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

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

Modifying source-controlled targets and objects

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

Effects of source control on object management

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

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

For information on removing an object from source control, see “Removing objects from source control” on page 142.

Copy and move operations on source-controlled objects

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

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

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

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

1 Export the object from the first PKL.

2 Remove the object from source control.

See “Removing objects from source control” on page 142.

3 Delete the object from the first PKL.

4 Import the object into the second PKL.

5 Register the object in source control once again.

Editing the PKG file for a source-controlled target

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

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

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

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

You can manually add objects to the PKG file for a PocketBuilder library by including a new line for each object after the `@begin Objects` line.

PART 3

Coding Fundamentals

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

CHAPTER 6

Writing Scripts

About this chapter

PocketBuilder applications are event driven. You specify the processing that takes place when an event occurs by writing a script. This chapter describes how to use the Script view to write scripts using the PowerScript language.

Contents

For more information

For complete information about the PowerScript language, see the *PowerScript Reference* in the online Help.

About the Script view

You use the Script view to code functions and events, define your own functions and events, and declare variables and external functions.

Script views are part of the default layout in the Application, Window, User Object, Menu, and Function painters. In Application, Window, and User Object painters, the initial layout has one Script view that displays the default event script for the object and a second Script view set up for declaring instance variables. You can open as many Script views as you need, or perform all coding tasks in a single Script view.

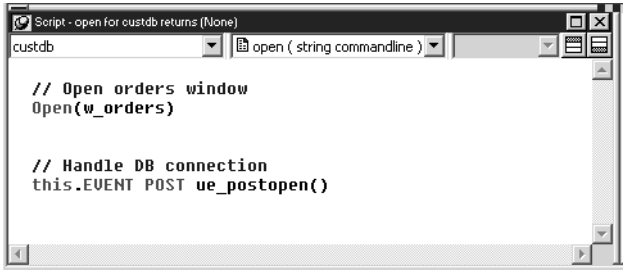
Titlebar

The Script view's titlebar shows the name and return type of the current event or function, as well as the name of the current control for events and the argument list for functions. If the Script view is being used to declare variables or functions, the titlebar shows the type of declaration.

Dropdown lists

There are three drop-down lists at the top of the Script view.

Figure 6-1: Script view for application Open event



In the first list, you can select the object, control, or menu item for which you want to write a script. You can also select Functions to edit function scripts or Declare to declare variables and external functions.

The second list lets you select the event or function you want to edit or the kind of declaration you want to make. A script icon next to an event name indicates there is a script for that event, and the icon's appearance tells you more about the script.

Table 6-1: Script icons in the Script view

If there is a script	The script icon displays
For the current object or control	With text
In an ancestor object or control only	In color
In an ancestor as well as in the object or control you are working with	Half in color

The same script icons display in the Event List view.

The third list is available in descendent objects. It lists the current object and all its ancestors so that you can view scripts in the ancestor objects.

Toggle buttons for Prototype and Error windows

A Prototype window displays at the top of the Script view when you define a new function or event. An Error window displays at the bottom of the view when there are compilation errors. You can toggle the display of these windows with the two toggle buttons to the right of the lists.

For more information about the Prototype window, see Chapter 7, "Working with User-Defined Functions," and Chapter 8, "Working with User Events."

Opening Script views

If there is no open Script view, selecting a menu or PainterBar item that requires a Script view opens one automatically. If you want to edit more than one script at a time, you can open additional Script views from the View menu.

❖ **To open a new Script view:**

- Select View>Script from a painter menu bar

❖ **To edit a script for a control:**

- Double-click a scriptable control, or select Script from the PainterBar or a pop-up menu

  The Script view shows the default script for the control. If the Script view is in a stacked pane and is hidden, it pops to the front. If there is no open Script view, PocketBuilder creates a new one.

Using drag and drop
If a Script view is visible, you can drag a control from the Control list view to the Script view to edit a script for the control.

❖ **To edit a script for a function or event:**

- Double-click an item in the Event list or Function list view of a painter

  If the Script view is in a tabbed pane and is hidden, it pops to the front. If there is no open Script view, PocketBuilder creates a new one. The Script view shows the script for the selected event or function. You can select a different function or event from the second drop-down list in an open Script view.

Modifying Script view properties

The Script view automatically:

- Color-codes scripts to identify datatypes, system-level functions, flow-of-control statements, comments, and literals
- Indents the script based on flow-of-control statements

You can modify these and other properties.

Some properties are shared
Some properties you specify for the Script view also affect the File editor, Source editor, Debugger, and the Interactive SQL and Activity Log views in the Database painter.

❖ **To specify Script view properties:**

1 Select Design>Options to display the Options dialog box for the painter.

The Options dialog box includes four tab pages that affect the Script view: Script, Font, Coloring, and AutoScript.

2 Choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
Tab size, automatic indenting, whether dashes are allowed in identifiers, and which compiler and database messages display	Script
Font family, size, and color for the Script view	Font
Text and background coloring for PowerScript syntax elements	Coloring
Whether AutoScript is enabled and what kind of assistance it provides	AutoScript

Editing scripts

You can perform standard editing tasks in the Script view using the Edit menu, the pop-up menu in the Script view, or the PainterBars. There are shortcuts for many editing actions.

Setting up shortcuts
In a painter with a Script view, select Tools>Keyboard Shortcuts. Expand the Edit menu to view existing shortcuts and set up your own shortcuts.

Printing scripts

You can print a description of the object you are editing, including all its scripts, by selecting File>Print from the menu bar. To print a specific script, select File>Print Script.

Pasting information into scripts

You can paste the names of variables, functions, objects, controls, and other items directly into your scripts. (You can also use AutoScript. See "Using AutoScript" on page 155.) If what you paste includes commented text that you need to replace, such as function arguments or clauses in a statement, you can use Edit>Go To>Next Marker to move your cursor to the next commented item in the template.

Table 6-2: Pasting information into scripts

To paste	Use
PocketBuilder objects and their properties, functions, and events	System Tree
Properties, datatypes, functions, structures, variables, and objects	Browser
Contents of clipboard	Edit>Paste
Contents of Clipboard window	Drag and drop
Objects, controls, arguments, and global and instance variables	Paste buttons on PainterBar *or* Edit>Paste Special
PowerScript statements	Paste Statement button *or* Edit>Paste Special>Statement
SQL statements	Paste SQL button *or* Edit>Paste Special>SQL
Built-in, user-defined, and external functions	Paste Function button *or* Edit>Paste Special>Function
Contents of text files	Edit>Paste Special>From File

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

Some of these techniques are explained in the sections that follow.

Using the System Tree

To paste the name of a PocketBuilder object, or of any of its properties, functions, or events, select the item you want to paste in the System Tree and drag it into your script.

Using the Browser

You can use the Browser to paste the name of any property, datatype, function, structure, variable, or object in the application.

Most tab pages in the Browser have two panes. The left pane displays one type of object, such as a window or menu. The right pane displays the properties, events, functions, external functions, instance variables, shared variables, and structures associated with the object.

Getting context-sensitive Help in the Browser
To get context-sensitive Help for an object, control, or function, select Help from its pop-up menu.

❖ **To use the Browser to paste information into the Script view:**

1 Click the Browser button in the PowerBar, or select Tools>Browser.

2 Select the PowerScript target you want to browse.

3 Select the appropriate tab and then select the object in the left pane.

4 Select the category of information you want to display by expanding the appropriate folder in the right pane.

5 Select the information and click Copy.

6 In the Script view, move the cursor where you want to paste the information and select any text you want to replace.

7 Select Paste from the pop-up menu.

PocketBuilder displays the information at the insertion point in the script, replacing any selected text.

Pasting statements

You can paste a template for all basic forms of the following PowerScript statements:

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

- CHOOSE CASE
- TRY ... CATCH ... FINALLY

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

❖ **To paste a PowerScript statement into the script:**

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

2 Select the Paste Statement button from the PainterBar, or select Edit>Paste Special>Statement from the menu bar.

3 From the cascading menu, select the statement you want to paste.

The statement prototype displays at the insertion point in the script.

4 Replace the prototype values with the conditions you want to test and the actions you want to take based on the test results.

For more about PowerScript statements, see the *PowerScript Reference* in the online Help.

Pasting SQL

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

❖ **To paste a SQL statement:**

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

2 Click the Paste SQL button in the PainterBar, or select Edit>Paste Special>SQL from the menu bar.

3 From the cascading menu, select the type of statement you want to insert.

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

4 Create the statement, then return to the Script view.

The statement displays at the insertion point in the workspace.

For more about embedding SQL in scripts, see the *PowerScript Reference* in the online Help.

Pasting functions

You can paste any function into a script.

❖ **To paste a function into a script:**

1. Place the insertion point where you want to paste the function in the script.
2. Click the Paste Function button in the PainterBar, or select Edit>Paste Special>Function from the menu bar.
3. Choose the type of function you want to paste: built-in, user-defined, or external.
4. Double-click the function you want from the list that displays.

   PocketBuilder pastes the function into the script and places the cursor within the parentheses so that you can define any needed arguments.

For more about pasting user-defined functions, see “Pasting user-defined functions” on page 178. For more about external and built-in functions, see the *Resource Guide*.

Pasting contents of files

If you have code that is common across different scripts, you can keep that code in a text file, then paste it into new scripts you write. For shorter snippets of code, you can also use the Clip window. See “The Clip window” on page 11.

❖ **To import the contents of a file into the Script view:**

1. Place the insertion point where you want the file contents pasted.
2. Select Edit>Paste Special>From File from the menu bar.

   The Paste From File dialog box displays, listing all files with the extension *SCR*. If necessary, navigate to the directory that contains the script you want to paste.
3. Choose the file containing the code you want. You can change the type of files displayed by changing the file specification in the File Name box.

   PocketBuilder copies the file into the Script view at the insertion point.

Saving a script to a file

To save all or part of a script to an external text file, select the code you want to save and copy and paste it to the File editor. Use the extension *SCR* to identify it as PowerScript code. You might want to use this technique to save a backup copy before you make major changes or so that you can use the code in other scripts.

Reverting to the unedited version of a script

You can discard the edits you have made to a script and revert to the unedited version by selecting Edit>Revert Script from the menu.

Using AutoScript

AutoScript is a tool designed to help you write PowerScript code more quickly by providing a lookup and paste service inside the Script view. It is an alternative to using the paste toolbar buttons or the Browser. It allows you to paste functions, events, variables, properties, and templates for TRY, DO, FOR, IF, and CHOOSE statements into your script without moving your hands away from the keyboard.

If you are not sure what the name or syntax of a function is or what the names of certain variables are, AutoScript can show you a list to choose from. You can paste what you need right into the script. If you can remember part of the name, start typing and select Edit>Activate AutoScript (or do nothing if automatic pop-up is turned on). If you cannot remember the name at all, right-click in Script view white space and select Activate AutoScript from the pop-up menu.

Where you use AutoScript

You can use AutoScript in three different contexts:

- When you can remember part of a name and you want AutoScript to finish typing it for you or show you a list of alternatives.
- When you cannot remember a name or you just want a list. AutoScript options can help you narrow the list if you do not know the name but you do know the type you are looking for. For example, you can choose to see a list showing all variables, or only all local variables.
- When you want a list of the properties and/or functions and events that apply to an identifier followed by a dot.

For how to use AutoScript options, see “Customizing AutoScript” on page 157.

Two ways to use AutoScript

AutoScript can pop up a list automatically when you pause while typing, or when you request it:

- Turn automatic pop-up on to have AutoScript pop up the list or complete what you are typing. It does this when you pause for a few seconds after typing one or more characters or an identifier followed by a dot. See “Using automatic pop-up” on page 160.

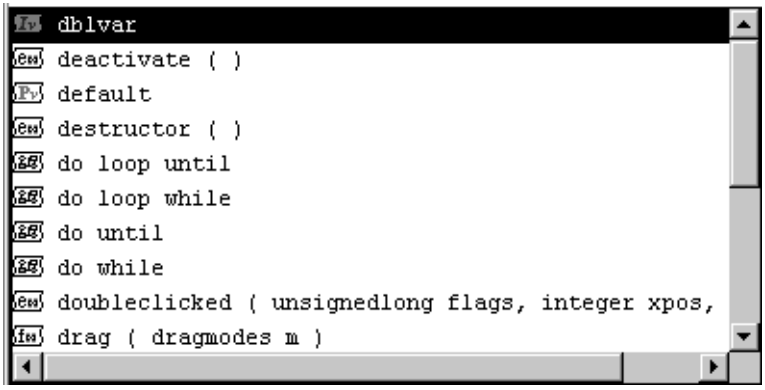
- Invoke AutoScript when you need it by selecting Edit>Activate AutoScript when you have typed one or more characters or an identifier followed by a dot, or when the cursor is in white space. This activates AutoScript only once. It does not turn automatic pop-up on.

For how to paste an item from the pop-up window into a script, see "Using the AutoScript pop-up window" next.

Using the AutoScript pop-up window

If there is more than one property, variable, method, or statement that could be inserted, AutoScript pops up an alphabetical list of possible completions or insertions. An icon next to each item indicates its type. The following screen includes an instance variable, events, properties, statements, and functions:

Figure 6-2: The AutoScript pop-up window showing script choices



If a function is overloaded, each version displays on a different line in the AutoScript pop-up window.

If you have started typing a word, only completions that begin with the string you have already typed display in the list.

Case sensitivity
AutoScript always pastes lowercase characters, but the case of any characters you have already typed is preserved. For example, if you are using AutoScript to complete a function name and you want to use mixed case, you can type up to the last uppercase letter before invoking AutoScript. AutoScript completes the function name and pastes an argument template.

Pasting an item into the script

To paste an item into the script, press Tab or Enter or double-click the item. Use the arrow and page up and down keys to scroll through the list. If the item is a function, event, or statement, the template that is pasted includes descriptive comments that you replace with argument names, conditions, and so forth. The first commented argument or statement is selected so that it is easy to replace. You can jump to the next comment by selecting Edit>Go To>Next Marker.

Go to next marker
You can use Edit>Go To>Next Marker to jump to the next comment enclosed by `/*` and `*/` anywhere in the Script view, not just in AutoScript templates. For how to create a shortcut for this menu item, see "Customizing AutoScript" next.

If you do not want to paste from the list

To dismiss the pop-up window without pasting into the script, press the Backspace key or click anywhere outside the pop-up.

If nothing displays

AutoScript does not pop up a list if the cursor is in a comment or string literal, or if an identifier is complete. If neither condition is true and nothing displays when you select Edit>Activate AutoScript, there may be no appropriate completions in the current context. Check that the options you need are selected on the AutoScript options page as described in "Customizing AutoScript" next.

Customizing AutoScript

There are four ways to customize AutoScript:

- Creating shortcut keys
- Specifying what displays in the AutoScript list
- Using automatic pop-up
- Using AutoScript only with dot notation

Creating shortcut keys

AutoScript is easier to use if you create shortcuts for the menu items that you use frequently. For example, the Edit>Activate AutoScript menu item uses F8 as a shortcut key by default, but you can select a different shortcut key or key combination for this item.

❖ **To modify or create shortcut keys for using AutoScript:**

1 Select Tools>Keyboard Shortcuts from the menu bar and expand the Edit menu in the Keyboard Shortcuts dialog box.

2 If you want to select a default key for activating AutoScript, scroll down and select Activate AutoScript and type a key sequence, such as Ctrl+Space.

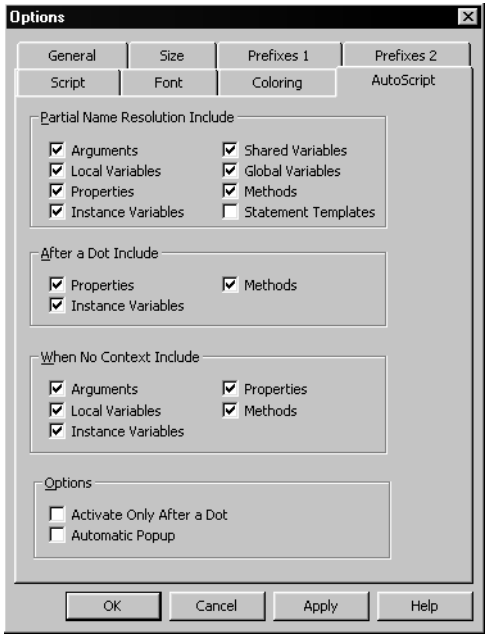
3 Expand the Go To menu, select Next Marker, and type a key sequence, such as Ctrl+M.

After you click OK, the shortcuts display in the Edit menu.

AutoScript display customizations

You can change what gets displayed when you activate AutoScript. You make these changes in the AutoScript page of the Options dialog box that you open from a painter's Design>Options menu.

Figure 6-3: The AutoScript page of the Options dialog box



You can select different items to include in these different contexts:

- When you have started typing a variable or method name or the beginning of a PowerScript statement
- When you have typed the name of an object followed by a dot
- When the cursor is at the beginning of a new line or in white space

Specifying what displays in the AutoScript list

Table 6-3 shows what is included in an AutoScript list or pasted into a Script view when you make particular selections on the AutoScript page of the Options dialog box.

Table 6-3: Setting options for AutoScript in the Options dialog box

Check box	Displays
Arguments	Arguments for the current function or event.
Local Variables	Variables defined in the current script.
Instance Variables	Variables defined for and associated with an instance of the current object or, after a dot, variables associated with the object preceding the dot.
Shared Variables	Variables defined for the current object and associated with all instances of it.
Global Variables	Variables defined for the current application.
Properties	Properties for the current object or, after a dot, properties for the object preceding the dot. Includes controls on the current window.
Methods	Functions and events for the current object or, after a dot, functions and events for the object preceding the dot.
Statement Templates	PowerScript statement templates for each type of IF, FOR, CHOOSE CASE, TRY, or DO statement with comments indicating what code should be inserted. This option is off by default.

Turning options off reduces the length of the list that displays when you invoke AutoScript. This makes it faster and easier to paste a completion or insert code into the script.

- To show all variables and methods when typing, check all the boxes except Statement Templates in the Partial Name Resolution Include group box. When you pause or select Edit>Activate AutoScript from the main menu, the list shows variables and methods that begin with the string you typed.
- To find functions for an object quickly, clear all the boxes except Methods in the After A Dot Include group box. When you type an instance name followed by a dot, only function and event names for the instance display.
- To see a list of arguments and local variables when the cursor is in white space, check the Arguments and Local Variables boxes in the When No Context Include group box. When you select Edit>Activate AutoScript, the list shows only arguments and local variables.

Using automatic pop-up

Most of the time you are likely to use a shortcut key to invoke AutoScript, but you can also have AutoScript pop up a list or paste a selection automatically whenever you pause for several seconds while typing. To do so, check the Automatic Popup box on the AutoScript options page. Automatic pop-up does not operate when the cursor is at the beginning of a line or in white space.

This feature is most useful when you are entering new code. You can customize the options in the Partial Name Resolution Include and After A Dot Include group boxes to reduce the number of times AutoScript pops up.

When you are editing existing code, it is easier to work with automatic pop-up off. AutoScript might pop up a list or paste a template for a function when you do not want it to. Using only the shortcut key to invoke AutoScript gives you complete control.

Using AutoScript only with dot notation

If you want AutoScript to work only when you have typed an identifier followed by a dot, check the Activate Only After a Dot box on the AutoScript options page. The effect of checking this box applies whether or not you have checked Automatic Popup. You might find it most useful when you have checked Automatic Popup, because it provides another way to limit the number of times AutoScript pops up automatically.

Example

The following simple example illustrates how AutoScript works with automatic pop-up turned off and different settings for each context. To set up the example:

1. Create a new window and place on it a DataWindow control and a CommandButton control.
2. Select all the boxes in the Partial Name Resolution Include group box.
3. Clear all the boxes in the After A Dot Include group box except Methods.
4. Clear all the boxes in the When No Context Include group box except Arguments and Local Variables.
5. Clear both boxes in the Options group box.

Table 6-4: AutoScript example

Context	Do this	What happens
Partial name resolution	In the Clicked event script for cb_1, type `long ll_rtn`. On a new line, type `ll` and select Edit>Activate AutoScript.	AutoScript pastes the local variable ll_rtn into the script because it is the only completion that begins with `ll`.
	Type `= d` and select Edit>Activate AutoScript.	The list displays all properties, events, functions, variables, and statements that begin with `d`.
	Type `w` and press Tab or Enter.	The list scrolls to dw_1 and AutoScript pastes it into the script when you press Tab or Enter.
After a dot	Type a dot after dw_1 and select Edit>Activate AutoScript.	The list shows all the functions and events for a DataWindow control.
	Type `GetNextM` and press Tab or Enter.	AutoScript pastes the rest of the GetNextModified function name and template into the script, retaining your capitalization.
	Select Edit>Go To>Next Marker.	AutoScript selects the next function argument so you can replace it. Complete or comment out the statement.
No context	In the empty ItemChanged event for dw_1, declare some local variables, press Tab or Enter, and select Edit>Activate AutoScript.	The list displays the local variables and the arguments for the ItemChanged event.

Getting context-sensitive Help

In addition to accessing Help through the Help menu and F1 key, you can use context-sensitive Help in the Script view to display Help for reserved words and built-in functions.

❖ **To use context-sensitive Help:**

1 Place the insertion point within a reserved word (such as DO or CREATE) or built-in function (such as Open or Retrieve).

2 Press Shift+F1.

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

Copying Help text
You can copy text from the Help window into the Script view. This is an easy way to get more information about arguments required by built-in functions.

Compiling the script

Before you can execute a script, you must compile it.

❖ **To compile a script:**

- Click the Compile button, or select Edit>Compile from the menu bar.

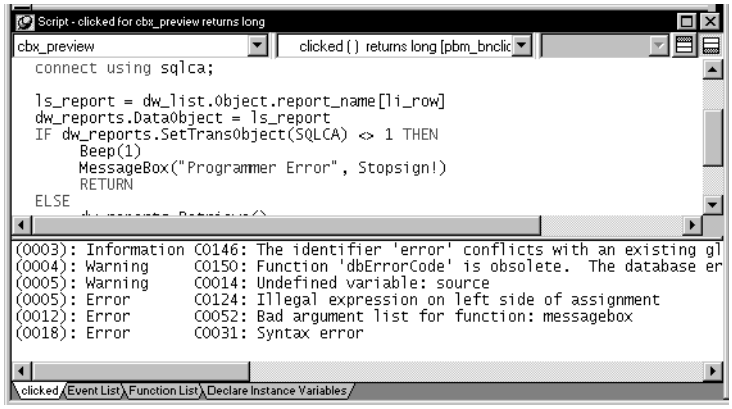
  PocketBuilder compiles the script and reports any problems it finds, as described in "Handling problems" next.

PocketBuilder compiles automatically
When you attempt to open a different script in a Script view, PocketBuilder compiles the current script. When you save the object, such as the window containing a control you wrote a script for, PocketBuilder recompiles all scripts in the object to make sure they are still valid. For example, PocketBuilder checks that all objects that were referenced when you wrote the script still exist.

Handling problems

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

Figure 6-4: Example of a script compilation error message



There are three kinds of messages:

- Errors
- Warnings
- Information messages

Understanding errors

Errors indicate serious problems that you must fix before a script will compile and before you can close the Script view or open another script in the same view. Errors are shown in the Message window as:

```
line number: Error error number:message
```

Understanding warnings

Warnings indicate problems that you should be aware of but that do not prevent a script from compiling.

There are three types of warnings:

Compiler warnings Compiler warnings inform you of syntactic problems, such as undeclared variables. PocketBuilder lets you compile a script that contains compiler warnings, but you must fix the problem in the script before you can save the object that the script is for, such as the window or menu. Compiler warnings are shown in the Message window as:

```
line number: Warning warning number:message
```

Obsolete warnings Obsolete warnings inform you when you use any obsolete functions or syntax in your script. Obsolete functions, although they still compile and run, have been replaced by more efficient functions. You should replace all references to obsolete functions as soon as possible. Obsolete warnings are shown in the Message window as:

```
line number: Warning warning number:message
```

Database warnings Database warnings come from the database manager you are connected to. PocketBuilder connects to the database manager when you compile a script containing embedded SQL. Typically, these warnings arise because you are referencing a database you are not connected to. Database warnings are shown in the Message window as:

```
line number: Database warning number:message
```

PocketBuilder lets you compile scripts with database warnings and also lets you save the associated object. It does this because the execution environment might be different from the compile-time environment, and the problem might not apply during execution.

You should study database warnings carefully to make sure the problems will not occur during execution.

Understanding Information messages

Information messages are issued when there is a potential problem. For example, an information message is issued when you have used a global variable name as a local variable, because that might result in a conflict later.

Information messages are shown in the Message window as:

```
line number: Information number:message
```

Displaying warnings and messages

To specify which messages display when you compile, select Design>Options to open the Options dialog box, select the Script tab page, and check or clear the Display Compiler Warnings, Display Obsolete Messages, Display Information Messages, and Display Database Warnings check boxes. The default is to display compiler and database warning messages. Error messages always display.

Fixing problems

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

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

Declaring variables and external functions

The default layout in the Application, Window, and User Object painters includes a Script view set up to declare variables. Keeping a separate Script view open makes it easy to declare any variables or external functions you need to use in your code without closing and compiling the script.

❖ **To declare variables and external functions:**

1 Select [Declare] from the first list in the Script view.

2 Select the variable type (instance, shared, or global) or the function type (local or global) from the second list.

3 Type the declaration in the Script view.

For more information about declaring variables, see the *PowerScript Reference* in the online Help. For more information about declaring and using external functions, see the *PowerScript Reference* and the *Resource Guide*.

CHAPTER 7

Working with User-Defined Functions

About this chapter

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

Contents

About user-defined functions

The PowerScript language has many built-in functions, but you might find that you need to code the same procedure over and over again. For example, you might need to perform a certain calculation in several places in an application or in different applications. In such a situation, create a user-defined function to perform the processing.

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

There are two kinds of user-defined functions, global and object-level functions.

Global functions

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

They correspond to the PocketBuilder built-in functions that are not associated with an object, such as the mathematical and string-handling functions. You define global functions in the Function painter.

Object-level functions

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

These functions correspond to built-in functions that are defined for specific PocketBuilder objects such as windows or controls. You define object-level functions in a Script view for the object.

Deciding which kind you want

When you design your application, you need to decide how you will use the functions you will define:

- If a function is general-purpose and applies throughout an application, make it a global function.
- If a function applies only to a particular kind of object, make it an object-level function. You can still call the function from anywhere in the application, but the function acts only on a particular object type.

  For example, suppose you want a function that returns the contents of a SingleLineEdit control in one window to another window. Make it a window-level function, defined in the window containing the SingleLineEdit control. Then, anywhere in your application that you need this value, call the window-level function.

Multiple objects can have functions with the same name

Two or more objects can have functions with the same name that do different things. In object-oriented terms, this is called polymorphism. For example, each window type can have its own Initialize function that performs processing unique to that window type. There is never any ambiguity about which function is being called, because you always specify the object's name when you call an object-level function.

Object-level functions can also be overloaded—two or more functions can have the same name but different argument lists. Global functions cannot be overloaded.

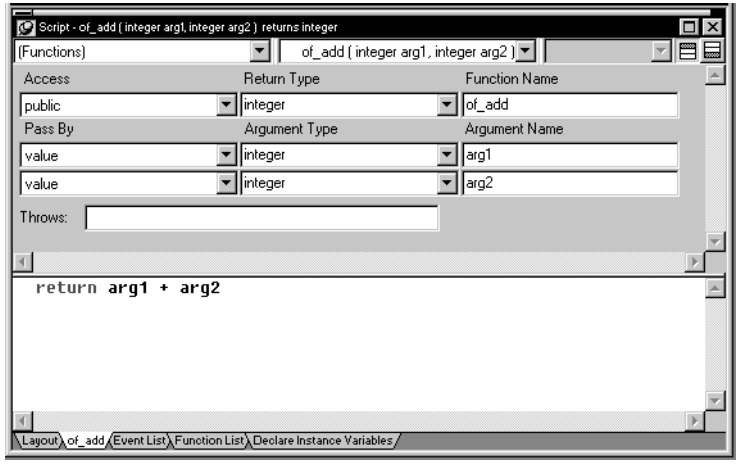
Defining user-defined functions

Although you define global functions in the Function painter and object-level functions in the painter for a specific object, in both cases you define and code the function in a Script view.

When you add a new function, a Prototype window displays above the script area in the Script view. The fields in the Prototype window are in the same order as the function's signature:

- The function's access level, return type, and name
- For each parameter, how it is passed, its datatype, and its name
- The exceptions the function can throw, if any

Figure 7-1: Defining a function prototype



The following sections describe each of the steps required to define and code a new function:

1 Opening a Prototype window to add a new function.

2 Defining the access level (for object-level functions).

3 Defining a return type.

4 Naming the function.

5 Defining arguments.

6 Defining a THROWS clause.

7 Coding the function.

8 Compiling and saving the function.

Opening a Prototype window to add a new function

How you create a new function depends on whether you are defining a global function or an object-level function.

❖ **To create a new global function:**

- Select File>New from the menu bar and select Function from the PB Object tab

  The Function painter opens, displaying a Script view with an open Prototype window in which you define the function.

❖ **To create a new object-level function:**

1 Open the object for which you want to declare a function.

  You can declare functions for windows, menus, user objects, or applications.

2 Select Insert>Function from the menu bar, or select Add from the Function List view pop-up menu.

  The Prototype window opens in a Script view.

Defining the access level

In the Prototype window, use the drop-down list labeled Access to specify where you can call the function in the application.

For global functions

Global functions can always be called anywhere in the application. In PocketBuilder terms, they are public. When you are defining a global function, you cannot modify the access level; the field is read-only.

For object-level functions

You can restrict access to an object-level function by setting its access level, as described in Table 7-1.

Table 7-1: Access levels for object-level functions

Access	Means you can call the function
Public	In any script in the application.
Private	Only in scripts for events in the object in which the function is defined. You cannot call the function from descendants of the object.
Protected	Only in scripts for the object in which the function is defined and scripts for that object's descendants.

If a function is to be used only internally within an object, you should define its access as private or protected. This ensures that the function is never called inappropriately from outside the object. In object-oriented terms, defining a function as protected or private encapsulates the function within the object.

Defining a return type

Many functions perform some processing and then return a value. That value can be the result of the processing or a value that indicates whether the function executed successfully or not. To have your function return a value, you need to define its return type, which specifies the datatype of the returned value.

You must code a return statement in the function that specifies the value to return. See "Returning a value" on page 175. When you call the function in a script or another function, you can use an assignment statement to assign the returned value to a variable in the calling script or function. You can also use the returned value directly in an expression in place of a variable of the same type.

❖ **To define a function's return type:**

- Select the return type from the Return Type drop-down list in the Prototype window, or type in the name of an object type you have defined

  You can specify any PocketBuilder datatype, including the standard datatypes, such as integer and string, as well as objects and controls, such as DataStore or MultiLineEdit.

  You can also specify as the return type any object type that you have defined. For example, if you defined a window named w_calculator and want the function to process the window and return it, type `w_calculator` in the Return Type list. You cannot select w_calculator from the drop-down list, because the list shows built-in data types only.

❖ **To specify that a function does not return a value:**

- Select (None) from the Return Type list

  This tells PocketBuilder that the function does not return a value. This is similar to defining a procedure or a void function in some programming languages.

Examples of functions returning values

The following table shows the return type you would specify for some different types of function.

Table 7-2: Examples of return types for different types of functions

If you are defining	Specify this return type
A mathematical function that does some processing and returns a real number	real
A function that takes a string as an argument and returns the string in reverse order	string
A function that is passed an instance of window w_calculator, does some processing (such as changing the window's color), then returns the modified window	w_calculator

Naming the function

Name the function in the Function Name box. Function names can have up to 40 characters. For valid characters, see “identifier names” in the online Help.

For object-level functions, the function is added to the Function List view when you tab off the Function Name box. It is saved as part of the object whenever you save the object.

Using a naming convention for user-defined functions makes them easy to recognize and distinguish from built-in PowerScript functions. A commonly used convention is to preface all global function names with f_ and object-level functions with of_, such as:

```
// global functions
f_calc
f_get_result

// object-level functions
of_refreshwindow
of_checkparent
```

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

Defining arguments

Like built-in functions, user-defined functions can have any number of arguments, including none. You declare the arguments and their types when you define a function.

Passing arguments

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

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

By value When you pass by value, you are passing the function a temporary local copy of the argument. The function can alter the value of the local copy within the function, but the value of the argument is not changed in the calling script or function.

Read-only When you pass as read-only, the variable's value is available to the function, but it is treated as a constant. Read-only provides a performance advantage over passing by value for string, blob, date, time, and datetime arguments, because it does not create a copy of the data.

If the function takes no arguments
Leave the initial argument shown in the Prototype window blank.

❖ **To define arguments:**

1 Declare whether the argument is passed by reference, by value, or read-only.

   The order in which you specify arguments in the Prototype window is the order you use when calling the function.

2 Declare the argument's type. You can specify any datatype, including:

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

3 Name the argument.

4 If you want to add another argument, press the Tab key or select Add Parameter from the pop-up menu, and repeat steps 1 to 3.

Passing arrays
You must include the square brackets in the array definition, for example, `price[]` or `price[50]`, and the datatype of the array must be the datatype of the argument. For information on arrays, see the *PowerScript Reference* in the online Help.

Defining a THROWS clause

If you are using user-defined exceptions, you must define what exceptions might be thrown from a user-defined function or event. You use the Throws box in the Prototype window to do this.

When you need to add a THROWS clause

Any developers who call the function or event need to know what exceptions can be thrown from it so that their code can handle the exceptions. If a function contains a THROW statement that is not surrounded by a try-catch block that can deal with that type of exception, then the function must be declared to throw that type of an exception or some ancestor of that exception type.

There are two exception types that inherit from the Throwable object: Exception and RuntimeError. Typically, you add objects that inherit from Exception to the THROWS clause of a function. Exception objects are the parents of all checked exceptions, which are exceptions that must be dealt with when thrown and declared when throwing. You do not need to add runtime error objects to the THROWS clause, because they can occur at any time. You can catch these errors in a try-catch block, but you are not required to.

Adding a THROWS clause

You can add a THROWS clause to any PocketBuilder function or to any user event that is not defined by an event ID. To do so, drag and drop an exception object from the System Tree, or type the name of the object in the Throws box. If you type the names of multiple user objects in the Throws box, use a comma to separate the object names. When you drag and drop multiple user objects, PocketBuilder automatically adds the comma separators.

The PocketBuilder compiler checks whether a user-defined exception thrown on a function call in a script matches an exception in the THROWS clause for that function and prompts you if there is no matching exception in the THROWS clause.

You can create a user-defined exception object and inherit from it to define more specific lower-level exceptions. If you add a high-level exception to the THROWS clause, you can throw any lower-level exception in the script, but you risk hiding any useful information obtainable from the lower-level exception.

For more information about exception handling, see the *Resource Guide*.

Coding the function

When you have finished defining the function prototype, you specify the code for the function, just as you specify the script for an event in the Script view. For information about using the Script view, see Chapter 6, "Writing Scripts."

What functions can contain

User-defined functions can include PowerScript statements, embedded SQL statements, and calls to built-in, user-defined, and external functions.

You can type the statements in the Script view, or you can use the buttons in the PainterBar or items on the Edit>Paste Special menu to insert them into the function. For more information, see "Pasting information into scripts" on page 151.

Returning a value

If you specified a return type for your function in the Prototype window, you must return a value in the body of the function. To return a value in a function, use the RETURN statement:

RETURN *expression*

where *expression* is the value you want returned by the function. The datatype of the expression must be the datatype you specified for the return value for the function.

Example

The following function returns the result of dividing *arg1* by *arg2* when *arg2* does not equal zero. It returns –1 if *arg2* equals zero:

```
IF arg2 <> 0 THEN
   RETURN arg1 / arg2
ELSE
   RETURN -1
END IF
```

Compiling and saving the function

When you finish building a function, compile it and save it in a library. Then you can use it in scripts or other user-defined functions in any application that includes the library containing the function in its library search path. You compile the script and handle errors as described in “Compiling the script” on page 162.

Modifying user-defined functions

You can change the definition of a user-defined function at any time. You change the processing performed by the function by modifying the statements in the Script view. You can also change the return type, argument list, or access level for a function.

❖ **To change a function's return type, arguments, or access level:**

1 Do one of the following:

- In the Function painter, open the global function
- Open the object that contains the object-level function you want to edit and select the function from the Function list

2 Make the changes you want in the Prototype window.

If the Prototype window is hidden, click the toggle button to display it.

3 Select File>Save from the menu bar.

❖ **To change a function’s name:**

1 If desired, modify the function’s return type, arguments, or access level as described in the previous procedure.

2 Do one of the following:

- In the Function painter, select File>Save As from the menu bar and enter a name
- In the Script view, enter a new name in the Function Name box

When you tab off the box, the new function name displays in the Function List view.

Adding, inserting, and deleting arguments

You can change a user-defined function's arguments at any time using the pop-up menu in the Prototype window.

- Add an argument by selecting the Add Parameter menu item. Boxes for defining the new argument display below the last argument in the list.
- Insert an argument by moving the pointer to the argument before which you want to insert the new argument and selecting the Insert Parameter menu item. Boxes for defining the new argument display above the selected argument.
- Delete an argument by selecting it and clicking the Delete Parameter menu item.

To change the position of an argument
To change the position of an argument, delete the argument and insert it as a new argument in the correct position.

Recompiling other scripts

Changing arguments and the return type of a function affects scripts and other functions that call that function. You should recompile any script in which the function is used. This guarantees that the scripts and functions work correctly during execution.

Seeing where a function is used

PocketBuilder provides browsing facilities to help you find where you have referenced your functions. In the System Tree or Library painter, select a target, library, or object and select Search from the pop-up menu. You can also search multiple entries in the Library painter.

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

1. Open the Library painter.
2. In a List view, select all the entries you want to search for references to the user-defined function.
3. Select Entry>Search from the menu bar.

   The Search Library Entries dialog box displays.
4. Specify the user-defined function as the search text and specify the types of components you want to search.
5. Click OK.

   PocketBuilder displays all specified components that reference the function in the Output window. You can double-click a listed component to open the appropriate painter.

For more about browsing library entities, see “Searching targets, libraries, and objects” on page 101.

Using your functions

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

For more information about calling functions, see the *Resource Guide*.

Pasting user-defined functions

When you build a script in the Script view, you can type the call to the user-defined function. You can also paste the function into the script. There are several ways to paste a user-defined function into a script:

- Drag the function from the System Tree to the Script view
- Select Edit>Paste Special>Function>User-defined from the menu bar
- Enable AutoScript, select the function’s signature in the list that displays when you pause, and press Tab or Enter
- Select the function in the Browser and copy and paste it into the script

Using the System Tree, AutoScript, or the Browser pastes the function’s prototype arguments as well as its name into the script.

For more information about AutoScript, see “Using AutoScript” on page 155.

❖ **To paste a user-defined function into a script from the Browser:**

1. Select Tools>Browser from the menu bar.
2. Do one of the following:
   - Select a global function from the Function page
   - Select the object that contains the object-level function you want to paste from the corresponding page (such as the Window page)
3. Double-click the Functions category in the right pane.
4. Select the function you want to paste and select Copy from its pop-up menu.

5 In the Script view, move the insertion point to where you want to paste the function and select Paste from the pop-up menu.

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

6 Specify the required arguments.

CHAPTER 8

Working with User Events

About this chapter

This chapter introduces user events, describes how to define them, and discusses how to use them in an application.

Contents

About user events

Windows, user objects, controls, menus, and Application objects each have a predefined set of events. In most cases, the predefined events are all you need, but there are times when you need to declare your own user event. You can use predefined event IDs to trigger a user event, or you can trigger it exclusively from within your application scripts.

Features that you might want to add to your application by creating user events include keystroke processing, communication between a user object and a window, and the ability to perform a task in multiple ways.

Keystroke processing

Suppose you want to modify the way keystrokes are processed in your application. For example, in a DataWindow control, suppose you want the user to be able to press the Down Arrow and Up Arrow keys to scroll among radio buttons in a DataWindow column. Normally, pressing these keys moves the focus to the next or preceding row.

To make this change, you can define user events corresponding to operating system events that PocketBuilder does not define.

Multiple methods

Suppose you want to provide several ways to accomplish a certain task within a window. For example, you want the user to be able to update the database by either clicking a button or selecting a menu item. In addition, you want to provide the option of updating the database when the user closes the window.

To do this, you define a user event to update the database.

Communication between user object and window

Suppose you have placed a custom visual user object in a window and need to pass information between the user object and the window. You can create a user event that communicates this information either synchronously or asynchronously.

For more information, see "Communicating between a window and a user object" on page 327.

User events and event IDs

About event IDs

An event ID connects events related to user actions or system activity to a system message. PocketBuilder defines (or maps) events to commonly used event IDs, and when it receives a system message, it uses the mapped event ID to trigger an event.

User-defined events do not have to be mapped to an event ID. See "Defining user events" on page 183.

The PocketBuilder naming convention for user event IDs is similar to the convention Windows uses to name messages. All PocketBuilder event IDs begin with pbm_.

Event IDs associated with Windows messages

Several Windows messages and notifications map to event IDs.

For Windows messages that begin with wm_, the PocketBuilder event ID typically has the same name with pbm_ substituted for wm_. For messages from controls, the event ID typically has the same name but begins with pbm_ and has the Windows prefix for the control added to the message name. For example:

- wm_keydown maps to pbm_keydown
- bm_getcheck (a button control message) maps to pbm_bmgetcheck
- bn_clicked (a button control notification message) maps to pbm_bnclicked

To see a list of event IDs to which you can map a user-defined event, select Insert>Event and display the Event ID drop-down list in the Prototype window that displays.

Windows messages that are not mapped to an event ID map to the pbm_other event ID. The PocketBuilder Message object is populated with information about system events that are not mapped to event IDs. For more information about the Message object, see the online Help.

For information about Windows messages on Windows Mobile or Windows CE devices, see the Microsoft Web site at http://msdn2.microsoft.com/en-us/library/aa929414.aspx.

Event IDs associated with PocketBuilder events

PocketBuilder has its own events, each of which has an event ID. For example, the PocketBuilder event DragDrop has the event ID pbm_dragdrop. The event name and event ID of the predefined PocketBuilder events are protected; they cannot be modified. The event IDs for predefined events are shown in the Event List view.

Figure 8-1: The Event List view for a window

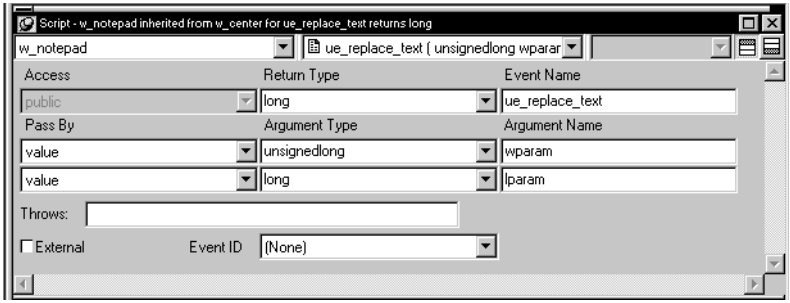
Defining user events

In PocketBuilder, you can define both mapped and unmapped user events for windows, user objects, controls, menus, and the Application object. The access level for events is always public.

When you add a new event, a Prototype window displays above the script area in the Script view. Most of the fields in the Prototype window are the same as when you define a user-defined function. They are in the same order as the event's signature: access level, return type, name, required parameters, and any exception objects for its THROWS clause. For each parameter, you define how it is passed, its datatype, and the parameter name.

For information about filling in these fields, see "Defining user-defined functions" on page 169.

Figure 8-2: Prototype window for a user-defined event



The Prototype window for user-defined events has an additional field that you use if you want to map the user event to an event ID.

External check box
When you check the External check box, PocketBuilder sets the IsExternalEvent property of the ScriptDefinition object associated with the event to "true". This has no effect on your application in this release. The feature might be used in a future release.

Mapped user events

When a system message occurs, PocketBuilder triggers any user event that has been mapped to the message and passes the appropriate values to the event script as arguments. When you define a user event and map it to an event ID, you must use the return value and arguments that are associated with the event ID.

❖ **To define a mapped user event:**

1 Open the object for which you want to define a user event.

2 If you want to define a user event for a control on a window or visual user object, double-click the control to select it.

3 Select Insert>Event from the menu bar or select Add from the Event List view pop-up menu.

The Prototype window opens in the Script view. If you display the Script view's title bar, you see `(Untitled)` because you have not yet named the event.

4 Name the event and tab to the next field.

To recognize a user event easily, consider prefacing the name with an easily recognizable prefix such as ue_. Event names can have up to 40 characters. For valid characters, see "identifier names" in the online Help.

When you tab to the next field, the user event is added to the Event List view. It is saved as part of the object whenever you save the object.

5 Select an ID from the drop-down list box at the bottom of the Prototype window.

Unmapped user events

Unmapped user events are associated with PocketBuilder activities and do not have event IDs. When you define an unmapped user event, you specify the arguments and return datatype; only your application scripts can trigger the user event. For example, if you create an event called ue_update that updates a database, you might trigger or post the event in the Clicked event of an Update command button.

❖ **To define an unmapped user event:**

1 Open the object for which you want to define a user event.

2 If you want to define a user event for a control on a window or visual user object, double-click the control to select it.

3 Select Insert>Event from the menu bar or select Add from the pop-up menu for the Event List view.

The Prototype window opens in the Script view. If you display the Script view's title bar, you see `(Untitled)` because you have not yet named the event.

4 Select a return type and tab to the next field.

Defining return types for events is similar to defining them for functions. See "Defining a return type" on page 171.

5 Name the event and tab to the next field.

To recognize a user event easily, consider prefacing the name with an easily recognizable prefix such as ue_. Event names can have up to 40 characters. For valid characters, see "identifier names" in the online Help.

When you tab to the next field, the user event is added to the Event List view. It is saved as part of the object whenever you save the object.

6 If the event takes arguments, define arguments for the event.

Defining arguments for events is similar to defining them for functions. See "Defining arguments" on page 173 and "Adding, inserting, and deleting arguments" on page 177.

7 Optionally enter the name of exceptions that can be thrown by the event.

❖ **To open a user event for editing:**

- In the Event List view, double-click the event's name

❖ **To delete a user event:**

- In the Event List view, select the user event's name and select Delete from the Edit menu or the pop-up menu

Using a user event

After you define a user event, you must write the script that PocketBuilder will execute when that user event is triggered. If it is an unmapped user event, you also write the code that will trigger the user event.

User events display in alphabetical order in the Event List view and the event drop-down list in the Script view, along with the predefined events. As with predefined events, the script tells PocketBuilder what processing to perform when the user event occurs.

If the user event is not mapped to a Windows message (that is, if there is no event ID associated with it), you must trigger the event in a script. You can trigger the user event in an object using the EVENT syntax. For information about calling events, see the *PowerScript Reference* in the online Help.

Examples of user event scripts

This section includes two examples that use a mapped user event. For more user event examples, see "Communicating between a window and a user object" on page 327.

Example 1: mapped user event for a control

Situation You have several SingleLineEdit controls in a window and want the Enter key to behave like the Tab key. For example, you might want users to tab to the next SingleLineEdit when they press Enter.

Using the KeyDown function
You cannot use the KeyDown PowerScript function to obtain a key code value entered by a user on a Windows CE platform.

Solution Define a user event for the first SingleLineEdit. Give the event any name you want, such as ue_CheckKey. Map the event to the event ID pbm_keydown. Write a script for the user event that tests for the key that was pressed on the Soft Input Panel (SIP) or peripheral keyboard. If Enter was pressed, set the focus to the SingleLineEdit that you want the user to go to.

For example, in the script for the user event for sle_1, you could code:

```
// Script for user event ue_CheckKey,
// which is mapped to pbm_keydown.
IF Key=KeyEnter! THEN   // Go to sle_2 if
   sle_2.SetFocus( )          // Enter pressed.
END IF
```

Example 2: mapped user event for an edit style

Situation You have a DataWindow control with a column that uses the RadioButton edit style and you want to allow users to scroll through the RadioButtons when they press Down Arrow or Up Arrow. (Normally, pressing Down Arrow or Up Arrow scrolls to the next or preceding row.)

Solution Declare a user event for the DataWindow control that maps to the event ID pbm_dwnkey and write a script like the following for it. (dwn stands for DataWindow notification.)

```
// Script is in a user event for a DataWindow control.
// It is mapped to pbm_dwnkey. If user is in column
// number 2, which uses the RadioButton edit style, and
// presses DownArrow, the cursor moves to the next item
// in the RadioButton list instead of going to the next
// row in the DataWindow, which is the default behavior.
// Pressing UpArrow moves to the preceding RadioButton.
//
// Note that the CHOOSE CASE below tests for data
// values, not display values, for the RadioButtons.


int colnum = 2          // Column number
long rownum
rownum = dw_1.GetRow( ) // Current row
IF This.GetColumn( ) = colnum THEN
  CHOOSE CASE key
    CASE KeyDownArrow!
      CHOOSE CASE dw_1.GetItemString(rownum, colnum)
          CASE "a"             // First value in RB
             This.SetItem(rownum, colnum,"b")  // Next
          CASE "b"             // Second value in RB
             This.SetItem(rownum, colnum,"c")  // Next
          CASE "c"             // Last value in RB
             This.SetItem(rownum, colnum,"a")  // First
```

```
        END CHOOSE

    CASE KeyUpArrow!
      CHOOSE CASE dw_1.GetItemString(rownum, colnum)
          CASE "a"             // First value in RB
             This.SetItem(rownum, colnum,"c")  // Last
          CASE "b"             // Second value in RB
             This.SetItem(rownum, colnum,"a")  // First
          CASE "c"             // First value in RB
            This.SetItem(rownum, colnum,"b") // Previous
       END CHOOSE
    END CHOOSE
END IF
return(1)
```

CHAPTER 9

Working with Structures

About this chapter

This chapter describes how to build and use structures.

Contents

About structures

A structure is a collection of one or more related variables of the same or different datatypes grouped under a single name. In some languages, such as Pascal and COBOL, structures are called records.

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

Two kinds

There are two kinds of structures:

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

Deciding which kind you want

When you design your application, think about how the structures you are defining will be used:

- If the structure is general-purpose and applies throughout the application, make it a global structure
- If the structure applies only to a particular type of object, make it an object-level structure

Defining structures

Although you define object-level structures in the painter for a specific object and global structures in the Structure painter, in both cases you define the structure in a Structure view. The following sections describe each of the steps you take to define a new structure:

1 Open a Structure view.

2 For object-level structures, name the structure.

3 Define the variables that make up the structure.

4 Save the structure.

Opening a Structure view

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

Figure 9-1: Structure view in a PocketBuilder object painter

❖ **To define an object-level structure:**

1 Open the object for which you want to declare the structure.

You can declare structures for windows, menus, user objects, or applications.

2 Select Insert>Structure from the menu bar.

A Structure view opens.

❖ **To define a global structure:**

- Select Structure from the Objects tab in the New dialog box.

The Structure painter opens. It has one view, the Structure view. In the Structure painter, there is no Structure Name text box in the Structure view.

Naming the structure

If you are defining an object-level structure, you name it in the Structure Name box in the Structure view. If you are defining a global structure, you name it when you save the structure.

Structure names can have up to 40 characters. For information about valid characters, see "identifier names" in the online Help.

You might want to adopt a naming convention for structures so that you can recognize them easily. A common convention is to preface all global structure names with s_ and all object-level structure names with str_.

Defining the variables

By default, the Structure view displays a single string datatype for the structure's initial variable. You can change this to any PocketBuilder datatype, including the standard datatypes such as integer and boolean, as well as objects and controls such as Window or MultiLineEdit. The default for subsequent variables is the datatype of the previous variable.

Specifying a defined object as a variable You can specify any object types that you have defined. For example, if you have defined a window named w_calculator and you want the structure to include the window, type `w_calculator` as the datatype. You cannot select w_calculator from the list, since the list shows only built-in datatypes.

Specifying a structure as a variable A variable in a structure can itself be a structure. Specify the structure's name as the variable's datatype.

Specifying a decimal as a variable If you select decimal as the datatype, the default number of decimal places is 2. You can also select decimal{2} or decimal{4} to specify 2 or 4 decimal places explicitly.

❖ **To define the variables that compose the structure:**

1 Select or enter the datatype of a variable that you want to include in the structure.

2 Enter the name of the variable.

3 Repeat until you have entered all the variables.

Saving the structure

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

The names of object-level structures are added to the Structure List view and display in the title bar of the Structure view as soon as you tab off the Structure Name box. As you add variables to the structure, the changes are saved automatically. When you save the object that contains the structure, the structure is saved as part of the object in the library in which the object resides.

Comments and object-level structures
You cannot enter comments for an object-level structure, because it is not a PocketBuilder object.

❖ **To name and save a global structure:**

1. Select File>Save from the menu bar, or close the Structure painter.

   The Save Structure dialog box displays.

2. Name the structure.

   See “Naming the structure” on page 191.

3. (Optional) Add comments to describe your structure.
4. Choose the library where you want to save the structure.
5. Click OK.

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

Modifying structures

❖ **To modify a structure:**

1. Do one of the following:
   - In the Open dialog box, select the global structure you want to modify
   - Open the painter for the object that contains the object-level structure, open the Structure List view if it is not already open, and from there select the structure you want to modify

2 Review the variable information displayed in the Structure view and modify the structure as necessary.

   To insert a variable before an existing variable, highlight it and select Insert>Row from the menu bar or Insert Row from the pop-up menu.

   To delete a variable, select Delete Row from the pop-up menu.

3 Save the modified structure.

Building a similar structure

If you want to create a structure that is similar to one that already exists, you can use the existing structure as a starting point and modify it.

❖ **To build an object-level structure that is similar to an existing object-level structure:**

1 Right-click the existing structure in the Structure List view.

2 Select Duplicate from the pop-up menu.

3 Name the new structure in the Structure Name box.

4 Modify variables as needed.

❖ **To build a global structure that is similar to an existing global structure:**

1 Open and modify the existing structure.

2 Select File>Save As from the PocketBuilder menu.

3 Type a name for the new structure, optionally add or modify a comment, select the library where you want to save the structure, and click OK.

Using structures

After you define the structure, you can:

- Reference an instance of the structure in scripts and functions
- Pass the structure to functions
- Display and paste information about structures by using the Browser

Referencing structures

When you define a structure, you are defining a new datatype. You can use this new datatype in scripts and user-defined functions as long as the structure definition is stored in a library in the application's library search path.

❖ **To use a structure in a script or user-defined function:**

1 Declare a variable of the structure type.

2 Reference the variable in the structure.

Referencing global structures

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

structure.variable

Example Assume that s_empdata is a global structure with the variables emp_id, emp_dept, emp_fname, emp_lname, and emp_salary. To use this structure definition, declare a variable of type s_empdata and use dot notation to reference the structure's variables, as shown in the following script:

```
s_empdata   lstr_emp1, lstr_emp2 // Declare 2 variables
                                 // of type emp_data.

lstr_emp1.emp_id = 100       // Assign values to the
lstr_emp1.emp_dept = 200     // structure variables.
lstr_emp1.emp_fname = "John"
lstr_emp1.emp_lname = "Paul-Jones"
lstr_emp1.emp_salary = 99908.23

// Retrieve the value of a structure variable.
lstr_emp2.emp_salary = lstr_emp1.emp_salary * 1.05

// Use a structure variable in a
// PowerScript function.
MessageBox ("New Salary", &
   String(lstr_emp2.emp_salary,"$###,##0.00"))
```

Referencing object-level structures

You reference object-level structures in scripts for the object itself exactly as you do global structures. Declare a variable of the structure type, then use dot notation:

structure.variable

Example Assume that the structure str_custdata is defined for the window w_history and you are writing a script for a CommandButton in the window. To use the structure definition in the script, write:

```
str_custdata lstr_cust1
lstr_cust1.name = "Joe"
```

No access to object-level structures outside the object
You cannot make object-level structures accessible outside an object because object-level structures are implicitly private.

Copying structures

❖ **To copy the values of a structure to another structure of the same type:**

- Assign the structure to be copied to the other structure using this syntax:

  struct1 = struct2

  PocketBuilder copies all the variable values from *struct2* to *struct1*.

Example These statements copy the values in lstr_emp2 to lstr_emp1:

```
str_empdata lstr_emp1, lstr_emp2
...
lstr_emp1 = lstr_emp2
```

Using structures with functions

You can pass structures as arguments in user-defined functions. Simply name the structure as the datatype when defining the argument. Similarly, user-defined functions can return structures. Name the structure as the return type for the function.

You can also define external functions that take structures as arguments.

Example Assume the following:

- Revise is an external function that expects a structure as its argument
- lstr_empdata is a declared variable of a structure datatype

You can call the function as follows:

```
Revise(lstr_empdata)
```

Declare the function first
The external function must be declared before you can reference it in a script.

For more about passing arguments to external functions, see the *Resource Guide*.

Displaying and pasting structure information

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

❖ **To display information about a global structure in the Browser:**

1 Select the Structure tab and select a structure.

2 Double-click the properties folder in the right pane.

The properties folder expands to show the structure variables as properties of the structure.

❖ **To display information about an object-level structure in the Browser:**

1 Select the tab for the type of object for which the structure is defined.

2 Select the object that contains the structure.

3 Double-click the structure folder in the right pane.

The structure folder expands to display the structure variables using dot notation.

❖ **To paste the information into a script:**

1 Scroll to the structure variable you want to paste.

2 Select Copy from the variable's pop-up menu.

3 Insert the cursor in the script where you want to paste the variable and select Paste from the pop-up menu.

The variable name displays at the insertion point in the script.

PART 4

Working with Windows, Controls, and User Objects

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

CHAPTER 10

Working with Windows

About this chapter

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

Contents

About windows

Windows form the interface between the user and a PocketBuilder application. Windows can display information, request information from a user, and respond to the user's stylus or keyboard actions.

A window consists of:

- Properties that define the window's appearance and behavior

  For example, a window might have a caption bar with a dismiss or OK button, and it might have a menu bar.

- Events

  Like other PocketBuilder objects, windows have events.

- Controls placed in the window

At the window level

When you create a window, you specify its properties in the Window painter's Properties view. You can dynamically change window properties in scripts during execution.

You can write scripts for window events to specify what should happen when a window is manipulated. For example, you can connect to a database when a window is opened by coding the appropriate statements in the script for the window's Open event.

At the control level

You place PocketBuilder controls, such as CheckBox, CommandButton, or MultiLineEdit controls, in a window to request and receive information from the user and to present information to the user.

After you place a control in a window, you can define the style of the control, move and resize the control, and code scripts to determine how the control responds to events.

Designing windows

The Microsoft Windows CE operating environment has certain standards that graphical applications are expected to conform to. Windows, menus, and controls are supposed to look and behave in predictable ways from application to application.

This chapter describes some of the guidelines you should follow when designing windows and applications, but a full discussion is beyond the scope of this book. Information about design guidelines for Windows CE platforms is available on the Microsoft Web site at http://msdn2.microsoft.com/en-us/library/ms894178.aspx. See also Appendix D, “Designing Applications for Windows CE Platforms.”

Building windows

When you build a window, you:

- Specify the appearance and behavior of the window by setting its properties
- Add controls to the window
- Compile scripts that determine how to respond to events in the window and its controls

  To support these scripts, you can define new events for the window and its controls, and declare functions, structures, and variables for the window.

Two ways

There are two ways to build a window. You can:

- Build a new window from scratch

  You use this technique to create windows that are not based on existing windows.

- Build a window that inherits its style, events, functions, structures, variables, and scripts from an existing window

  You use inheritance to create windows that are derived from existing windows, thereby saving time and coding effort.

For more information

For information on building a window from scratch, see "Building a new window" on page 204.

For information on using inheritance to build a window, see "Using inheritance to build a window" on page 222.

Types of windows

PocketBuilder provides support for main and response windows. Windows CE platforms do not support MDI windows.

Main windows

Main windows are standalone windows; they are independent of all other windows. They provide the principal user interface for your applications, and can overlay other windows of the same type.

Using main windows

Define your independent windows as main windows. For example, suppose your application contains a calculator or scratch pad window that you want always to have available to the user. Make it a main window, which can be displayed at any time anywhere on the screen. As a main window, it can display on top of other windows on the screen.

Response windows

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

Response windows are application modal. That is, when a response window displays, it is the active window (it has focus), and no other window in the application is accessible until the user responds to the response window. The user can go to other applications, but when the user returns to the application, the response window is still active. Response windows act like modal pop-up windows.

Using response windows

For example, if you want to display a confirmation window when a user tries to close a window with unsaved changes, use a response window. The user is not allowed to proceed until the response window is closed.

Figure 10-1: Example of a response window

Using message boxes

PocketBuilder also provides message boxes, which are predefined windows that act like response windows in that they are application modal. You open message boxes using the PowerScript MessageBox function.

For more information, see "MessageBox" in the online Help.

About the Window painter

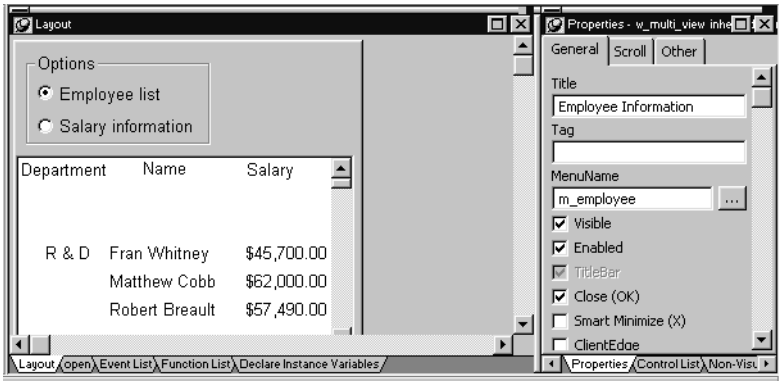
Views in the Window painter

You design windows in the Window painter. The Window painter has several views where you specify how a window looks and how it behaves. The Window painter looks similar to the User Object painter for visual user objects, and it has the same views. For details about the views, how you use them, and how they are related, see "Views in painters that edit objects" on page 58.

Window painter workspace

The default layout for the Window painter workspace has two stacked panes with the Layout and Properties views at the top of the stacks.

Figure 10-2: Default view layout for the Window painter



Most of your work in the Window painter is done in three views:

- The Layout view, where you design the appearance of the window
- The Properties view, where you set window properties and control properties
- The Script view, where you modify behavior by coding window and control scripts

For information about specifying window properties, see “Defining the window's properties” on page 204.

For information about adding controls and nonvisual objects to a window, see “Adding controls” on page 213 and “Adding nonvisual objects” on page 213.

For information about coding in the Script view, see “Writing scripts in windows” on page 217 and Chapter 6, “Writing Scripts.”

Displaying device skins at design time

You can display windows inside a device skin at design time in the Window painter by selecting an XML file containing a device skin definition on the General tab of the Options dialog box. (You open this dialog box from the Window painter’s Design>Options menu.) Clicking the browse button on the General tab displays a file selection dialog box. The file you select in the file selection dialog box is automatically entered in the Current Skin text box on the General tab of the Options dialog box.

The PocketBuilder setup program installs two generic skin files that you can use in the Window painter to frame the windows you create at design time. The *Generic_Phone_Skin.xml* file lets you display windows in a generic Smartphone skin and the *Generic_PPC_Skin.xml* file lets you display windows in a generic Pocket PC skin. The setup program installs these files in the *PocketBuilder Support\IDE_Skins* directory. .

After you select a skin and click the Show Skin button, all windows that you subsequently open in the Window painter are displayed in the selected skin—and the windows are repositioned inside the skin.You can remove the skin by clicking the Hide Skin button.

Building a new window

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

Creating a new window

❖ **To create a new window:**

1 Open the New dialog box.

2 On the PB Object tab page, select Window.

3 Click OK.

The Window painter opens. The new window displays in the Window painter's Layout view, and its default properties display in the Properties view.

Defining the window's properties

Every window (and control) has a style that determines how it appears to the user. You define a window's style by choosing settings in the Window painter's Properties view. A window's style encompasses its:

Type
Basic appearance
Initial position on the screen (response windows)

Size at design time

Runtime size of main windows depends on the device where you deploy your application.

When you define a window's style in the Window painter, you are actually assigning values to the properties for the window. You can programmatically change some elements of a window's style during execution by setting its properties in scripts, although other style selections must be made prior to the loading of the window on the deployment device.

For a list of window properties, see *Objects and Controls* in the online Help. Because of target platform differences, some of the properties listed for PowerBuilder windows do not apply to PocketBuilder windows.

For descriptions of window properties that are specific to PocketBuilder (that is, they do not apply to windows in PowerBuilder applications), see Appendix B, "PowerBuilder and PocketBuilder Product Differences," in this *User's Guide*.

❖ **To specify window properties:**

1 Click the window's background to display the window's properties in the Properties view.

Another way to display window properties
You can also select the window name in the Control List view.

2 Choose the tab appropriate to the property you want to specify:

Choose this tab	To specify the window's
General	Name, type, state, navigation bar button, default design-time size setting, color, and whether a menu is associated with it
Scroll	Horizontal and vertical scroll bar placement
Other	Position and size on the desktop and whether or not to display a tap-and-hold indicator on the Pocket PC

Using the General properties page

Use the General properties page to specify the following window information:

Window type
Title bar text
Menu name

Navigation bar button (OK or X)
IDE window size (design-time window size)
Color

Depending on the type of window, PocketBuilder enables or disables certain check boxes that specify other properties of the window. For example, if you are creating a main window, the Title Bar check box is selected and disabled. Main windows always have title bars, so you cannot clear the Title Bar check box.

By default, the title of a main window is `Untitled` (except if the window is created by the Template Application wizard, in which case the default window title is `Main Window`). You can change the window title by replacing the text in the Title box.

Main windows must have titles
If you delete a title for a main window, you risk creating problems for application users on Windows CE devices. Applications without titles are not included in the list of running programs on these devices. If users start a different program or otherwise minimize a title-less PocketBuilder application, they might have difficulty redisplaying the application, or even closing it, without performing a soft reset.

Specifying the window's type

The first thing you should do on the General properties page is specify the type of window you are creating.

❖ **To specify the window's type:**

1 In the Properties view for the window, select the General tab.

2 Scroll down the property page and select the appropriate window type from the WindowType drop-down list.

Specifying other basic window properties

By selecting and clearing check boxes on the General property page, you can specify whether the window is enabled, has a menu bar, has an OK button in the navigation bar, and so on.

Note the following:

- A main window must have a title bar. It has a default minimum size and cannot be repositioned through the PocketBuilder IDE.
- A response window cannot have a menu. (Response windows should not be used on the Smartphone.)

Associating a menu with the window

Only main windows can have a menu associated with them. If you do not associate a named menu with a main window, you can still select the MenuBar property on the General page of the Properties view for the window. Selecting this property resizes the window at design time, allowing you to see how much space would be left in the window if you added a menu dynamically at runtime.

❖ **To associate a menu with the window:**

1 Do one of the following:

- Enter the name of the menu in the Menu Name text box on the General properties page.
- Click the Browse button and select the menu from the Select Object dialog box, which displays a list of all menus available to the application.

2 Click the Preview button in the PainterBar to see the menu.

For information about preview, see "Viewing your work" on page 215.

Changing the menu
At runtime, you can change a menu associated with a window by using the ChangeMenu function. For more information, see the online Help.

Choosing a window color

You can change the background color of your window.

❖ **To specify the color of a window:**

- Specify the color of the window from the BackColor drop-down list on the General properties page

Changing default window colors

The default window background color for new windows is ButtonFace if you are defining a 3D window, and white if you are not. (If you have specified different display colors for 3D objects in the Windows Control Panel, the default color for a new 3D window will be the color that you set, rather than ButtonFace.)

You can change the default for windows that are not 3D in the Application painter Properties view. To do so, click the Additional Properties button on the General page and modify the Background color on the Text Font tab page. New windows that are not 3D will have the new color you specified.

For more about using colors in windows, including how to define your own custom colors, see Chapter 11, "Working with Controls."

Choosing the window's size and position

Design-time window size

Main windows are automatically configured at runtime to match a deployment device's full screen settings. At design time you can easily change a window's size to match the size it will have on a deployment device. Selections for the IDE window size are:

- PDA Portrait 240 x 320 QVGA
- PDA Landscape 320 x 240 QVGA
- Smartphone Portrait 176 x 220
- Smartphone Square 220 x 220
- VGA Portrait 480 x 640
- VGA Landscape 640 x 480
- Design Size Default
- Unconstrained

The Design Size Default value sets the window displayed in the Window painter to the size selected on the Size tab of the Options dialog box. You display this dialog box by selecting Design>Options from the Window painter menu.

The Unconstrained value lets you set the design-time window size on the Other tab of the Properties view for the current window or by dragging the window handles in the Layout view. If you do not select Unconstrained before you set design-time window size on the Other tab or before you resize the window in the Layout view, PocketBuilder automatically changes the IDE Window Size drop-down list value to Unconstrained.

About PowerBuilder units

All window measurements are in PowerBuilder units (PBUs), except for those on the Size tab of the Options dialog box for the Window painter. Using PBUs, you can build applications that look similar on screens of different resolution. A PBU is defined in terms of logical inches. The size of a logical inch is defined by your operating system as a specific number of pixels. The number is dependent on the display device. Windows typically uses 96 pixels per logical inch for small fonts and 120 pixels per logical inch for large fonts. Windows CE devices typically use 96 pixels per logical inch.

Almost all sizes in the Window painter and in scripts are expressed as PBUs. The two exceptions are text size, which is expressed in points, and grid size in the Window and DataWindow painters, which is in pixels.

The PowerScript functions PixelsToUnits and UnitsToPixels return conversion values for a measurement that you assign as a function argument. For more about these functions, see the online Help.

Display idiosyncracies on VGA devices and emulators

The Windows CE operating system acts as though a VGA screen has the same dimensions as a QVGA screen, using the greater resolution to provide a higher quality picture and font definition rather than to increase the screen surface area. Although the PocketBuilder Window painter lets you set a window surface size for VGA screens in portrait (480 x 640 pixels) and landscape (640 x 480 pixels) orientations, it displays the full physical screen size for the extra resolution, in a manner consistent with the desktop dots-per-inch setting.

This results in certain display issues, with controls and fonts occupying a larger area of the display screen at runtime than at design time. When a DataWindow control is not completely visible on the screen, you might not be able to see a vertical or horizontal scroll bar if the number of rows or columns exceeds the screen display capability.

To work around this problem, you can call the ScreenDisplayZoom function. If you set the zoom factor to 50%, the sizes of controls and fonts on VGA devices and emulators at runtime match their pixel sizes at design time. Alternatively, you can use a QVGA screen size at design time, even when you deploy your applications to a VGA device or emulator. The windows automatically resize to fit the runtime screen dimensions, and the runtime controls retain the sizes that you set for them at design time.

Changing the default sizes of new windows

You can change settings for the default size of a main window. The PocketBuilder install program sets up a portrait orientation for main windows, but you can change this default on the Size tab of the Options dialog box for the Window painter.

The default size of a main window with a portrait orientation is 1097 by 1280 PBUs (240 by 320 pixels) without a menu bar, or 1097 by 1176 PBUs (240 by 294 pixels) if a menu bar is added. (You add a menu bar by selecting the MenuBar check box, or by assigning a menu to the window in the MenuName text box.)

The default size of a response window is 823 by 600 PBUs.

❖ **To change the default size of new windows**

1 Select Design>Options from the main menu for the Window painter.

2 Click the Size tab and select the default size option you want for main windows.

   If you select the Custom radio button for a main window, you must enter values in pixels for the width and height of any new main windows you create in your target.

Resizing a main window in the Layout view

You can resize main and response windows in the Layout view of the Window painter. When you resize a window, the IDE Window Size selection value changes automatically to Unconstrained. At runtime, the size of a main window is modified automatically to fit the full screen size of the device where the window is deployed (unless you are deploying to the desktop).

❖ **To resize a window in the Layout view:**

1 Place the cursor on an outer edge of a window in the Layout view.

   The cursor becomes a two-headed arrow. The edge of the window where you place the cursor determines the angle of the arrow and the window size property that you can change.

2 Drag the edge of the window to a new location.

   The new location determines the current window's new width or height.

Resizing and repositioning windows in the Properties view

You can change a window's width and height properties on the Other page of the Properties view. When you resize a window, the IDE Window Size selection value changes automatically to Unconstrained. At runtime, the size of a main window is modified automatically to fit the full screen size of the device where the window is deployed (unless you are deploying to the desktop).

You can change the x and y values of a response window only. The x and y values for a response window at runtime are relative to the upper-left corner of its main window parent (the bottom left edge of the Windows CE navigation bar).

❖ **To specify a window's position and size:**

1 Click the window's background so that the Properties view displays window properties.

2 Select the Other tab in the current window's Properties view.

3 Enter values for the x and y location of the window in PBUs.

   You can enter values only for a response window.

4 Enter values for width and height in PBUs.

   The size of the window changes in the Layout view.

Multiple Orientation Painter for windows

About the Multiple Orientation Painter

From the Window painter, you can select the View>Runtime MOP Views>MOPView Manager menu item to create multiple views for each PocketBuilder window.

The MOPView Manager can create only views that are based on the types of devices supported by PocketBuilder and the screen orientations available for these devices. When device users switch screen orientations (WM 2003 or higher), the corresponding view for the current orientation displays automatically.

When screen orientations do not have a corresponding view
If you do not create a specific view for the selected orientation, the last view saved in the Window painter is displayed at runtime. In this case, the objects in the window do not necessarily display in optimal sizes and positions for the user-selected orientation.

Available view types

Table 10-1 indicates the screen sizes of the six available view types in the MOPView Manager:

Table 10-1: Available view types in the MOPView Manager

View type	Screen size in pixels
PDA Portrait	240 x 320 (QVGA)
PDA Landscape	320 x 240 (QVGA)
Smartphone Portrait	176 x 220
Smartphone Square	220 x 220
VGA Portrait	480 x 640
VGA Landscape	640 x 480

Each view type that you add in the MOPView Manager becomes a specific view defined for the current window in the Window painter. To add a view, you select a view type in the rightmost list box of the manager, then click Add. You can delete a specific view by selecting it in the leftmost list box, and then clicking Delete. You click OK to save your view selections. The last view you select before clicking OK becomes the current view.

Changing views at runtime and design time

Views that you add in the MOPView Manager are automatically listed as menu items in the View>Runtime MOP Views menu. At design time you can toggle between different views or between a view and no view at all by selecting these menu items. The current view is listed in the Runtime MOP Views submenu with a check mark beside it.

You can also change views by selecting an item from the IDE Window Size drop-down list on the General page of a window's Properties view. Selecting "Unconstrained" or changing the size of the window in the Layout view or on the Other page of the Properties view can take you out of all specific views defined for the current window. Response windows are the only window types that do not need to conform to a specific screen size on the handheld device.

Property specificity

For a window with multiple views, the position (x and y) and dimension (width and height) properties of the window and its controls are view-specific. All other window and control properties are window-specific—that is, they apply to all of the defined window views.

The source code in the PKL file includes only the last position and dimension properties saved in the Window painter for the window and its controls. Therefore, when you export a window with MOP views, the MOP views are not included in the exported code. The exported code contains only the most recent position and dimension properties from the original window.

However, when you copy or inherit from a window, the copied or inherited window retains all the MOP views set in the original window. You can also move a window from one PKL to another without losing its MOP views.

Specifying window scrolling

If your window is larger than the default size, it is probable that not all the window's contents will be visible during execution. In such cases, you should make the window scrollable by providing vertical and horizontal scroll bars. You do this on the Scroll property page.

By default, PocketBuilder controls scrolling when scroll bars are present. You can set values for the size of the scrolling action when a user clicks on a scroll bar arrow.

Table 10-2: Options that control the size of a scrolling action

Option	Meaning
UnitsPerLine	The number of PBUs to scroll up or down when the user clicks the up or down arrow in the vertical scroll bar. When the value is 0 (the default), a scrolling action equals 1/100 the height of the window. The value you enter defines a logical line in the context of the current window.
UnitsPerColumn	The number of PBUs to scroll right or left when the user clicks the right or left arrow in the horizontal scroll bar. When the value is 0 (the default), a scrolling action equals 1/100 the width of the window. The value you enter defines a logical column in the context of the current window.

Option	Meaning
ColumnsPerPage	The number of columns to scroll when the user clicks the horizontal scroll bar itself. When the value is 0 (the default), a scrolling action equals 10 columns.
LinesPerPage	The number of lines to scroll when the user clicks the vertical scroll bar itself. When the value is 0 (the default), a scrolling action equals 10 lines.

❖ **To specify window scrolling:**

1 Click the window's background so that the Properties view displays window properties.

2 Select the Scroll tab.

3 Indicate which scroll bars you want to display by selecting the HScrollBar and VScrollBar check boxes.

4 For horizontal scroll bars, specify scrolling options for the units per column and the columns per page. For vertical scroll bars, specify scrolling options for the units per line and the lines per page.

Adding controls

When you build a window, you place controls, such as CheckBox, CommandButton, and MultiLineEdit controls, in the window to request and receive information from the user and to present information to the user.

After you place a control in the window, you can define its style, move and resize it, and write scripts to determine how the control responds to events.

For more information, see Chapter 11, “Working with Controls.”

Adding nonvisual objects

If you need the services of a nonvisual object, you can insert it in a window. Nonvisual objects are listed in the Non-Visual Object List view for the window. A nonvisual object that you insert in a window can be a custom class or standard class user object.

You insert a nonvisual object in a window in the same way you insert one in a user object. For more information, see “Using class user objects” on page 325.

Saving the window

You can save the window you are working on at any time.

Naming the window

The name you assign to a window when you save it can be any valid identifier of up to 40 characters. For information, see "valid identifiers" in the online Help.

A commonly used convention is to preface all window names with w_ and use a suffix that helps you identify the particular window. For example, you might name a window that displays employee data w_empdata.

❖ **To save a window:**

1 Select File>Save from the menu bar.

If you have previously saved the window, PocketBuilder saves the new version in the same library and returns you to the Window painter workspace.

If you have not previously saved the window, PocketBuilder displays the Save Window dialog box.

All orientation views of a window are saved at the same time. For more information about multiple orientation views, see "Multiple Orientation Painter for windows" on page 211.

2 Name the window in the Windows text box.

See the window naming considerations at the beginning of this section.

3 Type comments in the Comments text box to describe the window.

These comments display in the Select Window window and in the Library painter. It is a good idea to use comments so that you and others can easily remember the purpose of the window later.

4 Specify which library you want to save the window in.

5 Click OK.

Viewing your work

While building a window, you can preview it and print its definition.

Previewing a window

As you develop a window, you can preview its appearance from the Window painter. By previewing the window, you get a quick desktop view of how it might look during execution, only without the personal digital assistant (PDA) skin. If the window that you preview has an associated menu, the menu bar will be visible at the top of the window, not at the bottom as on a PDA device or emulator. Size restrictions for windows on the PDA device or emulator are also not applicable to windows that you preview on the desktop.

The control bar is present for main windows that you preview or run on the desktop. You can close a main window that you preview by clicking the close button in its control bar.

You can add a control bar to response windows by selecting the TitleBar check box or adding a Close (OK) button. If you preview a response window that does not have a control bar, you can close the response window by right-clicking the PocketBuilder icon in the Windows task bar and selecting Restore from the pop-up menu. A message box informs you that an application is running. You can then click Terminate in the message box to close the response window and return to PocketBuilder.

Preview button on the PainterBar and the PowerBar
You can preview a window from the Window painter by using the Preview button on the PainterBar or clicking the Preview button on the PowerBar. When you use the Preview button on the PainterBar, you do not have to save the window first, but you cannot trigger events, as described later. For information about previewing using the PowerBar button, see “Running a window” on page 221.

❖ **To preview a window:**

- Click the Preview button in the PainterBar, or select Design>Preview from the menu bar

  PocketBuilder is minimized, and the window displays with the properties you have defined for it.

For information about previewing using the PowerBar button, see "Running a window" on page 221.

What you can do

While previewing the window, you can get a sense of its look and feel. You can:

- Move the window (although you cannot do this on the Windows CE device)
- Tab from control to control
- Select controls
- Use scroll bars that you place on the window

What you cannot do

You cannot:

- Change properties of the window

  Changes you make while previewing the window, such as changing radio button selections or entering text in a text box, are not saved.

- Trigger events

  For example, clicking a CommandButton while previewing a window does not trigger the button's Clicked event.

- Connect to a database

❖ **To return to the Window painter:**

- Do one of the following:
  - Click the Close button in the control menu in the upper-right corner of the window

    On the Windows CE device or emulator, you will not have a control menu in the title bar, although for the Pocket PC platform you can add a Close (OK) button or a SmartMinimize (X) button to the navigation bar. (Smartphone applications do not have Close or SmartMinimize buttons, but typically have a Quit or Done menu assigned to the left soft key.)

    You can add a Close button to the navigation bar or title bar for a response window, but not a SmartMinimize button. You cannot preview the navigation bar on the desktop using the Preview feature.

  - Right-click the title bar and select Close from the pop-up menu

- For a main window that is not visible, click PocketBuilder on the task bar and then click the Terminate button
- For a response window that does not have a control bar, right-click PocketBuilder on the task bar, select Restore from the pop-up menu, and then click the Terminate button

Printing a window's definition

You can print a window's definition for documentation purposes.

❖ **To print information about the current window:**

- Select File>Print from the menu bar

  Information about the current window is sent to the printer specified in Printer Setup. You can change the information sent to the printer by right-clicking the window in the System Tree or Library painter, selecting Print from the pop-up menu, then selecting or clearing check boxes for the different print options for the window or user object.

Print settings
You can also view and change the print options from the Library painter by selecting any PocketBuilder object and then selecting Entry>Library Item>Print from the menu bar.

PocketBuilder records your printing options preferences in variables in the [Library] section of the PocketBuilder initialization file, so the preference is maintained across sessions. If there is no [Library] section, PocketBuilder creates one after you print an object definition for the first time.

Writing scripts in windows

You write scripts for window and control events. To support these scripts, you can define:

- Window-level and control-level functions
- Instance variables for the window

About events for windows and controls

Windows have several events, including Open, which is triggered when the window is opened (before it is displayed), and Close, which is triggered when the window is closed. For example, you might connect to a database and initialize some values in the window's Open event, and disconnect from a database in the Close event.

Each type of control also has its own set of events. A button, for example, has a Clicked event, which triggers when a user clicks the button. SingleLineEdits and MultiLineEdits have Modified events, which trigger when the contents of the edit control change.

Defining your own events

You can also define your own events, called user events, for a window or control, then use the EVENT keyword to trigger your user event.

For example, suppose that you offer the user several ways to update the database from a window, such as clicking a button or selecting a menu item. When the user closes the window, you want to update the database as well (after asking for confirmation). You want the same type of processing to occur after different system events.

You can define a user event for the window, write a script for that event, and then everywhere you want that event triggered, use the EVENT keyword.

To learn how to use user events, see Chapter 8, “Working with User Events.”

About functions for windows and controls

PocketBuilder provides built-in functions that act on windows and on different types of controls. You can use these functions in scripts to manipulate your windows and controls. For example, to open a window, you can use the built-in window-level function Open, or you can pass parameters between windows by opening them with the function OpenWithParm and closing them with CloseWithReturn.

You can define your own window-level functions to make it easier to manipulate your windows. For more information, see Chapter 7, “Working with User-Defined Functions.”

About properties of windows and controls

In scripts, you can assign values to the properties of objects and controls to change their appearance or behavior. You can also test the values of properties to obtain information about an object.

For example, you can change the text displayed in a StaticText control when the user clicks a CommandButton, or use data entered in a SingleLineEdit to determine the information that is retrieved and displayed in a DataWindow control.

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

object.property

control.property

If you do not identify the object or control when you refer to a property, PocketBuilder assumes you are referring to the object or control the script is written for.

The reserved word Parent

In the script for a window control, you can use the reserved word Parent to refer to the window containing the control. For example, the following line in a script for a CommandButton closes the window containing the button:

```
close(Parent)
```

It is easier to reuse a script if you use Parent instead of the name of the window.

Properties, events, and built-in functions for all PowerBuilder objects, including windows, and all types of controls are described in *Objects and Controls*. In most cases, descriptions apply to properties, events, and built-in functions for PocketBuilder objects. Differences between PowerBuilder and PocketBuilder are described in Appendix B, "PowerBuilder and PocketBuilder Product Differences," of this *User's Guide*.

Declaring instance variables

Often data needs to be accessible to several scripts within a window. For example, suppose that a window displays information about one customer. You might want several CommandButtons to manipulate the data, and the script for each button needs to know the customer's ID. There are several ways to accomplish this:

- Declare a global variable containing the current customer ID

  All scripts in the application have access to this variable.

- Declare an instance variable within the window

  All scripts for the window and controls in the window have access to this variable.

- Declare a shared variable within the window

  All scripts for the window and its controls have access to this variable. In addition, all other windows of the same type have access to the same variable.

When declaring a variable, you need to consider what the scope of the variable is. If the variable is meaningful only within a window, declare it as a window-level variable, generally an instance variable. If the variable is meaningful throughout the entire application, make it a global variable.

For a complete description of the types of variables and how to declare them, see the *PowerScript Reference* in the online Help.

Examples of statements

The following assignment statement in the script for the Clicked event for a CommandButton changes the text in the StaticText object st_greeting when the button is clicked:

```
st_greeting.Text = "Hello User"
```

The following statement tests the value entered in the SingleLineEdit sle_state and displays the window w_state1 if the text is "AL":

```
if sle_State.Text= "AL" then Open(w_state1)
```

Running a window

During development, you can test a window on the desktop without running the whole application or deploying it to an emulation environment.

You can run a window using the PowerBar Run/Preview Object button, even if the window is not in the current target or library. Because the window is functional, if the window is currently open in the Window painter, you are prompted to save any changes to the window. You can trigger events and open other windows while running the window in this manner.

Previewing the window
You can preview a window from the Window painter by using the Preview button on the PainterBar or run the window by clicking the Preview button on the PowerBar. For information about previewing using the PainterBar button, see "Previewing a window" on page 215.

❖ **To run a window:**

1 Click the Preview button in the PowerBar (not the PainterBar).

2 In the Run/Preview dialog box, select Windows as the Objects of Type.

3 Select the target that includes the window you want to run.

4 Select the library that includes the window.

5 Select the window you want to run and click OK.

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

PocketBuilder runs the window.

When running a window, you can trigger events, open other windows, connect to a database, and so on. The window is fully functional. It has access to global variables you have defined for the application and to built-in global variables, such as SQLCA. The SystemError event is not triggered if there is an error, because SystemError is an Application object event.

❖ **To return to the Window painter:**

- Do one of the following:
  - Click the Close button in the upper-right corner of the window
  - If the window is not visible, click PocketBuilder on the task bar and then click the Terminate button

Using inheritance to build a window

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

This section provides an overview of using inheritance in the Window painter. The issues concerning inheritance with windows are the same as the issues concerning inheritance with user objects and menus. They are described in more detail in Chapter 12, "Understanding Inheritance."

Building two windows with similar definitions

Suppose that your application needs two windows with similar definitions. One window, w_employee, needs:

- A title (Employee Data)
- Text that prompts a user to select a file
- A drop-down list with a list of available employee files
- An Open button with a script that opens the selected file in a multiline edit box
- An Exit button with a script that asks the user to confirm that the window should be closed and then closes the window

Figure 10-3 shows how the w_employee window might look.

Figure 10-3: Example window w_employee

The only differences in the second window, w_customer, are that the title is Customer Data, the drop-down list displays customer files instead of employee files, and there is a Delete button so that the user can delete files.

Your choices

To build these windows, you have three choices:

- Build two new windows from scratch as described in "Building a new window" on page 204
- Build one window from scratch and then modify it and save it under another name
- Use inheritance to build two windows that inherit a definition from an ancestor window

Using inheritance

To build the two windows using inheritance, follow these steps:

1 Create an ancestor window, w_ancestor, that contains the text, drop-down list, and the open and exit buttons, and save and close it.

Using the Window painter with inherited windows
You cannot inherit a window from an existing window when the existing window is open, and you cannot open a window when its ancestor or descendant is open.

2 Select File>Inherit, select w_ancestor in the Inherit From dialog box, and click OK.

3 Add the Employee Data title, specify that the DropDownListBox control display employee files, and save the window as w_employee.

4 Select File>Inherit, select w_ancestor in the Inherit From dialog box, and click OK.

5 Add the Customer Data title, specify that the DropDownListBox control display customer files, add the Delete button, and save the window as w_customer.

Advantages of using inheritance

Using inheritance has a number of advantages:

- When you change the ancestor window, the changes are reflected in all descendants of the window. You do not have to make manual changes in the descendants, as you would in a copy. This saves you coding time and makes the application easier to maintain.

- Each descendant inherits the ancestor's scripts, so you do not have to re-enter the code to add to the script.
- You get consistency in the code and in the application windows.

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

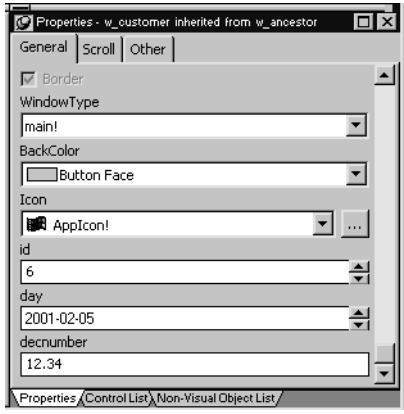
- Change the properties of the window
- Add controls to the window and modify existing controls
- Size and position the window and the controls in the window
- Build new scripts for events in the window or its controls
- Reference the ancestor's functions and events
- Reference the ancestor’s structures if the ancestor contains a public or protected instance variable of the structure datatype
- Access ancestor properties, such as instance variables, if the scope of the property is public or protected
- Extend or override inherited scripts
- Declare functions, structures, and variables for the window
- Declare user events for the window and its controls

The only thing you cannot do is delete inherited controls. If you do not need an inherited control, you can make it invisible in the descendent window.

Instance variables in descendants

If you create a window by inheriting it from an existing window that has public or protected instance variables with simple datatypes, the instance variables display and can be modified in the descendent window’s Properties view. You see them at the bottom of the General tab page.

Figure 10-4: Instance variables inherited from an ancestor window



All public instance variables with simple datatypes such as integer, boolean, character, date, string, and so on display. Instance variables with the any or blob datatype or instance variables that are objects or arrays do not display.

Control names in descendants

PocketBuilder uses this syntax to show names of inherited controls:

ancestorwindow::*control*

For example, if you select the Open button in w_customer, which is inherited from w_ancestor, its name displays on the General page in the properties view as `w_ancestor::cb_open`.

Names of controls must be unique in an inheritance hierarchy. For example, you cannot have a CommandButton named cb_close defined in an ancestor and a different CommandButton named cb_close defined in a child. You should develop a naming convention for controls in windows that you plan to use as ancestors.

CHAPTER 11

Working with Controls

About this chapter

Users run your application primarily by interacting with the controls you place in windows. This chapter describes the use of controls.

Contents

About controls

About window controls

You place controls in a window to request and receive information from the user and to present information to the user. For a complete list of standard window controls, open a window in the Window painter and select Insert>Control.

If you often use a control or set of controls with certain properties, such as a group of related radio buttons, you can create a visual user object that contains the control or set of controls. For more about user objects, see Chapter 14, "Working with User Objects." For information about objects and controls that are native to the Pocket PC and Smartphone platforms, see Chapter 15, "Working with Native Objects and Controls for Windows CE Devices."

About events

All window controls have events so that users can act on the controls. You write scripts that determine the processing that takes place when an event occurs in the control.

Drawing objects are usually used only to make your window more attractive or to group controls. Only constructor and destructor events are defined for them, but you can define your own events if needed. The drawing objects are Line, Oval, Rectangle, and RoundRectangle.

Inserting controls in a window

You insert controls in a window in the Window painter.

❖ **To insert a control in a window:**

1. Select Insert>Control from the menu bar, or display the Controls drop-down toolbar on the PainterBar.
2. Select the control you want to insert.

   If you selected User Object, the Select Object dialog box displays, listing all user objects defined for the application. In this case, select the library and the user object you want to insert, and click OK.
3. In the Layout view, click where you want the control.

After you insert the control, you can size it, move it, define its appearance and behavior, and create scripts for its events.

Duplicating controls

To place multiple controls of the same type in a window, place a control in the window and make sure it is selected. Then press Ctrl+T or select Duplicate from the pop-up menu once for each duplicate control you want to place in the window. The controls are placed one under another. You can drag them to other locations as needed.

Inserting controls with undefined content

When you insert a DataWindow, Picture, PictureButton, or PictureHyperLink control in a window, you are inserting only the control. You see only an empty box for a DataWindow control, the dotted outline of a box for a Picture or a PictureHyperLink control, and a large button resembling a CommandButton for a PictureButton control. You must specify a DataWindow object or picture later.

Selecting controls

You select controls so that you can change their properties or write scripts using the Layout view or the Control List view.

❖ **To select a control:**

- Click the control in the Layout view, or click the control in the Control List view

  In the Layout view, the control you clicked displays with handles on it. Any previously selected controls are no longer selected.

Acting on multiple controls

You can act on all or multiple selected controls as a unit. For example, you can move all of the controls or change the fonts for all the text displayed in the controls.

❖ **To select multiple controls:**

- In the Layout or Control List view, click the first control, press and hold the Ctrl key, then click additional controls

❖ **To select neighboring multiple controls:**

- In the Layout view, press the left mouse button, drag the mouse over the controls you want to select, and release the mouse button

---

Selecting all controls
You can select all controls by selecting Edit>Select All from the menu bar.

Information displayed in the MicroHelp bar

The name, x and y coordinates, width, and height of a selected control are displayed in the MicroHelp bar. If you select multiple objects, `Group Selected` displays in the Name area and the coordinates and size do not display.

Defining a control's properties

Just like the window object, each control has properties that determine the control's style—how it looks and behaves during execution.

You define a control's properties by using the Properties view for the control. The properties and values displayed in the Properties view change dynamically when you change the selected object or control. To see this, click the window background to display the window properties in the Properties view and then click a control in the window to display the control's properties in the Properties view.

❖ **To define a control's properties:**

1 Select the control.

   The selected control's properties display in the Properties view.

2 Use the tab pages in the Properties view to change the control's properties.

About tab pages in the Properties view

The Properties view presents information in a consistent arrangement of tabbed property pages. You select items on the individual property pages to change the control's definition.

All controls have a General properties page, which contains much of the style information—such as the visibility of the control, whether it is enabled, and so on—about the control. The General properties page is always the first page of a control's Properties view.

Getting Help on properties

You can get Help when you are defining properties. In any tab page in the Properties view, right-click on the background and select Help from the pop-up menu. The Help displays information about the control and a link to an alphabetical list of properties for the control.

Naming controls

When you place a control in a window, PocketBuilder assigns it a unique name. The name is the concatenation of the default prefix for the control name and the lowest 1- to 4-digit number that makes the name unique.

For example, assume the prefix for ListBoxes is lb_ and you add a ListBox to the window:

- If the names lb_1, lb_2, and lb_3 are currently used, the default name is lb_4
- If lb_1 and lb_3 are currently used but lb_2 is not, the default name is lb_2

About the default prefixes

Each type of control has a default prefix for its name. Table 11-1 lists the initial default prefix for each control in a window.

Table 11-1: Default prefixes for window control names

Control	Prefix
CheckBox	cbx_
CommandButton	cb_
DataWindow	dw_
DropDownListBox	ddlb_
EditMask	em_
Graph	gr_
GroupBox	gb_
HProgressBar	hpb_
HScrollBar	hsb_
HTrackBar	htb_
Line	ln_
ListBox	lb_
ListView	lv_
MultiLineEdit	mle_
NotificationBubble	nb_
Oval	ov_
Picture	p_
PictureHyperLink	phl_
PictureButton	pb_
PocketOutlookObjectManager	po_
RadioButton	rb_
Rectangle	r_
RoundRectangle	rr_
Signature	sig_
SingleLineEdit	sle_

Control	Prefix
StaticText	st_
StaticHyperLink	shl_
Tab	tab_
Toolbar	tlbr_
TreeView	tv_
User Object	uo_
VProgressBar	vpb_
VScrollBar	vsb_
VTrackBar	vtb_

Changing the default prefixes

You can change the default prefixes for controls in the Window painter's Options dialog box. Select Design>Options from the menu bar to open the Options dialog box. The changes you make are saved in the PocketBuilder initialization file. For more about the initialization file, see "About the initialization file" on page 51.

Changing the name

You should change the default suffix to a suffix that is meaningful in your application. For example, if you have command buttons that update and retrieve database information, you might call them cb_update and cb_retrieve. If you have many controls on a window, using intuitive names makes it easier for you and others to write and understand scripts for these controls.

Using application-based names instead of sequential numbers also minimizes the likelihood of name conflicts when you use inheritance to create windows.

❖ **To change a control's name:**

1 Select the control in the Layout view or in the Control List view; this displays the control's properties in the Properties view.

2 On the General properties page, select the application-specific suffix (for example, the `1` in the `cb_1` command button name) and type a more meaningful one.

You can use any valid identifier of up to 40 characters. For information, see "identifier names" in the online Help.

Changing text

You can specify the text and text display characteristics for a control in the Properties view for the control. You can also use the Window painter StyleBar to change:

- The text itself
- The font, point size, and characteristics such as bold
- The alignment of text within the control

CommandButton text
Text in CommandButtons is always center aligned.

The default text for most controls that have a text property is `none`. To display an empty StaticText or SingleLineEdit control, clear the Text box in the Properties view or the StyleBar.

When you add text to a control's text property, the width of the control changes automatically to accommodate the text as you type it in the StyleBar, or when you tab off the Text box in the Properties view.

❖ **To change text properties of controls:**

1 Select one or more controls whose properties you want to change.

2 Specify changes in the Font tab page in the Properties view, or specify changes using the StyleBar.

How text size is stored

A control's text size is specified in the control's TextSize property. The values in the TextSize drop-down list determine the point size of the control's text. PocketBuilder uses negative numbers when saving the text size in points. For example, if you define the text size for the StaticText control st_prompt to be 12 points, PocketBuilder sets the value of the st_prompt TextSize property to –12.

If you want to change the text size in points at runtime, you must use a negative value. For example, to change the point size for st_prompt to 14 points, code:

```
st_prompt.TextSize = -14
```

If you want to specify text size in pixels, you can do so by using positive numbers. The following statement sets the text size to be 14 pixels:

```
st_prompt.TextSize = 14
```

Moving and resizing controls

There are several ways to move and resize controls in the Layout view.

Moving and resizing controls using the mouse

To move a control using the mouse, drag the control to where you want it.

To resize a control, select it, then grab an edge and drag the edge with the mouse.

Moving and resizing controls using the keyboard

To move a control using the keyboard, select the control, then press an arrow key to move it in the corresponding direction.

To resize a control, select the control, then press:

- Shift+Right Arrow to make the control wider
- Shift+Left Arrow to make the control narrower
- Shift+Down Arrow to make the control taller
- Shift+Up Arrow to make the control shorter

Aligning controls using the grid

The Window painter provides a grid to help you align controls at design time. You can use the grid options to:

- Make controls snap to a grid position when you place them or move them in a window

- Show or hide the grid when the workspace displays
- Specify the height and width of the grid cells

❖ **To use the grid:**

1 Choose Design>Options from the menu bar and select the General tab.

2 Do one or more of the following:

- Select Snap to Grid to align controls with a horizontal and vertical grid when you place or move them
- Select Show Grid to display the grid in the Layout view
- Specify the width of each cell in the grid in pixels in the X text box
- Specify the height of each cell in the grid in pixels in the Y text box

Hiding the grid
Window painting is slower when the grid is displayed, so you might want to display the grid only when necessary.

Aligning controls with each other

You can align selected controls by their left, right, top, or bottom edges or their horizontal or vertical centers.

PainterBars in the Window painter
The Window painter has three PainterBars. PainterBar1 includes buttons that perform operations that are common to many painters, including save, cut, copy, paste, and close. PainterBar2 includes buttons used with the Script view. PainterBar3 contains buttons that manipulate the display of the selected control or controls. The tools used to align, resize, and adjust the space between controls are on a drop-down toolbar on PainterBar3.

❖ **To align controls:**

1 Select the control whose position you want to use to align the others.

PocketBuilder displays handles around the selected control.

2 Press and hold the Ctrl key and click the controls you want to align with the first one.

All the selected controls have handles on them.

3 Select Format>Align from the menu bar, or select the Layout drop-down toolbar in PainterBar3.

4 Select the dimension along which you want to align the controls.

PocketBuilder aligns all the selected controls with the first control selected.

Equalizing the space between controls

You can manually move controls by dragging them with the mouse. You can also equalize the space around selected controls by using the Format menu or the Layout drop-down toolbar.

❖ **To equalize the space between controls:**

1 Select the two controls whose spacing is correct.

To do so, select one control, then press and hold Ctrl and click the second control.

2 Select the other controls whose spacing you want to have match the first two controls.

To do so, press and hold Ctrl while clicking each control whose spacing you want to change.

3 Select Format>Space from the menu bar, or select the Layout drop-down toolbar in PainterBar3.

4 Select horizontal or vertical spacing.

The PowerTip text for the corresponding toolbar items is:

- `Space evenly horizontally`
- `Space evenly vertically`

Equalizing the size of controls

Using the Format menu or the Layout drop-down toolbar, you can adjust the selected controls so that they are all the same size as the first control selected. This is something you might do if you have several SingleLineEdit or CommandButton controls on a window.

❖ **To equalize the size of controls:**

1 Select the control whose size is correct.

2 Select the other controls whose size you want to have match the first control.

To do so, press and hold Ctrl while clicking each control whose size you want to change.

3 Select Format>Size from the menu bar, or select the Layout drop-down toolbar in PainterBar3.

4 Select the menu or toolbar item for width, height, or both width and height.

The PowerTip text for the toolbar items is:

- `Make all widths same as first selected`
- `Make all heights same as first selected`
- `Make all widths and heights same as first selected`

Copying controls

You can copy a control within a window or to other windows. All properties of the control, as well as all of its scripts, are copied. You can use this technique to easily make a copy of an existing control and change whatever you want in the copy.

❖ **To copy a control:**

1 Select the control.

2 Select Edit>Copy from the menu bar or press Ctrl+C.

The control is copied to a private PocketBuilder clipboard.

3 Do one of the following:

- To copy the control within the same window, select Edit>Paste Controls from the menu bar or press Ctrl+V.
- To copy the control to another window, click the Open button in the PowerBar and open the window in another instance of the Window painter. Make that window active and select Edit>Paste Controls from the menu bar or press Ctrl+V.

If the control you are pasting has the same name as a control that already exists in the window, the Paste Control Name Conflict dialog box displays.

4 If prompted, change the name of the pasted control to be unique.

PocketBuilder pastes the control in the destination window at the same location as in the source window. (If you are pasting into the same window, you should move the pasted control so that it does not overlay the original control.) You can make whatever changes you want to the copy; the original control remains unaffected.

Defining the tab order

When you place controls in a window, PocketBuilder assigns them a default tab order, the default sequence in which focus moves from control to control when the user presses the Tab key. Although it is not standard practice to use the Tab key for changing focus in Windows CE applications, you have this capability with applications that you develop for Pocket PC devices in PocketBuilder.

Tab order in user objects
When the user tabs to a custom user object in a window and then presses the Tab key (or up/down arrow keys in a Smartphone application), focus moves to the next control in the tab order for the user object. After the user tabs to all the controls in the tab order for the user object, focus moves to the next control in the window tab order.

Establishing the default tab order

PocketBuilder uses the relative positions of controls in a window to establish the default tab order. It looks at the positions in the following order:

- The distance of the control from the top of the window (Y)
- The distance of the control from the left edge of the window (X)

The control with the smallest Y distance is the first control in the default tab order. If multiple controls have the same Y distance, PocketBuilder uses the X distance to determine the tab order among these controls.

Default tab values for drawing objects and radio buttons
The default tab value for drawing objects and RadioButtons in a GroupBox is 0, which means the control is skipped when the user tabs from control to control.

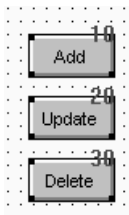
When you add a control to the window, PocketBuilder obtains the tab value of the control that precedes the new control in the tab order and assigns the new control the next number. The values assigned are multiples of 10.

For example, if the tab values for controls A, B, and C are 30, 10, and 20 respectively, and you add control D between controls A and B, PocketBuilder assigns control D the tab value 40.

Changing the window's tab order

You can view and modify the tab order of a window's controls through the tab order formatting mode of PocketBuilder. (You can also change the tab order of an individual control on the Other tab of the control's Properties view.)

Figure 11-1: Viewing the tab order of controls



❖ **To change the tab order:**

1 Select Format>Tab Order from the menu bar, or click the Tab Order button on PainterBar1 (next to the Preview button).

The current tab order displays. If this is the first time you have used Tab Order for the window, the default tab order displays.

2 Use the mouse or the Tab key to move the pointer to the tab value you want to change.

3 Enter a new tab value from 0 to 9999.

The value 0 removes the control from the tab order. Other than 0, the exact value you use does not matter. Only the relative value is significant. For example, if you want the user to tab to control B after control A but before control C, set the tab value for control B so it is between the value for control A and the value for control C.

Tab tips

A tab order value of 0 does not prevent a control from being selected or activated, or from receiving keyboard events. To prevent a user from activating a control, clear the Enabled check box on its General properties page.

To permit tabbing in a group box, change the tab value of the GroupBox control to 0, then assign nonzero tab values to the controls in the group box.

4 Repeat the procedure until you have the tab order you want.

5 Select Format>Tab Order or the Tab Order button again.

PocketBuilder saves the tab order.

Each time you select Tab Order, PocketBuilder renumbers the tab order values to include any controls that have been added to the window and to allow space to insert new controls in the tab order. For example, if the original tab values for controls A, B, and C were 10, 20, and 30, and you insert control D between A and B and give it a tab value of 15, then when you select tab order again, the controls A, B, and C will have the tab values 10, 30, and 40, and control D will have the tab value 20.

Defining accelerator keys

You can define an accelerator key for a control to allow users to change focus to the control. An accelerator key is sometimes referred to as a mnemonic access key.

Although it is not standard practice to include accelerator keys in Windows CE applications, you have this capability with applications that you develop in PocketBuilder.

Using accelerator keys with noneditable controls

For noneditable controls, such as a command button or check box, users need only press the accelerator key to change focus to the control and precipitate control events.

❖ **To define an accelerator key for a CommandButton, CheckBox, or RadioButton:**

1 Click the control to display the control's properties in the Properties view.

2 In the Text box on the General page, precede the letter that you want to use as the accelerator key with an ampersand character (&).

When you perform your next action (such as tabbing to the next property, saving the window, or selecting another control in the Layout view), the property is set and PocketBuilder displays an underline to indicate the accelerator key.

Displaying an ampersand
If you want to display an ampersand character in the text of a control, type a double ampersand. The first ampersand acts as an escape character.

Using accelerator keys with editable controls

You can add accelerator keys to editable controls, but for Pocket PC applications, these are useful only with peripheral keyboards that have an Alt key. Users must press Alt followed by the accelerator key to use an accelerator, and the Alt key is not available on the Soft Input Panel (SIP), or even on some peripheral keyboards, for Pocket PC devices.

❖ **To define an accelerator key for a SingleLineEdit, MultiLineEdit, ListBox, or DropDownListBox:**

1 Click the control to display the control's properties in the Properties view.

2 In the General properties page, type the letter of the accelerator key in the Accelerator box.

For example, if the control contains a user's name and you want to make Alt+N the accelerator for the control, type `n` in the Accelerator box.

At this point you have defined the accelerator key, but the user has no way of knowing it, so you need to label the control.

3 Place a StaticText control next to the control that was assigned the accelerator key.

4 Click the StaticText control to display its properties in the Properties view.

5 In the Text box on the General page, precede the letter that you want to use as the accelerator key with an ampersand character (&).

For example, if the StaticText control will display the label Name, type `&Name` in the Text box so that the letter N is underlined. The underlined letter in the StaticText label lets the user know that there is an accelerator key for changing focus to the associated editable control.

Specifying accessibility of controls

Controls have two boolean properties that affect their accessibility:

- Visible
- Enabled

Using the Visible property

If the Visible property of a control is selected, the control displays in the window. If you want a control to be initially invisible, be sure the Visible property is not selected in the General properties page in the control's Properties view.

Hidden controls do not display by default in the Window painter's Layout view.

❖ **To display hidden controls in the Layout view:**

- Select Design>Show Invisibles from the menu bar.

❖ **To display hidden controls during execution:**

- Assign the value true to the Visible property of each hidden control:

```
controlname.Visible = TRUE
```

Using the Enabled property

If the Enabled property is selected, the control is active. For example, an enabled CommandButton can be clicked, a disabled CommandButton cannot.

If you want a control to display but be inactive, be sure the Enabled property is not selected in the General properties page in the control's Properties view. For example, a CommandButton might be active only after the user has selected an option. In this case, display the CommandButton initially disabled so that it appears grayed out. Then when the user selects the option, enable the CommandButton in a script:

```
CommandButtonName.Enabled = TRUE
```

Choosing colors

The Window painter has two Color drop-down toolbars on PainterBar3 that display colors you can use for the background and foreground of components of the window. Initially, the drop-down toolbars display these color selections:

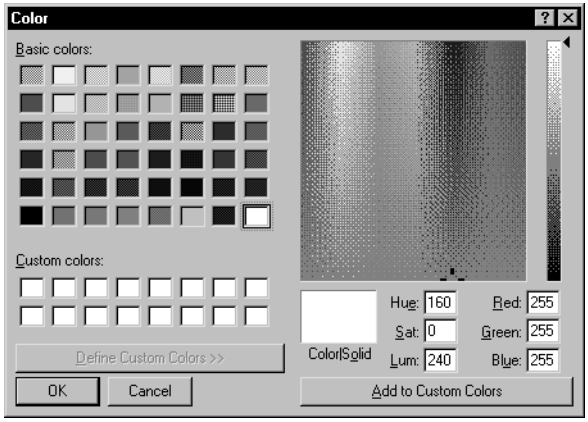
- 20 predefined colors
- 16 custom colors (labeled C)
- The full set of Windows system colors

Do not use Windows system colors for deployed applications
The Windows system colors are those defined by the user in the desktop Windows Control Panel. They are labeled with letters that indicate the type of display element they represent. As they are not defined for the Windows CE device, you should not use them in the applications you build with PocketBuilder.

Defining custom colors

You can define your own custom colors for use in windows, user objects, and DataWindow objects.

Figure 11-2: Dialog box for defining custom colors



❖ **To define and maintain custom colors:**

1 Select Design>Custom Colors from the menu bar.

The Color dialog box displays.

2 Click in an empty color box in the list of custom colors.

3 Choose an existing color or create the color you want.

You can start with one of the basic colors and customize it in the palette to the right by dragging the color indicator with the mouse. You can also specify precise values to define the color.

4 When you have the color you want, click Add to Custom Colors.

The new color displays in the list of custom colors.

5 Select the new color in the list of custom colors.

6 Click OK.

The new color displays in the Color drop-down toolbars and is available in all windows, user objects, and DataWindow objects you create.

PocketBuilder saves custom colors in the [Colors] section of the PocketBuilder initialization file, so they are available across sessions.

Specifying foreground and background colors

You can assign colors to controls using the PainterBar or the Properties view. The page in the Properties view that you use to assign colors depends on the control.

For some controls you can change the background color only. For others, including the CommandButton, PictureButton, PictureHyperLink, Picture, ScrollBars, TrackBars, and ProgressBars, you can change neither the foreground nor the background color.

❖ **To assign a color using the PainterBar:**

1 Select the control.

2 Select either the foreground or background color button from the PainterBar.

3 Select a color from the drop-down toolbar.

Using the 3D look

Although it is not standard for Windows CE applications, you can give the applications you develop with PocketBuilder a three-dimensional look and feel. To use this appearance for an application, select a 3D border for your SingleLineEdit boxes and other controls and make the window background gray.

❖ **To use the 3D look by default:**

1 Select Design>Options from the menu bar.

The Options dialog box displays.

2 On the General properties page, select Default to 3D.

When you build a new window, PocketBuilder automatically sets the window background color to gray and uses 3D borders when you place controls.

PocketBuilder records this preference in the Default3D variable in the [Window] section of the PocketBuilder initialization file, so the preference is maintained across sessions.

Using the individual controls

Table 11-2 lists basic types of controls by the purpose that they serve in typical applications.

Table 11-2: Summary of basic control types by function

Function	Controls include
Invoke actions	CommandButtons, PictureButtons, PictureHyperLinks, StaticHyperLinks, Tabs, Toolbars, User Objects
Display and/or accept data	ListBoxes, DropDownListBoxes, DataWindow controls, StaticText, ListViews, TreeViews, Graphs, Pictures, ProgressBars, ScrollBars, SingleLineEdits, MultiLineEdits, EditMasks, Tabs, Signature controls, User Objects
Indicate choices	RadioButtons (you can group these controls in a GroupBox), CheckBoxes, TrackBars
Enhance presentation	Line, Rectangle, RoundRectangle, Oval

Smartphone platform
Several control types are not fully supported on Smartphone platforms. These include Tabs, Toolbars, RadioButtons, and Signature controls. ListBoxes and DropDownListBoxes are converted automatically to Spinner controls on the Smartphone platform.

How to use the controls

Generally you should use the controls only for the purposes shown in the preceding table. For example, users expect to use radio buttons to select an option. Do not also have them use a RadioButton to invoke an action, such as opening a window or printing. Use a CommandButton for that purpose.

There are, however, several exceptions: user objects can be created for any purpose, and ListBoxes, ListViews, TreeViews, and Tabs are often used both to display data and to invoke actions. For example, double-clicking a ListBox item often causes some action to occur.

The rest of this chapter describes features that are unique to individual controls:

- "Using CommandButtons" on page 248
- "Using PictureButtons" on page 249
- "Using RadioButtons" on page 250
- "Using CheckBoxes" on page 251
- "Using StaticText" on page 252
- "Using StaticHyperLinks" on page 252

Objects and controls described elsewhere

For controls that are not described here, see the following chapters of this *User's Guide*:

- For information about user objects, see Chapter 14, "Working with User Objects"
- For information about controls and objects that are specific to the Windows CE platforms, see Chapter 15, "Working with Native Objects and Controls for Windows CE Devices"
- For information about DataWindow controls and objects, see Chapter 17, "Defining DataWindow Objects"
- For information about graph controls, see Chapter 24, "Working with Graphs"
- For information about the Today item, see "Application object properties for a custom Today item" on page 66

Using CommandButtons

CommandButtons are used to carry out actions. For example, you can use an OK button to confirm a deletion or a Cancel button to cancel a requested deletion. If there are many related CommandButtons, place them along the right side of the window; otherwise, place them along the bottom of the window.

You cannot change the color or alignment of text in a CommandButton.

If clicking the button opens a window that requires user interaction before any other action takes place, use ellipsis points in the button text; for example, "`Print...`".

Specifying Default and Cancel buttons

Default command button

You can specify that a CommandButton is the default button in a window by selecting Default in the General properties page in the button's Properties view.

When there is a default CommandButton and the user presses the Enter key:

- If the focus is not on another CommandButton, the default button's Clicked event is triggered
- If the focus is on another CommandButton, the Clicked event of the button with focus is triggered

---

Other controls affect default behavior
If the window does not contain an editable field, use the SetFocus function or the tab order setting to make sure the default button behaves as described above.

A bold border is placed around the default CommandButton.

Cancel command button

You can define a CommandButton as the cancel button by selecting Cancel in the General properties page in the button's Properties view. If you define a cancel CommandButton, the cancel button's Clicked event is triggered when the user presses the Esc key.

Default and cancel actions in Windows CE applications
Because it is usually easier to click a command button with the stylus than to use the SIP or peripheral keyboard, it is not standard practice in Windows CE applications to use the Enter and Esc keys for default and cancel actions. There is no Esc key on the Smartphone, but you can use the Action key for the Enter key.

Using PictureButtons

A PictureButton is identical to a CommandButton in its functionality. The only difference is that you can specify a picture to display on the PictureButton. The picture can be a PocketBuilder stock icon or a BMP, GIF, animated GIF, JPEG, or PNG file.

Using JPEG or PNG image files
Unlike BMP and GIF images, which can reside in a PocketBuilder resource file (PKR) or in a database blob, JPEG and PNG images must reside in the file system. You must deploy JPEG or PNG files to the current application directory or include the full path of the deployed image files that you want to use in a Pocket PC device or emulator.

PNG files are not supported on the desktop, only on Pocket PC devices and emulators. You cannot add a PNG file to a picture button control in the PocketBuilder UI, only in script. For example:

```
pb_1.PictureName = "\Program Files\my_pic.png"
```

You can choose to display one picture if the button is enabled and a different picture if the button is disabled.

Use these controls when you want to be able to represent the purpose of a button with a picture instead of text.

❖ **To specify a picture:**

1 Select the PictureButton to display its properties in the Properties view.

2 On the General properties page, enter the name of the image file you want to display when the button is enabled, or use the Browse button and choose a file.

3 Enter the name of the image file you want to display when the button is disabled, or use the Browse Disabled button and choose a file.

If the PictureButton is defined as initially enabled, the enabled picture displays in the Layout view. If the PictureButton is defined as initially disabled, the disabled picture displays in the Layout view.

❖ **To specify button text alignment:**

1 Select the PictureButton to display its properties in the Properties view.

2 On the General properties page, enter the text for the PictureButton in the Text box.

3 Use the HTextAlign and VTextAlign lists to choose how you want to display the button text.

Using RadioButtons

RadioButtons are round buttons that represent mutually exclusive options. When a RadioButton is selected, it has a dark center; when it is not selected, the center is blank. RadioButtons exist in groups. Exactly one RadioButton is selected in each group.

Smartphone platforms
The Action key on a Smartphone device or emulator selects or clears a single radio button, but does not change other radio buttons in the same radio button group. Avoid using radio buttons in applications deployed to Smartphone platforms. You should use check boxes or list boxes instead.

When a window opens, one RadioButton in a group must be selected. You specify which RadioButton is initially selected by selecting the Checked property in the General properties page in the RadioButton's Properties view. When the user clicks a RadioButton in a deployed application, the RadioButton becomes selected, and the previously selected RadioButton in the group becomes deselected.

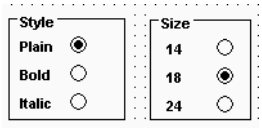
Use RadioButtons to represent the state of an option. Do not use them to invoke actions.

Grouping RadioButtons

By default, all RadioButtons in a window are in one group, regardless of their location in the window. Only one RadioButton can be selected at a time.

You use a GroupBox control to group related RadioButtons. All RadioButtons inside a GroupBox are considered to be in one group. One button can be selected in each group.

Figure 11-3: RadioButtons in 2 GroupBox controls



The Automatic property

When a window contains several RadioButtons that are outside of a GroupBox, the window acts as a GroupBox. Only one RadioButton can be selected at a time, unless the check box for the Automatic property on the RadioButton's General properties page is cleared.

When the Automatic property is not set, you must use scripts to control when a button is selected. Multiple RadioButtons can be selected outside of a group.

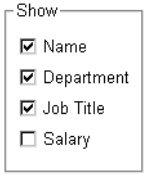
The Automatic property does not change how RadioButtons are processed inside a GroupBox.

Using CheckBoxes

CheckBoxes are square boxes used to set independent options. When a CheckBox is selected, it contains a check mark; when it is not selected, it is blank.

CheckBoxes are independent of each other. You can group them in a GroupBox or rectangle to make the window easier to understand and use, but that does not affect the CheckBoxes' behavior; they are still independent.

Figure 11-4: Multiple selections in a GroupBox



Using three states

CheckBoxes usually have two states, on and off, but sometimes you want to represent a third state, such as Unknown. The third state displays as a grayed box with a check mark.

❖ **To enable the third state:**

- Select the ThreeState property in the General page of the CheckBox Properties view

❖ **To specify that a CheckBox's current state is the third state:**

- Select the ThreeState and the ThirdState properties in the General page of the CheckBox Properties view

Using StaticText

You use a StaticText control to display text to the user or to describe a control that does not have text associated with it, such as a list box or edit control.

The user cannot change the text, but you can change the text for a StaticText control in a script by assigning a string to the control's Text property.

StaticText controls have events associated with them, but you will probably seldom write scripts for them because users do not expect to interact with static text.

Indicating accelerator keys

One use of a StaticText control is to label a list box or edit control. If you assign an accelerator key to a list box or edit control, you need to indicate the accelerator key in the text that labels the control. Otherwise, the user would have no way of knowing that an accelerator key is defined for the control. This technique is described in "Defining accelerator keys" on page 240.

Indicating a border style

You can select a border style using the BorderStyle property on the General properties page.

The Border property must also be selected
The BorderStyle property affects the StaticText control only if the Border property check box is selected.

Using StaticHyperLinks

A StaticHyperLink is display text that provides a hot link to a specified Web page. When a user clicks the StaticHyperLink in a window, the user's Web browser opens to display the page.

The StaticHyperLink control has a URL property that specifies the target of the link. You specify the text and URL on the StaticHyperLink control's General properties page in the Properties view.

Since browsers on Windows CE devices all support URL completion, you can enter a partial address—for example, `sybase.com`—instead of the complete address: `http://www.sybase.com`.

Using SingleLineEdits and MultiLineEdits

A SingleLineEdit is a box in which users can enter a single line of text. A MultiLineEdit is a box in which users can enter more than one line of text. SingleLineEdits and MultiLineEdits are typically used for input and output of data.

Smartphone platforms
Smartphones support several modes of text or data entry. In MultiLineEdits, users can press the Action key on the Smartphone keypad to enter a new line. For more information about text entry modes, see Appendix D, "Designing Applications for Windows CE Platforms."

For these controls, you can specify many properties, including:

- Whether the box has a border (the Border property).
- Whether the box automatically scrolls as needed (AutoHScroll and, for MultiLineEdits, AutoVScroll).
- For SingleLineEdits, whether the box is a Password box so that asterisks are displayed instead of the actual entry (Password).
- The case in which to accept and display the entry (TextCase).
- Whether the selection displays when the control does not have focus (HideSelection). This property is ignored on Smartphone platforms.

For more information about properties of these controls, right-click in any tab page in the Properties view and select Help from the pop-up menu.

Using EditMasks

Sometimes users need to enter data that has a fixed format. For example, U.S. and Canadian phone numbers have a three-digit area code, followed by three digits, followed by four digits. You can use an EditMask control that specifies that format to make it easier for users to enter values. Think of an EditMask control as a smart SingleLineEdit: it knows the format of the data that can be entered.

An edit mask consists of special characters that determine what can be entered in the box. An edit mask can also contain punctuation characters to aid the user.

For example, to make it easier for users to enter phone numbers in the proper format, you can specify the following mask, where # indicates a number:

```
(###) ###-####
```

During execution, the punctuation characters (the parentheses and dash) display in the box and the cursor jumps over them as the user types.

Masks in EditMask controls work in windows as they do in display formats and in the EditMask edit style in DataWindow objects. For more information about specifying masks, see the discussion of display formats in Chapter 21, "Displaying and Validating Data."

❖ **To use an EditMask control:**

1 Select the EditMask to display its properties in the Properties view.

2 Name the control on the General properties page.

3 Select the Mask tab.

4 In the MaskDataType drop-down list, specify the type of data that users will enter into the control.

5 In the Mask edit box, type the mask.

You can click the button on the right and select masks. The masks have the special characters used for the specified data type.

6 Specify other properties for the EditMask control.

For information on the other properties, right-click in any tab page in the Properties view and select Help from the pop-up menu.

Control size and text entry
The size of the EditMask control affects its behavior. If the control is too small for the specified font size, users might not be able to enter text.

To correct this, either specify a smaller font size or resize the EditMask control.

Validation for EditMask controls

The EditMask control checks the validity of a date when you enter it, but if you change a date so that it is no longer valid, its validity is not checked when you tab away from the control. For example, if you enter the date 12/31/2007 in an EditMask control with the mask mm/dd/yyyy, you can delete the 1 in 12 so that the date becomes 02/31/2007. To catch problems like this, add validation code to the LoseFocus event for the control.

Keyboard behavior

Some keystrokes have special behavior in EditMask controls. For more information, see "The EditMask edit style" on page 547.

Using spin controls

You can define an EditMask as a spin control, which is an edit control that contains up and down arrows that users can click to cycle through fixed values. For example, assume you want to allow your users to select how many copies of a report to print. You could define an EditMask as a spin control that allows users to select from a range of values.

Figure 11-5: Edit mask as a spin control

❖ **To define an EditMask as a spin control:**

1 Name the EditMask and provide the data type and mask, as described above.

2 Select the Spin check box on the Mask property page.

3 Specify the needed information.

For example, to allow users to select a number from 1 to 20 in increments of 1, specify a spin range of 1 to 20 and a spin increment of 1.

For more information on the options for spin controls, right-click in any tab page in the Properties view and select Help from the pop-up menu.

Using ListBoxes

A ListBox displays available choices. You can specify scroll bars for a ListBox if more choices exist than can be displayed in the ListBox at one time.

Smartphone platforms
ListBoxes are converted to spinner controls when you deploy them to a Smartphone device or emulator. Extended and multiple selection properties for spinner controls are not supported. For more information about spinner controls, see Appendix D, “Designing Applications for Windows CE Platforms.”

Selecting an item and invoking actions

ListBoxes are an exception to the rule that a control should either invoke an action or be used for viewing and entering data. ListBoxes can do both. ListBoxes display data, but they can also invoke actions.

In Windows applications, clicking an item in the ListBox typically selects the item and triggers the SelectionChanged event. On Pocket PC devices, tapping an item in the ListBox has the same effect. Double-clicking an item (double-tapping the item on a Pocket PC) acts upon the item by triggering the DoubleClicked event.

PocketBuilder automatically highlights an item when a user selects the item at runtime. If you want something to happen when users double-click (or double-tap) an item, you can add a script to the control's DoubleClicked event. The SelectionChanged event is always triggered before the DoubleClicked event. The RButtonDown event on a ListBox control translates to a tap-and-hold action on a Windows CE device.

Populating the list of items

To add items to a ListBox, select the ListBox to display its properties in the Properties view, select the Items tab, and enter the values for the list. Press the Tab key to go to the next line.

In the Items tab page, you can work with rows as described in Table 11-3.

Table 11-3: Working with rows in the Items page of the ListBox Properties view

To do this	Do this
Select a row	Click the row button on the left, or with the cursor in the edit box, press Shift+Space
Delete a row	Select the row and press Delete
Move a row	Click the row button and drag the row where you want it, or press Shift+Space to select the row, then press Ctrl+Up Arrow or Ctrl+Down Arrow to move the row
Delete text	Click the text and select Delete from the pop-up menu

Changing the list during execution
To change the items in the list at runtime, use the functions AddItem, DeleteItem, and InsertItem.

Setting tab stops

You can set tab stops for text in ListBoxes (and in MultiLineEdits) by setting the TabStop property on the General properties page. You can define up to 16 tab stops. The default is a tab stop every eight characters.

You can also define tab stops in a script. Here is an example that defines two tab stops and populates a ListBox control:

```
// lb_1 is the name of the ListBox.
string f1, f2, f3
f1 = "1"
f2 = "Emily"
```

```
f3 = "Foulkes"
// Define 1st tab stop at character 5.
lb_1.tabstop[1] = 5
// Define 2nd tab stop 10 characters after the 1st.
lb_1.tabstop[2] = 10
// Add an item, separated by tabs.
// Note that the ~t must have a space on either side
// and must be lowercase.
lb_1.AddItem(f1 + " ~t " + f2 + " ~t " + f3)
//Make sure a scroll bar displays for the list box
lb_1.HScrollBar=true
```

Note that this script will not work if it is in the window's Open event, because the controls have not yet been created. The best place to include this script is in a user event that is posted in the window's Open event using the PostEvent function.

Other properties

For ListBoxes, you can specify whether:

- Items in the ListBox are displayed in sorted order
- The ListBox allows the user to select multiple items
- The ListBox displays scroll bars if needed

For more information, right-click in any tab page in the Properties view for a ListBox and select Help from the pop-up menu.

Using DropDownListBoxes

DropDownListBoxes combine the features of a SingleLineEdit and a ListBox.

Smartphone platforms
ListBoxes and DropDownListBoxes are converted to spinner controls when you deploy them to a Smartphone device or emulator. Edit functions for spinner controls are not supported. For more information about spinner controls, see Appendix D, “Designing Applications for Windows CE Platforms.”

There are two types of DropDownListBoxes:

- Noneditable
- Editable

Noneditable boxes

If you want your user to choose only from a fixed set of choices, make the DropDownListBox noneditable.

In this type of DropDownListBox, the only valid values are those in the list.

The SIP keyboard as well as peripheral keyboards can be used to select items from the list in a deployed application when the DropDownListBox has focus. Users can make selections from the list box by:

- Using the arrow keys to scroll through the list.
- Typing a character. The ListBox scrolls to the first entry in the list that begins with the typed character. Typing the character again scrolls to the next entry beginning with the character unless the character can be combined with the first to match an entry.
- Using the stylus on the Pocket PC to select the down arrow in the DropDownListBox control to display the list, then selecting an entry from the list.

Editable boxes

If you want to give users the option of specifying a value that is not in the list, make the DropDownListBox editable by selecting the AllowEdit check box on the General properties page.

With an editable DropDownListBox, you can choose whether to have the list always display or not. If you do not choose to always display an editable list, the user can still display the list in the deployed application by tapping (Pocket PC device) or clicking (emulator) the down arrow.

Populating the list

You specify the list in a DropDownListBox the same way as for a ListBox. For information, see “Using ListBoxes” on page 255.

Specifying the size of the drop-down box

To indicate the size of the box that drops down, size the control in the Window painter using the mouse. When the control is selected in the painter, the full size, including the drop-down box, is shown.

Other properties

As with ListBoxes, you can specify whether the list is sorted and whether the edit control is scrollable.

For more information, right-click in any tab page in the Properties view and select Help from the pop-up menu.

Using Pictures

Pictures are controls that display images. These can be PocketBuilder stock icons or BMP, GIF, JPEG, or PNG files.

Using JPEG or PNG image files

Unlike BMP and GIF images, which can reside in a PocketBuilder resource file (PKR) or in a database blob, JPEG and PNG images must reside in the file system. You must deploy JPEG or PNG files to the current application directory or include the full path of the deployed image files that you want to use in a Pocket PC device or emulator.

PNG files are not supported on the desktop, only on Pocket PC devices and emulators. You cannot add a PNG file to a picture control in the PocketBuilder UI, only in script. For example:

```
p_1.PictureName = "\Program Files\my_pic.png"
```

❖ **To display a picture:**

1 Place a picture control in the window.

2 In the General properties page in the Properties view, enter the name of the file you want to display in the PictureName text box or browse to select a file.

The picture displays.

You can choose to resize or invert the image.

Picture controls have events that can be triggered when users click on them. They can also be used simply to display an image. Be consistent in their use so users know what they can do with them.

Using PictureHyperLinks

A PictureHyperLink is a picture that provides a hot link to a specified Web page. When a user clicks the PictureHyperLink in a window, the user's Web browser opens to display the page.

The PictureHyperLink control has a URL property that specifies the target of the link. You specify the picture and URL in the PictureHyperLink control's Properties view in the General properties page. Since browsers on Windows CE devices all support URL completion, you can enter a partial address—for example, `sybase.com`—instead of the complete address: `http://www.sybase.com`.

The PictureHyperLink control is a descendant of the Picture control. Like a Picture control, a PictureHyperLink control can display a PocketBuilder stock icon or an image from a BMP, GIF, animated GIF, JPEG, or PNG file. You select the picture in the same manner.

For more information, see "Using Pictures" on page 259.

Using drawing objects

PocketBuilder provides the following drawing objects: Line, Oval, Rectangle, and RoundRectangle. Drawing objects are usually used only to enhance the appearance of a window or to group controls. However, constructor and destructor events are available, and you can define your own unmapped events for a drawing object. A drawing object does not receive Windows messages, so a mapped event would not be useful.

You can use the following functions to manipulate drawing objects during execution:

- Hide
- Move
- Resize
- Show

In addition, each drawing object has a set of properties that define its appearance. You can assign values to the properties in a script to change the appearance of a drawing object.

Never in front
A drawing object cannot be placed on top of another control that is not a drawing object, such as a GroupBox. Drawing objects always appear behind other controls whether or not the Bring to Front or Send to Back items on the pop-up menu are set. However, drawing objects can be on top of or behind other drawing objects.

Using HProgressBars and VProgressBars

HProgressBars and VProgressBars are rectangles whose highlighting indicates the progress of a lengthy operation, such as an installation program that copies a large number of files. The progress bar gradually fills with the system highlight color as the operation progresses.

You can set the range and current position of a progress bar in the Properties view using the MinPosition, MaxPosition, and Position properties. To specify the size of each increment, set the SetStep property.

Using HScrollBars and VScrollBars

You can place freestanding scroll bar controls within a window. Typically, you use these controls to do one of the following:

- Provide a slider control that allows the user to specify a continuous value
- Graphically display information to the user

You can set the position of the scroll box by specifying the value for the control's Position property. When the user drags the scroll box, the value of Position is automatically updated.

Smartphone platforms
Avoid using scroll bars in applications that you deploy to a Smartphone device or emulator. User interaction with these controls is problematic on Smartphone platforms.

Using HTrackBars and VTrackBars

The HTrackBar and VTrackBar controls have sliders that move in discrete increments inside a horizontal or vertical channel. Like a scroll bar, a track bar typically functions as a slider control that allows users to specify a value or see a value you have displayed graphically, but on a discrete rather than continuous scale. Clicking on the slider moves it in discrete increments instead of continuously.

Smartphone platforms
Avoid using track bars in applications that you deploy to a Smartphone device or emulator. User interaction with these controls is problematic on Smartphone platforms.

Setting properties and placing tick marks

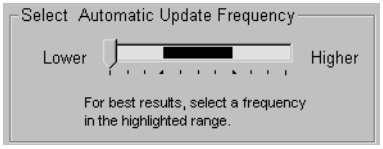
Use a track bar when you want the user to select a discrete value. For example, you might use a track bar to enable a user to select a timer interval or the size of a window.

You can set properties such as minimum and maximum values. You can also set the frequency of tick marks and where tick marks display. Typically a horizontal track bar has a series of tick marks along the bottom of the channel, and a vertical track bar has tick marks on the right.

Highlighting a range of values

You can highlight a range of values in the trackbar with the SelectionRange function. The range you select is indicated by a black fill in the channel and an arrow at each end of the range. This is useful if you want to indicate a range of preferred values. In a scheduling application, the selection range could indicate a block of time that is available for scheduling purposes. Setting a selection range does not prevent the user from selecting a value either inside or outside the range.

Figure 11-6: Horizontal track bar with a highlighted range



Using ListView controls

A ListView control lets you display items and icons in a variety of arrangements. You can display large or small icons in free-form lists. Using PowerScript functions such as AddColumn, AddLargePicture, SetItem, SetColumn, and so on, you can add columns, pictures, and items to the ListView and modify column properties.

Smartphone platforms
PocketBuilder application users can use the arrow keys on the Smartphone to navigate the items in a ListView control, but you must program a menu item or soft key to move the focus from the ListView to another control in the same main window.

For information about ListView functions, see the online Help.

Figure 11-7: List view with small icons

Adding ListView Items and pictures

The ListView control's Properties view has two tab pages for adding pictures: Large Picture (default size 32 by 32 pixels) and Small Picture (16 by 16 pixels). Each picture you enter gets an index number that you can associate with an item in the Items page.

You can choose from a group of stock images provided by PocketBuilder or use BMP, GIF, ICO, JPEG, or PNG files for the images that you add to a ListView. You cannot add a PNG file to a picture control in the PocketBuilder UI, only in script. (PNG files are not supported on the desktop, only on Pocket PC devices and emulators.)

❖ **To add ListView items:**

1 Select the ListView control to display its properties in the Properties view and then select the Items tab.

2 Enter the name of the ListView item and the picture index you want to associate with it.

The picture index corresponds to the images you select on the Large Picture, Small Picture, and State property pages.

On the Items tab page, you work with rows in the same way that you do for a ListBox control. For more information on working with rows on the Items tab page, see Table 11-3 on page 256.

Setting the picture index to zero
Setting the picture index for the first item to zero clears all the settings on the tab page.

3 Set properties for the item on the Large Picture, Small Picture, and/or State tab pages as you did on the Items tab page.

   On these pages, you can also browse for a picture. To do so, click the browse button or press F2.

4 Repeat until all the items are added to the ListView.

Choosing a ListView style

You can display a ListView in four styles:

- Large icon (default)
- Small icon
- List
- Report

❖ **To select a ListView style:**

1 Select the ListView control to display its properties in the Properties view and then select the General tab.

2 Select the type of view you want from the View drop-down list.

Setting other properties

You can modify other properties of the ListView control on the tab pages of the control's Properties view.

❖ **To specify other ListView properties:**

1 Select the ListView control to display its properties in the Properties view.

2 Choose the tab appropriate to the property you want to specify and change the properties as needed.

Choose this tab	To specify
General	The border style
	Whether the user can delete items
Large Picture	The images for ListView items in large icon view
Small Picture	The images for ListView items in small icon, list, and report views
State	The state images for ListView items
Items	The names and associated picture index for ListView items

Choose this tab	To specify
Font	The font size, family, and color for ListView items
Other	The size and position of the ListView

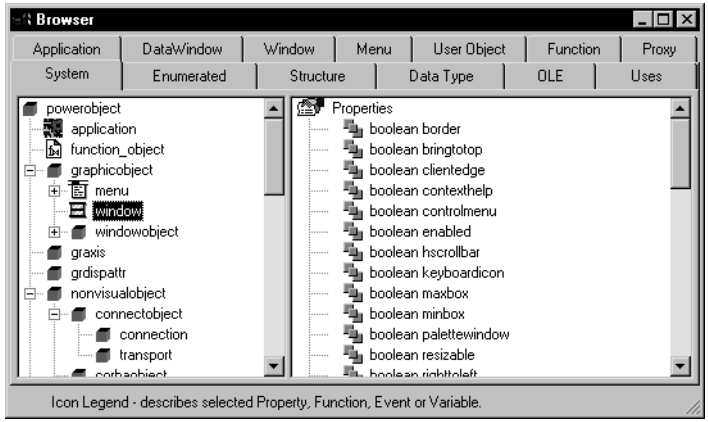
For more information about other properties, right-click in any tab page in the Properties view and select Help from the pop-up menu or see *Objects and Controls*. Because of target platform differences, some of the properties listed for PowerBuilder objects and controls do not apply to PocketBuilder objects and controls.

For more information on the ListView control, see the chapter on using lists and tree views in the *Resource Guide.*

Using TreeView controls

You can use TreeView controls in your application to represent relationships among hierarchical data. An example of a TreeView implementation is PocketBuilder's Browser. The tab pages in the Browser contain TreeView controls.

Figure 11-8: Tree view panes in the PocketBuilder Browser



Smartphone platforms
PocketBuilder application users can use the arrow keys on the Smartphone to navigate the items in a TreeView control, but you must program a menu item or soft key to move the focus from the TreeView to another control in the same main window.

Adding TreeView items and pictures

A TreeView consists of TreeView items that are associated with one or more pictures. You add images to a TreeView in the same way you add images to a ListView, except that you use the Pictures tab page instead of the Large Picture or Small Picture tab pages.

You can choose from a group of stock images provided by PocketBuilder, or use BMP, GIF, ICO, JPEG, or PNG files for the images that you add to a TreeView. You cannot add a PNG file to a picture control in the PocketBuilder UI, only in script. (PNG files are not supported on the desktop, only on Pocket PC devices and emulators.)

Dynamically changing image size
The image size can be changed during execution by setting the PictureHeight and PictureWidth properties when you create a TreeView.

For more information, see "PictureHeight" and "PictureWidth" in the online Help.

❖ **To add items to a TreeView:**

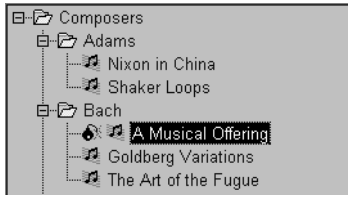
- Write a script in the TreeView constructor event to create TreeView items

  For more information about populating a TreeView, see the chapter on using lists and tree views in the *Resource Guide*.

Adding state pictures to TreeView items

A "state" picture is an image that appears to the left of the TreeView item, indicating that the item is not in its normal mode. A state picture can indicate that a TreeView item is being changed, or that it is performing a process and is unavailable for action.

Figure 11-9: TreeView control showing a state picture



❖ **To specify a state picture for a TreeView item:**

1 Select the TreeView control to display its properties in the Properties view and then select the State tab.

2 Do one of the following:

- Use the StatePictureName drop-down list to select stock pictures to add to the TreeView
- Use the Browse button to select a BMP, ICO, GIF, or JPEG file

Working with the rows in the State or Pictures tab page in the Properties view is the same as working with rows in the State, Small Picture, or Large Picture tab pages in a ListView control. For information, see "Using ListView controls" on page 262.

❖ **To activate a state picture for a TreeView item:**

- Write a script that changes the image when appropriate

  For example, the following script gets the current TreeView item and displays the state picture for it.

```
long           ll_tvi
treeviewitem   tvi

ll_tvi = tv_foo.finditem(currenttreeitem! , 0)
tv_foo.getitem(ll_tvi , tvi)
tvi.statepictureindex = 1
tv_foo.setitem(ll_tvi, tvi)
```

  For more information on the TreeView control, see the chapter on using lists and tree views in the *Resource Guide.*

Setting other properties

You can modify other properties of the TreeView control on the tab pages of the control's Properties view.

❖ **To specify other TreeView properties:**

1 Select the TreeView control to display its properties in the Properties view and then select the General tab.

2 Enter a name for the TreeView in the Name text box and specify other properties as appropriate.

Among the properties you can specify on the General properties page are:

- The border style
- Whether the TreeView has lines showing the item hierarchy
- Whether the TreeView includes collapse and expand buttons
- Whether the user can delete items
- Whether the user can drag and drop items into the TreeView

3 For other options, choose the tab appropriate to the property you want to specify:

Choose this tab	To specify
Pictures	The images used to represent TreeView items
State	The state images for the TreeView items
Font	The font size, family, and color for TreeView items
Others	The size and position of the TreeView

For more information about TreeView properties, right-click in any tab page in the Properties view and select Help from the pop-up menu, or see *Objects and Controls* in the online Help. Because of differences in the target platform, some of the properties listed for PowerBuilder objects and controls do not apply to PocketBuilder objects and controls.

For more information on the TreeView control, see the chapter on using lists and tree views in the *Resource Guide*.

Using Tab controls

A Tab control is a container for tab pages that display other controls. You can add a Tab control to a window in your application to present information that can logically be grouped together but might also be divided into distinct categories. An example is the use of tab pages in the Properties view for objects in PocketBuilder.

Smartphone platforms
Tab controls are not supported on Smartphone devices or emulators.

When you add a Tab control to a window, PocketBuilder creates a Tab control with one tab page labeled “none”. The control is rectangular. By default, the tab label is at the bottom of the tab page.

Distinguishing tab controls and tab pages

Typically you use more than one tab page on a tab control. After you add a tab control with a single tab page to a window, you add as many tab pages as your application requires.

Each tab page displays a label that is visible even when a different tab page is selected. If the number of tab pages is too large to display all the tab pages, a miniature scroll bar is displayed on the screen at the level of the visible tab labels. The scroll bar enables users to scroll to hidden tab labels.

To select the Tab control at design time, click any of the tab labels, or click in the area adjacent to the tab labels but outside the current tab page.

To select a tab page, click its label and then click anywhere on the tab page except the label itself. Selection handles display at the corners of the tab page, but do not include the tab control labels. (When the tab control is selected rather than the tab page, the selection handles include the tab control labels.)

Adding tab pages to a Tab control

You can add a new Tab control to a window by selecting Insert>Control>Tab and clicking in the window. The control has one tab page when it is created. Use the following procedure to place additional tab pages in the tab control.

❖ **To create a new tab page in a Tab control:**

1 Select the Tab control by clicking on a tab label or in the area to the right of the labels when the labels display at the bottom or top of the tab pages.

   The handles that indicate that the Tab control is selected display at the corners of the Tab control. If you selected the tab page, the handles display at the corners of the area above the tab labels when the labels are at the bottom of the control.

   The pop-up menu for a tab page does not allow you to insert additional tab pages, only to modify the order of the controls on the selected tab.

2 Choose Insert TabPage from the pop-up menu.

3 Add controls to the new tab page.

Creating a reusable tab page

You can create reusable tab pages in the User Object painter by defining a tab page with controls on it that is independent of a Tab control. You can then add that tab page to one or more Tab controls.

❖ **To define a tab page that is independent of a Tab control:**

1. Click the New button on the PowerBar and use the Custom Visual icon on the Object tab page to create a custom visual user object.
2. Size the user object to match the size of the Tab controls in which you will use it.
3. To the user object, add the controls that you want to display on the tab page.
4. Select the user object (not one of the controls you added) and on the TabPage page in the Properties view, specify the information to be used by the tab page:
   - Text—The text to display on the tab
   - PictureName—A picture to display on the tab with or instead of the text
   - PowerTipText—Text for a pop-up message to display when the user moves the cursor to the tab
   - Colors for the tab and its text
5. Save and close the user object.

Adding a reusable tab page to a Tab control

After you have created a user object that can be used as a tab page, you can add it to a Tab control. You cannot add the user object to a Tab control if the user object is open. After you have added the user object to the control, you also cannot open the user object and the window that contains the Tab control at the same time.

❖ **To add a tab page that exists as an independent user object to a Tab control:**

1. In the Window painter, right-click the Tab control.
2. Choose Insert User Object from the pop-up menu.
3. Select a user object that you have set up as a tab page and click OK.

A tab page, inherited from the user object you selected, is inserted. You can select the tab page, set its tab page properties, and write scripts for the inherited user object just as you do for tab pages defined within the Tab control, but you cannot edit the content of the user object within the Tab control. If you want to edit the controls, close the Window painter and go back to the User Object painter to make changes.

Manipulating the Tab control

You can use the Layout view to open other views of the Tab control and to move, resize, or delete the Tab control.

❖ **To change the name and properties of the Tab control:**

1 Click any of the tabs in the Tab control to display the Tab control properties in the Properties view.

2 Edit the properties.

For more information, right-click in the Properties view and select Help from the pop-up menu.

❖ **To change the scripts of the Tab control:**

1 Double-click one of the tab labels or right-click a tab label and select Script from the pop-up menu.

2 Select the Tab control event you want from the second drop-down list of the Script view and edit the script.

❖ **To move a Tab control:**

- With the mouse pointer on one of the tab labels, hold down the left mouse button and drag to move the control to the new position

The Tab control and all tab pages are moved as a group.

❖ **To resize a Tab control:**

- Grab a border of the control and drag it to the new size

The Tab control and all tab pages are sized as a group.

❖ **To delete a Tab control:**

- With the mouse pointer on one of the tabs, select Cut or Delete from the pop-up menu

Manipulating the tab pages

You can use the Layout view to display any tab page in a Tab control, to open other views of the selected tab page, and to delete the selected tab page from the Tab control.

❖ **To view a different tab page:**

- Click on the label for the tab page you want to view

  The selected tab page is brought to the front. The tabs are rearranged according to the TabPosition setting you have chosen for the Tab control.

❖ **To change the name and properties of a tab page:**

1 Select the tab.

  Selecting a tab can move it to the position for a selected tab based on the TabPosition setting for the Tab control. For example, if TabPosition is set to tabsonbottomandtop! and a tab displays at the top, it moves to the bottom when you select it.

2 Click anywhere on the tab page except the tab label.

3 Edit the properties.

❖ **To change the scripts of the tab page:**

1 Select the tab.

  It may move to the position for a selected tab based on the Tab Position setting for the Tab control.

2 Click anywhere on the tab page except the tab label.

3 Select Script from the tab page's pop-up menu.

4 Select a script and edit it.

❖ **To delete a tab page from a Tab control:**

- With the mouse pointer anywhere on the tab page except the tab label, select Cut or Delete from the pop-up menu

Managing controls on tab pages

You can add and move controls on tab pages the same way you add and move controls on a window.

❖ **To add a control to a tab page:**

- Choose a control from the toolbar or the Control menu, then click in the tab page where you want to add the control

  You can add controls only to a tab page created within the Tab control. To add controls to an independent tab page, open it in the User Object painter.

❖ **To move a control from one tab page to another:**

- Cut or copy the control and paste it on the destination tab page

  The source and destination tab pages must be embedded tab pages, not independent ones created in the User Object painter.

❖ **To move a control between a tab page and the window containing the Tab control:**

- Cut or copy the control and paste it on the destination window or tab page

  Moving the control between a tab page and the window changes the control's parent, which affects scripts that refer to the control.

  For more information on the Tab control, see the chapter on using tab controls in the *Resource Guide*.

CHAPTER 12

Understanding Inheritance

About this chapter

This chapter describes how to use inheritance to build PocketBuilder objects.

Contents

About inheritance

One of the most powerful features of PocketBuilder is inheritance. It enables you to build windows, user objects, and menus that are derived from existing objects.

Using inheritance has a number of advantages:

- When you change an ancestor object, the changes are reflected in all the descendants. You do not have to make manual changes in the descendants as you would in a copy. This saves you coding time and makes the application easier to maintain.
- The descendant inherits the ancestor's scripts so you do not have to re-enter the code to add to the script.
- You get consistency in the code and objects in your applications.

This chapter describes how inheritance works in PocketBuilder and how to use it to maximize your productivity.

Opening ancestors and descendants
To enforce consistency, PocketBuilder does not let you open an ancestor object until you have closed any descendants that are open, or to open a descendent object when its ancestor is open.

Creating new objects using inheritance

You use the Inherit From Object dialog box to create a new window, user object, or menu using inheritance.

❖ **To create a new object using inheritance:**

1 Click the Inherit button in the PowerBar, or select File>Inherit from the menu.

2 In the Inherit From Object dialog box, select the object type (menu, user object, or window) from the Objects of Type drop-down list.

3 Select the target as well as the library or libraries you want to look in, then select the object from which you want to inherit the new object.

Displaying objects from many libraries
To find an object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to toggle selected libraries and Shift+Click to select a range.

4 Click OK.

The new object, which is a descendant of the object you chose to inherit from, opens in the appropriate painter.

The inheritance hierarchy

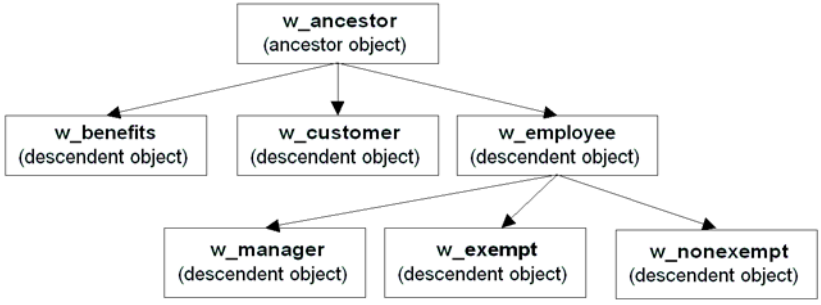
When you build an object that inherits from another object, you are creating a hierarchy (or tree structure) of ancestor objects and descendent objects. Chapter 10, "Working with Windows," uses the example of creating two windows, w_customer and w_employee, that inherit their properties from a common ancestor, w_ancestor. In this example, w_employee and w_customer are the descendants.

The object at the top of the hierarchy is a base class object, and the other objects are descendants of this object. Each descendant inherits information from its ancestor. The base class object typically performs generalized processing, and each descendant modifies the inherited processing as needed.

Multiple descendants

An object can have an unlimited number of descendants, and each descendant can also be an ancestor. For example, if you build three windows that are direct descendants of the w_ancestor window and three windows that are direct descendants of the w_employee window, the hierarchy looks like this:

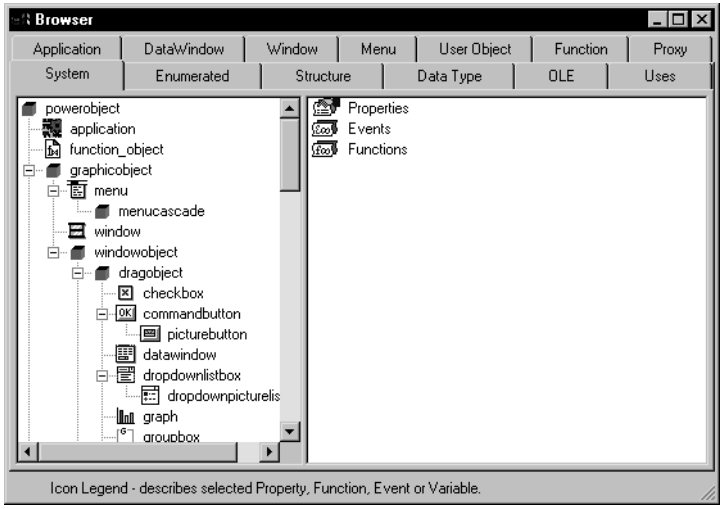
Figure 12-1: Object hierarchy example



Browsing the class hierarchy

PocketBuilder provides a Browser that can show the hierarchy of the built-in PocketBuilder system objects and the hierarchy of the ancestor and descendent windows, menus, and user objects you create. In object-oriented terms, these are called class hierarchies: each PocketBuilder object defines a class.

Figure 12-2: Displaying the hierarchy of system classes in the Browser



Regenerating objects
The Browser also provides a convenient way to regenerate objects and their descendants. For more information, see "Regenerating library entries" on page 104.

❖ **To browse the class hierarchy of PocketBuilder system objects:**

1 Click the Browser button in the PowerBar.

2 Choose the System tab to show the built-in PocketBuilder objects.

3 In the left pane, scroll down the object list and select "powerobject".

4 Display the pop-up menu for powerobject and choose Show Hierarchy.

5 Select Expand All from the pop-up menu and scroll to the top.

The hierarchy for the built-in PocketBuilder objects displays.

Getting context-sensitive Help in the Browser
To get context-sensitive Help for an object, control, or function, select Help from its pop-up menu.

❖ **To display the class hierarchy for other object types:**

1 Choose the Menu, Window, or User Object tab.

   If you choose any other object type, there is no inheritance for the object type, so you cannot display a class hierarchy.

2 In the left pane, select an object and choose Show Hierarchy from its pop-up menu.

3 Select an object and choose Expand All from its pop-up menu.

   PocketBuilder shows the selected object in the current application. Descendent objects are shown indented under their ancestors.

Working with inherited objects

This section describes:

- Working in a descendent object
- Working in an ancestor object
- Resetting properties in a descendant

Working in a descendent object

You can change descendent objects to meet specialized needs. For example, you can:

- Change properties of the descendent object
- Change properties of inherited controls in the object
- Add controls to a descendent window or user object
- Add menu items to a menu

For specifics about what you can do in inherited windows, user objects, and menus, see Chapter 10, “Working with Windows,” Chapter 14, “Working with User Objects,” and Chapter 13, “Working with Menus.”

Working in an ancestor object

When you use inheritance to build an object, the descendant is dependent on the definition of the ancestor. Therefore you should not delete the ancestor without deleting the descendants. You should also be careful when you change the definition of an ancestor object. You might want to regenerate descendent objects if you do any of the following:

- Delete or change the name of an instance variable in the ancestor

- Modify a user-defined function in the ancestor
- Delete a user event in an ancestor
- Rename or delete a control in an ancestor

When you regenerate the descendants, the compiler flags any references it cannot resolve so that you can fix them. For information about regenerating objects, see Chapter 4, "Working with Libraries."

About local changes
If you change a property in an ancestor object, the property will also change in all descendants—as long as you have not already changed that property in a descendant (in which case the property in the descendant stays the same). In other words, local changes always override inherited properties.

Using inherited scripts

In the hierarchy formed by ancestors and descendants, each descendant inherits its event scripts from its immediate ancestor. If an inherited event does not have a script, you can write a script for the event in the descendant. If the inherited event does have a script, the ancestor script will execute in the descendant unless you extend the script or override it. You can:

- Extend the ancestor script—build a script that executes after the ancestor script
- Override the ancestor script—build a script that executes instead of the ancestor script

You cannot delete or modify an ancestor script from within a descendant.

Executing code before the ancestor script

To write a script that executes before the ancestor script, first override the ancestor script and then in the descendent script explicitly call the ancestor script at the appropriate place. For more information, see "Calling an ancestor script" on page 283.

Getting the return value of the ancestor script

To get the return value of an ancestor script, you can use the AncestorReturnValue variable. This variable is always available in descendent scripts that extend an ancestor script. It is also available if you override the ancestor script and use the CALL syntax to call the ancestor event script. For more information, see the chapter on using the PowerScript language in the *Resource Guide*.

Viewing inherited scripts

If an inherited object or control has a script defined only in an ancestor, no script displays in the Script view.

Script icons in the second drop-down list

The second drop-down list in the Script view indicates which events have scripts written for an ancestor, as follows:

- If the event has a script in an ancestor only, the script icon next to the event name in the second drop-down list is displayed in color
- If the event has a script in an ancestor as well as in the object you are working with, the script icon is displayed half in color

Script icons in the third drop-down list

The third drop-down list in the Script view shows the current object followed by each of its ancestors in ascending order. The icon next to an object name indicates whether the object has a script for the event selected in the second drop-down list, as follows:

- If an object is the highest class in the hierarchy to have a script, a transparent script icon displays next to its name. No icons display next to the names of any of its ancestors.
- If an object does not have a script for the event but has an ancestor that has a script for the event, the script icon next to its name is displayed in color.
- If an object has a script for the event, and it has an ancestor that also has a script for the event, the script icon next to its name is displayed half in color.

❖ **To view an ancestor script:**

1 In the first drop-down list in the Script view for an inherited object, select the object itself, and in the second drop-down list, select the event whose script you want to see.

The Script view does not display the script for the ancestor. No script displays.

2 In the third drop-down list in the Script view, select an ancestor object that has a script for the selected event.

The Script view displays any script defined in the ancestor object.

3 To climb the inheritance hierarchy, in the third drop-down list select the script for the grandparent of the current object, the great-grandparent, and so on, until you display the scripts you want.

The Script view displays the scripts for each of the ancestor objects. You can traverse the entire inheritance hierarchy using the third drop-down list.

Extending a script

When you extend an ancestor script for an event, PocketBuilder executes the ancestor script, then executes the script for the descendant when the event is triggered.

❖ **To extend an ancestor script:**

1. In the first drop-down list in the Script view, select the object or a control, and in the second drop-down list, select the event for which you want to extend the script.
2. Make sure that Extend Ancestor Script on the Edit menu or the pop-up menu in the Script view is selected.

   Extending the ancestor script is the default.
3. In the Script view, enter the appropriate statements.

   You can call the script for any event in any ancestor as well as call any user-defined functions that have been defined for the ancestor. For information about calling an ancestor script or function, see "Calling an ancestor script" on page 283 and "Calling an ancestor function" on page 283.

Example of extending a script

If the ancestor script for the Clicked event in a button beeps when the user clicks the button without selecting an item in a list, you might extend the script in the descendant to display a message box in addition to beeping.

Overriding a script

❖ **To override an ancestor script:**

1. In the first drop-down list in the Script view, select the object or a control, and in the second drop-down list, select the event for which you want to override the script.
2. Code a script for the event in the descendant.

   You can call the script for any event in any ancestor as well as call any user-defined functions that have been defined for the ancestor.

   For information about calling an ancestor script or function, see "Calling an ancestor script" on page 283 and "Calling an ancestor function" on page 283.

Override but not execute
To override a script for the ancestor but not execute a script in the descendant, enter only a comment in the Script view.

3 Click on Extend Ancestor Script on the Edit menu or the pop-up menu to clear the check mark.

You do not want Extend Ancestor Script selected. Not extending the ancestor script means that you are overriding the script.

During execution, PocketBuilder executes the descendent script when the event is triggered. The ancestor script is not executed.

Example of overriding a script

If the script for the Open event in the ancestor window displays employee files and you want to display customer files in the descendent window, create a new script for the Open event in the descendant to display customer files, and clear the check mark for the Extend Ancestor Script menu item.

Calling an ancestor script

When you write a script for a descendent object or control, you can call scripts written for any ancestor. You can refer by name to any ancestor of the descendent object in a script, not just the immediate ancestor (parent). To reference the immediate ancestor (parent), you can use the Super reserved word.

For more information about calling scripts for an event in an ancestor window, user object, or menu, and about the Super reserved word, see the *PowerScript Reference* in the online Help.

Calling an ancestor function

When you write a script for a descendent window, user object, or menu, you can call user-defined functions that have been defined for any of its ancestors. To call the first function up the inheritance hierarchy, just call the function as usual:

function (*arguments*)

If there are several versions of the function up the inheritance hierarchy and you do not want to call the first one up, you need to specify the name of the object defining the function you want:

ancestorobject::*function* (*arguments*)

This syntax works only in scripts for the descendent object itself, not in scripts for controls or user objects in the descendent object or in menu item scripts. To call a specific version of an ancestor user-defined function in a script for a control, user object, or menu item in a descendent object, do the following:

1 Define an object-level user-defined function in the descendent object that calls the ancestor function.

2 Call the function you just defined in the descendent script.

For more information about calling an ancestor function, see the *PowerScript Reference* in the online Help.

CHAPTER 13

Working with Menus

About this chapter

You add menus to windows to give your users an easy, intuitive way to select commands and options in your applications. This chapter describes how to define and use menus.

Contents

About menus and menu items

The main windows in an application typically have menus. Menus are lists of related commands or options (menu items) that a user can select in the currently active window. Each choice in a menu is called a menu item. Menu items display in a menu bar or in drop-down or cascading menus.

About Menu objects
Each item in a menu is defined as a Menu object in PocketBuilder. You can see the Menu object in the list of objects in the Browser's System tab.

Menus you define in PocketBuilder work the same as standard menus in your operating environment. Menus cannot be associated with response windows.

About using menus

You can use menus that you build in PocketBuilder in two ways:

- **In the menu bar of windows** Menu bar menus are associated with a window in the Window painter and display whenever the window is opened.
- **As pop-up menus** Pop-up menus display only when a script executes the PopMenu function.

Both uses are described in this chapter.

PocketBuilder gives you freedom to design menus with respect to operating system constraints, but you should still follow conventions that make it easy to use the menus in your deployed applications. For example, you should keep menus simple and consistent, and group related items in a drop-down menu. You should use cascading menus sparingly and restrict them to one or two levels.

Smartphone platforms

Windows in applications for Smartphone devices require a menu object with exactly two main menu items. In deployed applications, the first menu item corresponds to the left menu key on the Smartphone, and the second menu item corresponds to the right menu key.

The right menu key typically opens a menu that has submenu items. If a right menu is not needed, it can be left blank, but it still must be included in any menu object that you deploy with your application. If you do not supply a menu object for a window, a nonfunctional menu is automatically assigned to that window.

Pop-up menus are not recommended for use in Smartphone applications.

For more information about designing applications for Smartphone devices, see Appendix D, “Designing Applications for Windows CE Platforms.”

WM 5 or later platforms

The Microsoft Windows Mobile 5 (WM 5) and Windows Mobile 6 (WM 6) platforms enable the use of Smartphone style menus for all handheld devices. To use this menu style, you must assign a maximum of two top-level menu items in your PocketBuilder applications. Although this is not yet a requirement for Pocket PC devices, it has been and continues to be a requirement for Smartphone devices.

At runtime, the PocketBuilder VM determines whether the operating system platform is WM 5 or later. For these platforms, if the application that you deploy opens a window with a menu containing one or two top-level menu items, PocketBuilder uses the Smartphone menu style—that is, the top-level menu displays as two adjoining buttons, with labels corresponding to the menu item names.

If, however, a window you open on a WM 5 or later Pocket PC uses a menu with more than two top-level menu items, the menu style remains the same as the menu style on Pocket PC devices that do not use a Windows Mobile platform. After a menu with this legacy style displays, all menus in the same application display with the legacy style, even if they have two top-level menu items.

About building menus

When you build a menu, you:

- Specify the appearance and behavior of the menu items by setting their properties.
- Build scripts that determine responses to events in the menu items. To support these scripts, you can declare functions, structures, and variables for the menu.

There are two ways to build a menu. You can:

- Build a new menu from scratch. See "Building a new menu" on page 289.
- Build a menu that inherits its style, functions, structures, variables, and scripts from an existing menu. You use inheritance to create menus that are derived from existing menus, thereby saving time and coding. See "Using inheritance to build a menu" on page 303.

Adding a toolbar with menu functions

You can add a toolbar with picture buttons to deliver the same functionality as an application menu.

For information about the PocketBuilder toolbar control, see Chapter 15, "Working with Native Objects and Controls for Windows CE Devices."

About the Menu painter

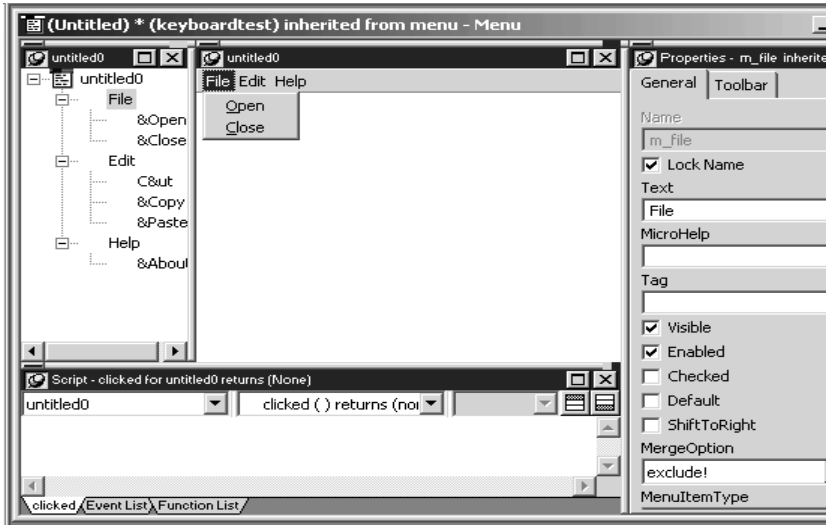
Views in the Menu painter

The Menu painter has several views where you specify menu items and how they look and behave. For general information about views, how you use them, and how they are related, see "Views in painters that edit objects" on page 58.

Tree Menu and WYSIWYG Menu views

The default Menu painter layout shows the Tree Menu view in the top left and the WYSIWYG Menu view in the top middle. The WYSIWYG Menu view displays how the menu will look on the desktop, not on the Windows CE device or emulator.

Figure 13-1: Default Menu painter layout



The Tree Menu and WYSIWYG Menu views are where you specify menu items that display in the menu bar and under items in the menu bar.

Table 13-1: Tree and WYSIWYG views in the Menu painter

This view	Displays
Tree Menu	All the menu items at the same time when the tree is fully expanded. To fully expand the tree or collapse the expanded tree, press Ctrl+Shift+*.
WYSIWYG Menu	The menu as it would appear in a desktop application, with the exception of menu items that do not display at runtime if you set their Visible property to false. In the WYSIWYG Menu view, these items diplay in a dithered mode. In applications that you deploy to the Windows CE device or emulator, menu items appear at the bottom of the window, not at the top as in the WYSIWYG view.

You can use either the Tree Menu view or the WYSIWYG Menu view to insert new menu items on the menu bar or on drop-down (cascading) menus, or to modify existing menu items. The menus in both views change when you make a change in either view.

Building a new menu

This section describes how to build menus from scratch. You use this technique to create menus that are not based on existing menus. For how to create a new menu using inheritance, see "Using inheritance to build a menu" on page 303.

Creating a new menu

You build a new menu by creating a new Menu object and then working on it in the Menu painter.

❖ **To create a new menu:**

1 Click the New button in the PowerBar.

2 On the PB Object tab page, select Menu and click OK.

The Menu painter opens. Because you are creating a new menu and have not added menu items yet, the only content in the Tree Menu view is an untitled top-level tree view item. There is no content visible in the WYSIWYG Menu view.

Working with menu items

A menu consists of at least one menu item on the menu bar and menu items in a drop-down menu. You can add menu items in three places:

- To the menu bar
- To a drop-down menu
- To a cascading menu

Using the pop-up menu
The procedures in this section use the Insert and Edit menus on the PocketBuilder main menu to insert and edit menu items. You can also use the equivalent items on the selected object's pop-up menu.

Inserting menu items

There are three choices on the Insert menu: Menu Item, Menu Item At End, and Submenu Item. Use the first two to insert menu items in the same menu as the selected item, and use Insert>Submenu Item to create a new drop-down or cascading menu for the selected item.

For example, suppose you have created a File menu on the menu bar with two menu items: Open and Exit. Here are the results of some insert operations:

- Select File and select Insert>Menu Item At End

  A new item is added to the menu bar after the File menu.

- Select Open and select Insert>Menu Item

  A new item is added to the File menu above Open.

- Select Open and select Insert>Menu Item At End

  A new item is added to the File menu after Exit.

- Select Open and select Insert>Submenu Item

  A new cascading menu is added to the Open menu item.

Getting the menu started

The first thing you do with a new menu is add the first item to the menu bar. After doing so, you can continue adding new items to the menu bar or to the menu bar item you just created. As you work, the changes you make display in both the WYSIWYG and Tree Menu views.

The first procedure in this section describes how to add a single first item to the menu bar. Use this procedure if you want to add the menu bar item, then work on its drop-down menu. Use the second procedure to add multiple items quickly to the menu bar.

❖ **To insert the first menu bar item in a new menu:**

1 Select Insert>Submenu Item.

  PocketBuilder displays an empty box on the menu bar in the WYSIWYG Menu view and a blank submenu item in the Tree Menu view.

2 Type the text you want for the menu item and press Enter.

❖ **To insert multiple menu bar items in a new menu:**

1 Select Insert>Submenu Item.

  PocketBuilder displays an empty box on the menu bar in the WYSIWYG Menu view and a blank submenu item in the Tree Menu view.

2 Type the text you want for the menu item and press Tab.

PocketBuilder displays a new empty box on the menu bar in the WYSIWYG Menu view and a blank submenu item in the Tree Menu view.

3 Repeat step 2 until you have added all the menu bar items you need.

4 Press Enter to save the last menu bar item.

Adding additional menu items

After you have created the first menu bar item, you can add more items to the menu bar or start building drop-down and cascading menus.

❖ **To insert additional menu items on the menu bar:**

1 Do one of the following:

- With any menu bar item selected, select Insert>Menu Item At End to add an item to the end of the menu bar
- Select a menu bar item and select Insert>Menu Item to add a menu bar item before the selected menu bar item

2 Type the text you want for the menu bar item and then press Enter.

❖ **To add a drop-down menu to an item on the menu bar:**

1 Select the item in the menu bar for which you want to create a drop-down menu.

2 Select Insert>Submenu Item.

PocketBuilder displays an empty box.

3 Type the text you want for the menu item and then press Tab.

4 Repeat Step 3 until you have added all the items you want on the drop-down menu.

5 Press Enter to save the last drop-down menu item.

❖ **To add a cascading menu to an item in a drop-down menu:**

1 Select the item in a drop-down menu for which you want to create a cascading menu.

2 Select Insert>Submenu Item.

PocketBuilder displays an empty box.

3 Type the text you want for the menu item and then press Tab.

4 Repeat step 3 until you have added all the items you want on the cascading menu.

5 Press Enter to save the last cascading menu item.

❖ **To add an item to the end of any menu:**

1 Select any item on the menu.

2 Select Insert>Menu Item At End.

PocketBuilder displays an empty box.

3 Type the text you want for the second menu item in the box and press Enter.

❖ **To insert an item in any existing menu:**

1 Select the item before which the new menu item will display.

2 Select Insert>Menu Item.

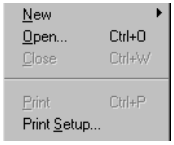
An empty box displays above the item you selected.

3 Type the text you want for the menu item and press Enter.

Creating separation lines in menus

You should separate groups of related menu items with lines.

Figure 13-2: Separation line between two menu groups



❖ **To create a line between items on a menu:**

1 Insert a new menu item where you want the separation line.

2 Type a single dash (-) as the menu item text and press Enter.

A separation line displays.

Duplicating menu items

You might find that you save time creating new menu items if you duplicate existing menu items. A duplicate menu item has the same properties and script as the original menu item. You might be able to modify a long script slightly to make it work for your duplicate menu item.

❖ **To duplicate a menu item or a submenu item:**

1 Select the menu item or the submenu item to duplicate.

2 Select Edit>Duplicate or press Ctrl+T.

The duplicate item displays at the same level of the menu and follows the item you selected. The name of the duplicate menu item is unique.

3 Change the text of the duplicate menu item.

4 Modify the properties and script associated with the duplicate item as needed.

Changing menu item text

You sometimes need to change the text of a menu item, and if you duplicate a menu item, you need to change the text of the duplicate item.

❖ **To change the text of a menu item:**

1 Do one of the following:

- Click the item to select it, then click it again
- Select the item and select Edit>Menu Item Text
- Select the item and open the general page in the Properties view

2 Type the new text for the menu item in the box in the WYSIWYG Menu or Tree Menu view or in the Text box in the Properties view.

Selecting menu items

You can select multiple menu items to move, delete, or change common properties.

❖ **To select multiple menu items:**

- Press and hold Ctrl while selecting each item you want

❖ **To select a range of menu items at the same level in the menu:**

- Select the first item in the range, press Shift, then select the last item in the range

Navigating in the menu

As you work in a menu, you can move to another menu item by selecting it. You can also use the Right Arrow, Left Arrow, Up Arrow, and Down Arrow keys on the keyboard to navigate.

Moving menu items

The easiest way to change the order of items in the menu bar or in a drop-down or cascading menu is to drag the item you want to move and drop it where you want it to be. You can drag items within the same level in a menu structure or to another level. For example, you can drag an item in the menu bar to a drop-down menu or an item in a cascading menu to the menu bar.

WYSIWYG Menu and Tree Menu views
You can use drag and drop within each view. You can also drag from one view and drop in another.

❖ **To move a menu item or submenu item using drag and drop:**

1 Select the item.

2 Press and hold the left mouse button and drag the item to a new location.

A feedback line appears at the new location that indicates where to drop it.

3 Release the mouse button to drop the menu item.

The menu item displays in the new location.

Dragging to copy
To copy a menu item by dragging it, press and hold the Ctrl key while you drag and drop the item. A copied menu item has the same properties and scripts as the original menu item.

You can also copy or move a menu item by selecting the item and using the Cut, Copy, and Paste items on the Edit menu or the pop-up menu.

Deleting menu items

❖ **To delete a menu item:**

1 Select the menu item you want to delete.

2 Click the Delete button in the PainterBar or select Edit>Delete from the menu bar.

How menu items are named

When you add a menu item, PocketBuilder gives it a default name, which displays in the Name box in the Properties view. This is the name by which you refer to a menu item in a script.

About the default menu item names

The default name is a concatenation of the default prefix for menus, m_, and the valid PowerBuilder characters and symbols in the text you typed for the menu item. If there are no valid characters or symbols in the text you typed for the menu item, PocketBuilder creates a unique name m_*n*, where *n* is the lowest number that can be combined with the prefix to create a unique name.

Prefix may be different
The default prefix is different if it has been changed in the Options dialog box.

The complete menu item name (prefix and suffix) can be up to 40 characters. If the prefix and suffix exceed this size, PocketBuilder uses only the first 40 characters without displaying a warning message.

Duplicate menu item names

If you add a menu item that has the same name as an existing menu item, PocketBuilder displays a dialog box that suggests a unique name for the menu item. For example, you might already have an Options item on the Edit menu with the default name m_options. If you add an Options item to another menu, PocketBuilder cannot give it the name m_options.

Menu item names are locked by default

After you add a menu item, the name that PocketBuilder assigns to the menu item is locked. Even if you later change the text that displays for the menu item, PocketBuilder does not rename the menu item. This ensures that you can change the text that displays in a menu without having to revise all your scripts that refer to the menu item by the name that PocketBuilder assigns to it.

If you want to rename a menu item after changing the text that displays for it, you can unlock the name.

❖ **To have PocketBuilder rename a menu item:**

1 On the General properties page in the Properties view, clear the Lock Name check box.

2 Change the text that displays for the menu item.

Saving the menu

You can save the menu you are working on at any time. When you save a menu, PocketBuilder saves the compiled menu items and scripts in the library you specify.

❖ **To save a menu:**

1 Select File>Save from the menu bar.

If you have previously saved the menu, PocketBuilder saves the new version in the same library and returns you to the Menu painter.

If you have not previously saved the menu, PocketBuilder displays the Save Menu dialog box.

2 Name the menu in the Menus box.

The menu name can be any valid identifier up to 40 characters. For information about valid identifiers, see “identifier names” in the online Help.

A common convention is to use a standard prefix of m_ and a suffix that helps you identify the particular menu. For example, you might name a menu used in a sales application m_sales.

3 Write comments to describe the menu.

These comments display in the Select Menu dialog box and in the Library painter. It is a good idea to use comments so that you and others can easily remember the purpose of the menu later.

4 Specify the library in which to save the menu and click OK.

Defining the appearance of menu items

You can use the Menu painter to change the appearance and behavior of your menu and menu items by choosing different settings in the tab pages in the Properties view.

For a list of menu item properties, see *Objects and Controls* in the online Help, or select Help from the pop-up menu in the menu's Properties view and select the Properties button in the Help window. Because of target platform differences, some of the properties listed for PowerBuilder menu items do not apply to PocketBuilder menu items.

Setting General properties

This section describes the properties you can set when you select a menu item and then select the General tab page in the Properties view.

Unlocking menu item names

For information, see "How menu items are named" on page 295.

Setting the appearance of a menu item

On the General tab page in the Properties view, you can also specify how a menu item appears during execution.

Table 13-2: Setting display properties for menu items

Property	Meaning
Visible	Whether the menu item is visible. An invisible menu item still displays in the WYSIWYG and Tree Menu views, but does not display during execution. In the WYSIWYG Menu view, the text of an invisible item has a dithered (faded and dotted) appearance.
Enabled	Whether the menu item can be selected.
Checked	Whether the menu item displays with a check mark next to it.
ShiftToRight	Whether the menu item shifts to the right (or down for a drop-down or cascading menu) when you add menu items in a menu that is inherited from this menu. Selecting this property allows you to insert menu items in descendent menus instead of being able to add them only to the end. For more information, see "Inserting menu items in a descendent menu" on page 305.

The settings you specify here determine how the menu items display by default. You can change the values of the properties in scripts during execution.

Assigning accelerator keys

A menu item can have an accelerator key, also called a mnemonic access key, which allows users to select the item from the SIP by pressing the designated key, or from a peripheral keyboard by pressing Alt+*key* when the menu item is displayed. Accelerator keys display with an underline in the menu item text.

It is not standard practice to include accelerator keys in Windows CE applications. In applications deployed to a Windows CE device or emulator, accelerator keys do not display well for menu bar items. Therefore, if used at all, accelerator keys should be used only in drop-down menus.

❖ **To assign an accelerator key:**

- Type an ampersand (&) before the letter in the menu item text that you want to designate as the accelerator key.

  For example, &Open designates the O in Open as an accelerator key and Co&py designates the p in Copy as an accelerator key.

Displaying an ampersand in the text
If you want an ampersand to display in the menu text, type two ampersands. For example, `Fish&&Chips` displays as Fish&Chips with no accelerator key. To display Fish&Chips as the menu text with the C underlined as the accelerator, type `Fish&&&Chips`.

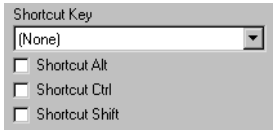
Assigning shortcut keys

Shortcut keys are combinations of keys that a user can press to select a menu item whether or not the menu is displayed. Although it is not standard practice to use shortcut keys in Windows CE applications, PocketBuilder permits you to define them for use in your applications.

If you specify the same shortcut for more than one MenuItem, the command that occurs later in the menu hierarchy is executed.

Some shortcut key combinations, such as Ctrl+C, Ctrl+V, and Ctrl+X, are commonly used by many applications. Avoid using these combinations when you assign shortcut keys for your application.

Figure 13-3: Shortcut key selection box in Properties view



❖ **To assign a shortcut key:**

1 Select the menu item to which you want to assign a shortcut key.

2 Select the General tab in the Properties view.

3 Select a key from the Key drop-down list.

4 Select Shortcut Alt, Shortcut Ctrl, and/or Shortcut Shift to create a key combination.

PocketBuilder displays the shortcut key next to the menu item name. Note that the SIP does not have an Alt key.

Writing scripts for menu items

You write scripts for menu items in the Script view. The scripts specify what happens when users select a menu item.

❖ **To write a script for a menu item:**

- Double-click the menu item or select Script from the menu item's pop-up menu

  The Script view displays for the clicked event, which is the default event for a menu item.

Using the menu item Clicked event

Typically, your application will contain Clicked scripts for each menu item in a drop-down or cascading menu. For example, the script for the Clicked event for the Open menu item on the File menu opens a file.

The Clicked event is triggered whenever:

- The menu item is tapped with a stylus (or clicked with a mouse on the desktop)
- The shortcut key for the menu item is pressed
- The menu containing the menu item is displayed and the accelerator key is pressed
- A pop-up menu displays

Using the Clicked event on Windows CE platforms
The Windows CE platform does not allow the Clicked event on top-level menu items. For example, script that you place in the Clicked event for the File menu is not triggered on the Windows CE platform, although the same script is triggered in the Clicked event for the File>Open menu item.

The Clicked event for a menu item can be triggered only if both its Visible and Enabled properties are set to true.

If the menu item has a drop-down or cascading menu under it, the script for its Clicked event (if any) is executed when a user taps the menu item with a stylus. The drop-down or cascading menu displays after the Clicked event is triggered. If the menu item does not have a menu under it, the script for the Clicked event is executed when the stylus is lifted.

Using the Clicked event to specify menu item properties
When the user clicks an item on the menu bar to display a drop-down menu, the Clicked event for the menu item on the menu bar is triggered and then the drop-down menu is displayed.

You can use the menu bar's Clicked event to specify the properties of the menu items in the drop-down menu. For example, if you want to disable items in a drop-down menu, you can disable them in the script for the Clicked event for the menu item in the menu bar.

Using functions and variables

You can use functions and variables in your scripts.

Using functions

PocketBuilder provides built-in functions that act on menu items. You can use these functions in scripts to manipulate menu items during execution. For example, to hide a menu, you can use the built-in Hide function.

For a complete list of the menu-level built-in functions, look at the Function List view or use the Browser.

Defining menu-level functions
You can define your own menu-level functions to make it easier to manipulate your menus. One way to do this is by selecting Add from the pop-up menu in the Function List view.

For more information, see Chapter 7, "Working with User-Defined Functions."

Using variables

Scripts for menu items have access to all global variables declared for the application. You can also declare local variables, which are accessible only in the script where they are declared.

You can declare instance variables for the menu when you have data that needs to be accessible to scripts in several menu items in a menu. Instance variables are accessible to all menu items in the menu.

For a complete description of variables and how to declare them, see the *PowerScript Reference* in the online Help.

Defining menu-level structures

If you need to manipulate a collection of related variables, you can define menu-level structures using the Structure view. You can do this by displaying the Structure List view and then selecting Add from the pop-up menu. The Structure and Structure List views are not part of the default layout.

For more information, see Chapter 9, "Working with Structures."

Referring to objects in your application

You can refer to any object in the application in scripts for menu items. You must fully qualify the reference, using the object name, as follows.

Referring to windows

When referring to a window, you simply name the window. When referring to a property in a window, you must always qualify the property with the window's name:

window.property

For example, this statement moves the window w_cust from within a menu item script:

```
w_cust.Move(300, 300)
```

Use with caution in Windows CE applications
It is not standard practice to move main windows in Windows CE applications. This can lead to inadvertent hiding of an application that continues to run in the background. Also, use caution in modifying a main window by changing its height or width, as described in another example in this section. Moving or resizing main windows is likely to confuse to users of your applications.

This statement minimizes w_cust, but on the desktop only. WindowState is not a recognized property on the Windows CE platform:

```
w_cust.WindowState = Minimized!
```

You can use the reserved word ParentWindow to refer to the window that the menu is associated with during execution. For example, the following statement closes the window the menu is associated with:

```
Close(ParentWindow)
```

You can also use ParentWindow to refer to properties of the window a menu is associated with, but not to refer to properties of controls or user objects in the window.

For example, the following statement is valid, because it refers to properties of the window itself:

```
ParentWindow.Height = ParentWindow.Height/2
```

But the following statement is invalid, because it refers to a control in the window:

```
ParentWindow.sle_result.Text = "Statement invalid"
```

Referring to controls and user objects in windows

When referring to a control or user object, you must always qualify the control or user object with the name of the window:

window.control.property

window.userobject.property

For example, this statement enables a CommandButton in window w_cust from a menu item script:

```
w_cust.cb_print.Enabled = TRUE
```

Referring to menu items

When referring to a menu item, use this syntax:

menu.menu_item

menu.menu_item.property

Reference within the same menu
When referring to a menu item within the same menu, you do not have to qualify the reference with the menu name.

When referring to a menu item in a drop-down or cascading menu, you must specify each menu item on the path to the menu item you are referencing, separating the names with periods.

For example, to place a check mark next to the menu item m_bold, which is on a drop-down menu under m_text in the menu saved in the library as m_menu, use this statement:

```
m_menu.m_text.m_bold.Check( )
```

If the previous script is for a menu item in the same menu (m_menu), you do not need to qualify the menu item with the name of the menu:

```
m_text.m_bold.Check( )
```

Using inheritance to build a menu

When you build a menu that inherits its style, events, functions, structures, variables, and scripts from an existing menu, you save coding time. All you have to do is modify the descendent object to meet the requirements of the current situation.

❖ **To use inheritance to build a descendent menu:**

1 Click the Inherit button on the PowerBar.

2 In the Inherit From Object dialog box, select Menus from the Object Type drop-down list, the library or libraries you want to look in, and the menu you want to use to create the descendant, and click OK.

Displaying menus from many libraries
To find a menu more easily, you can select more than one library in the Application Libraries list. Use Ctrl+Click to toggle selected libraries and Shift+Click to select a range.

The selected menu displays in the WYSIWYG Menu view and the Tree Menu view in the Menu painter. The title in the painter's title bar indicates that the menu is a descendant.

3 Make the changes you want to the descendent menu as described in "Modifying an inherited menu" next.

4 Save the menu under a new name.

Modifying an inherited menu

When you build and save a menu, PocketBuilder treats the menu as a unit that includes:

- All menu items and their scripts
- Any variables, functions, and structures declared for the menu

When you use inheritance to build a menu, everything in the ancestor menu is inherited in all of its descendants.

What you can do

In a descendent menu, you can do the following:

- Add menu items to the end of a menu
- Insert menu items in a menu (with some restrictions)

  For more information, see "Where you can insert menu items in a descendent menu" on page 306.
- Modify existing menu items

  For example, you can change the text displayed for a menu item or change its initial appearance by, for example, disabling it or making it invisible.
- Build scripts for menu items that do not have scripts in the ancestor menu
- Extend or override inherited scripts
- Declare functions, structures, and variables for the menu

What you cannot do

You cannot do the following in a descendent menu:

- Change the order of inherited menu items
- Delete an inherited menu item
- Insert menu items between inherited menu items that do not have the ShiftToRight property set (see "Modifying the ShiftToRight property" on page 305)
- Change the name of an inherited menu item
- Change the type of an inherited menu item

Hiding a menu item
If you do not need a menu item in a descendent menu, you can hide it by clearing the Visible property in the Properties view or by using the Hide function.

About menu item names in a descendant

PocketBuilder uses the following syntax to show names of inherited menu items:

```
AncestorMenuName::MenuItemName
```

For example, in a menu inherited from m_update_file, you see m_update_file::m_file for the m_file menu item, which is defined in m_update_file.

The inherited menu item name is also locked, so you cannot change it.

Understanding inheritance

The issues concerning inheritance with menus are similar to the issues concerning inheritance with windows and user objects. For information, see Chapter 12, "Understanding Inheritance."

Inserting menu items in a descendent menu

Modifying the ShiftToRight property

When defining a descendent menu, you might want to insert menu items in the middle of the menu bar or in the middle of a drop-down or cascading menu. To do this, you set the ShiftToRight property in a menu item's Properties view on the General properties page.

If the ancestor menu has no menu items with ShiftToRight set, you can add a new menu item to the end of the descendent menu. To add new menu items elsewhere in the menu, set the ShiftToRight property for the descendent menu items that will follow the new menu item.

The ShiftToRight property is used for menu items on the menu bar (where items need to shift right if a new item is inserted) and for menu items in a drop-down or cascading menu (where items might need to shift down if a new item is inserted). The property name is ShiftToRight, but in drop-down or cascading menus, it means shift down.

Where you can insert menu items in a descendent menu

In a descendent menu, a group of menu items can be one of four types. Each type has an insertion rule.

Table 13-3: Insertion rules for groups of menu items

Type of group	Insertion rule
Inherited menu items without ShiftToRight set	You cannot insert a new menu item before any of these menu items
Inherited menu items with ShiftToRight set in ancestor	You can insert before the first menu item in the group but not before the others
New items without ShiftToRight set	You can insert a new menu item before any of these menu items
New items with ShiftToRight set	You can insert a new menu item before any of these menu items

Where you set the ShiftToRight property

You set the ShiftToRight property in an ancestor menu only if you know that you will always want a group of menu items to shift right (or down) when you inherit from the menu and add a new menu item. For example, if you have New, Edit, and Tools menus on the menu bar, set the ShiftToRight property for the Tools menu items if you are going to inherit from this menu, because Tools is usually the last item on a menu bar.

How to insert menu items in a descendent menu

If you can insert a menu item in a descendent menu, the Insert Menu Item option on the Insert menu and the pop-up menu is enabled. The Insert Menu Item is enabled if ShiftToRight is set in the selected item before which you are inserting, and in all menu items following it.

To insert a menu item in a descendant, you use the same method you use to insert an item in a new menu, whether the menu item is on the menu bar or on a drop-down or cascading menu. For information about inserting menu items, see “Working with menu items” on page 289.

Using menus

You can use menus in two ways:

- Place them in the menu bar of a window
- Display a menu as a pop-up menu

Adding a menu bar to a window

To have a menu bar display when a window is opened by a user, you associate a menu with the window in the Window painter.

❖ **To associate a menu with a window:**

1 Click the Open button in the PowerBar, select the window with which you want to associate the menu, and open the window.

2 Do one of the following:

- In the Properties view for the window, enter the name of the menu in the MenuName text box on the General tab page
- Click the Browse button and select the menu from the Select Object dialog box, which lists all menus available to the application

  In the Select Object dialog box, you can search for a menu by clicking the Browse button.

3 Click Save to associate the selected menu with the window.

Identifying menu items in window scripts

You reference menu items in scripts in windows and controls using the following syntax:

menu.menu_item

You must always fully qualify the menu item with the name of the menu.

When referring to a menu item in a drop-down or cascading menu, you must specify each menu item on the path to the menu item you are referencing, separating the names with periods.

For example, to refer to the Enabled property of menu item m_open, which is under the menu bar item m_file in the menu saved in the library as m_menu, use:

```
m_menu.m_file.m_open.Enabled
```

Changing a window's menu during execution

You can use the ChangeMenu function in a script to change the menu associated with a window during execution.

Displaying pop-up menus

To display a pop-up menu in a window, use the PopMenu function to identify the menu and the location at which you want to display the menu.

If the menu is associated with the window

If the menu is currently associated with the window, you can simply call the PopMenu function.

The following statement in a CommandButton script displays m_appl.m_help as a pop-up menu at the current pointer position, assuming menu m_appl is already associated with the window:

```
m_appl.m_help.PopMenu(PointerX(), PointerY())
```

If the menu is not associated with the window

If the menu is not already associated with the window, you must create an instance of the menu before you can display it as a pop-up menu.

The following statements create an instance of the menu m_new, then pop up the menu mymenu.m_file at the pointer location, assuming m_new is not associated with the window containing the script:

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

CHAPTER 14

Working with User Objects

About this chapter

One of the features of object-oriented programming is reusability: you define a component once, then reuse it as many times as you need to without any additional work. One of the best ways to get reusability in PocketBuilder is with user objects. This chapter describes how to define and use user objects.

Contents

About user objects

Applications often have features in common. For example, you might often reuse features like the following:

- A processing package that calculates commissions or performs statistical analysis
- A Close button that performs a certain set of operations and then closes the window
- DataWindow controls that perform standard error checking
- A list that lists all departments
- A predefined file viewer that you plug into a window

If you find yourself using the same application feature repeatedly, you should define a user object: you define the user object once in the User Object painter and use it as many times as you need.

There are two main types of user objects: class and visual. Class user objects are also called nonvisual objects.

Class user objects

A class user object lets you reuse a set of business rules or other processing that acts as a unit but has no visual component. For example, you might define a class that calculates sales commissions or performs statistical analysis. Whenever you need to do this type of processing, you instantiate the user object in a script and call its functions.

You build class user objects in the User Object painter, specifying instance variables and object-level functions. Then you create an instance of the class user object in your application, thereby making its processing available.

There are two kinds of class user objects:

- Custom class
- Standard class

Custom class user objects

Custom class user objects are objects of your own design that encapsulate properties and functions not visible to the user. They are not derived from PocketBuilder objects. You define them to create units of processing that have no visual component.

For example, to calculate commissions in an application, you can define an n_CalculateCommission custom class user object that contains properties and user-defined functions that do the processing to calculate commissions.

Whenever you need to use this processing, you create an instance of the user object in a script, which then has access to the logic in the user object.

Standard class user objects

A standard class user object inherits its definition from one built-in, nonvisual PocketBuilder object, such as the POOM object, the Transaction object, or the Error object. You modify the definition to make the object specific to your application, and optionally add instance variables and functions to enhance the behavior of the built-in object. Once you define a standard class user object, you can go to the Application painter and specify that you want to use it instead of the corresponding built-in system object in your application.

Visual user objects

A visual user object is a reusable control or set of controls that has a certain behavior. You define it in the User Object painter, where you place controls in the user object and write scripts for those controls. Then you can place the user object as often as needed in windows you build in your applications.

The types of visual user objects are:

- **Custom visual** Most useful if you frequently group controls together in a window and always use the controls to perform the same processing.
- **External visual** Useful when you have a custom DLL.
- **Standard visual** Most useful if you frequently use a PocketBuilder control to perform the same processing.

Custom visual user objects

Custom visual user objects are objects that have several controls that function as a unit. You can think of a custom visual user object as a window that is a single unit and is used as a control.

Suppose that you frequently use a group of buttons, each of which performs standard processing. If you build a custom user object that contains all the buttons, you can place the buttons in the window as a unit when you place the user object in a window.

External visual user objects

External visual user objects contain controls from objects in the underlying windowing system that were created outside PocketBuilder. You can use a custom DLL in PocketBuilder to create an external user object.

You must know what classes the DLL supports, the messages or events the DLL responds to, and the style bits that you can set in the DLL.

Standard visual user objects

A standard visual user object inherits its definition from one standard PocketBuilder control. You modify the definition to make the control specific to your applications.

Suppose that you frequently use a CommandButton named Close to display a message box and close the parent window. If you build a standard visual user object that derives from a CommandButton to perform this processing, you can use the user object whenever you want to display a message box and then close a window.

Building user objects

You can build a user object from scratch, or you can create a user object that inherits its style, events, functions, structures, variables, and scripts from an existing user object.

For information on building a user object from scratch, see “Building a new user object” on page 313. To find out more about creating a user object based on an existing PocketBuilder object, see “Using inheritance to build user objects” on page 321.

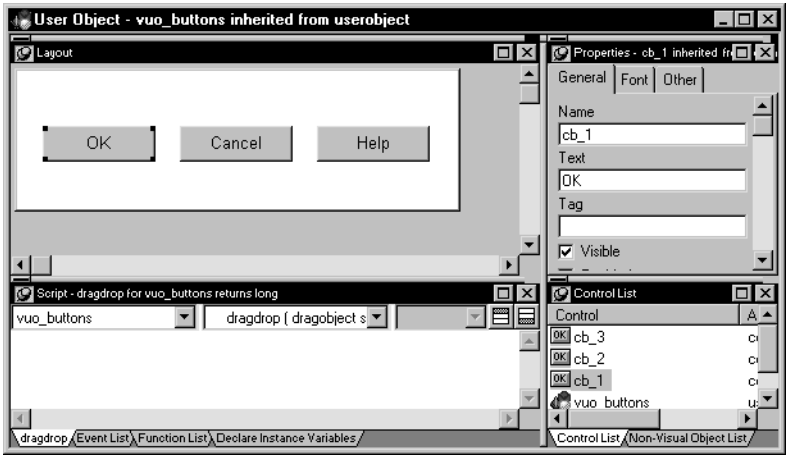
About the User Object painter

The User Object painter has different implementations, depending on the type of user object you are working with. It has several views where you specify how the user object behaves and, for custom visual and standard visual user objects, how it looks. For details about the views, how you use them, and how they are related, see “Views in painters that edit objects” on page 58.

Views for visual user objects

In this User Object painter for a custom visual user object, the Layout view and Script view have been arranged to display at the same time.

Figure 14-1: Custom visual object in the User Object painter



Most of your work in the User Object painter for visual objects is done in three views:

- The Layout view, where you design the appearance of the user object

- The Properties view, where you set user object properties and control properties
- The Script view, where you modify behavior by coding user object and control scripts

In the Layout view, you add controls to a visual user object in the same way you add controls to a window.

For information about specifying user object properties, see "Building a new user object" on page 313. For information about using the Script view, see Chapter 6, "Writing Scripts."

Views for nonvisual user objects

The Layout and Control List view are not needed for nonvisual user objects, but you use all the other views that you use for visual objects.

For nonvisual user objects, there is no layout design work to do, but otherwise, working in the User Object painter on the behavior of a nonvisual object is similar to working on the behavior of a visual user object.

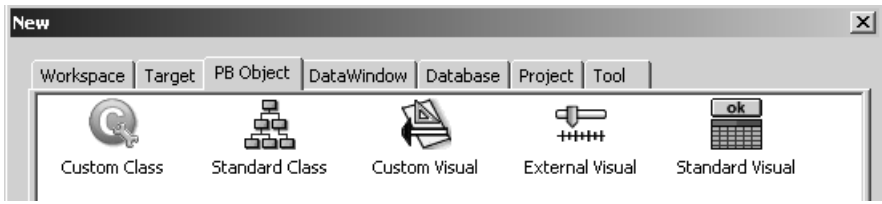
Building a new user object

This section describes how to build a user object from scratch.You use this technique to create user objects that are not based on existing user objects.

Creating a new user object

You can create any type of user object from the PB Object tab page of the New dialog box.

Figure 14-2: User object types on the PBObject tab page



❖ **To create a new user object:**

1 Open the New dialog box.

2 On the PB Object tab page, select the kind of user object you want to create.

The four user object choices display at the top of the tab page.

3 Click OK.

What you do next depends on the type of user object you selected. For custom user objects, the User Object painter opens. For standard user objects, a selection dialog box displays.

The remainder of this section describes how to build each type of user object.

Building a custom class user object

On the PB Object tab page of the New dialog box, if you select Custom Class and click OK, the User Object painter for custom class user objects opens.

❖ **To build the custom class user object:**

1 Declare functions, structures, or variables you need for the user object.

2 Create and compile scripts for the user object.

Custom class user objects have built-in constructor and destructor events.

3 Save the user object.

See “Saving a user object” on page 320.

Using AutoInstantiate

You can create custom class user objects that are autoinstantiated, which provides you with the ability to define methods.

Autoinstantiated user objects do not require explicit CREATE or DESTROY statements when you use them. They are instantiated when you call them in a script and destroyed automatically.

❖ **To define an autoinstantiated custom class user object:**

- In the Properties view, select the AutoInstantiate check box

For more information about autoinstantiation, see “autoinstantiate” in the online Help.

Building a standard class user object

On the PB Object tab page of the New dialog box, if you select Standard Class and click OK, the Select Standard Class Type dialog box displays.

❖ **To build the standard class user object:**

1 In the Select Standard Class Type dialog box, select the built-in system object that you want your user object to inherit from and click OK.

2 Declare functions, structures, or variables you need for the user object.

For a list of properties and functions
Use the Browser to list the built-in properties inherited from the selected system object. Use the Function List view or the Browser to list the functions inherited from the selected system object.

3 Declare any user events needed for the user object.

For information about user events, see "Communicating between a window and a user object" on page 327.

4 In the Script view, create and compile scripts for the user object.

Class user objects have built-in constructor and destructor events.

5 Save the user object.

See "Saving a user object" on page 320.

Building a custom visual user object

On the PB Object tab page of the New dialog box, if you select Custom Visual and click OK, the User Object painter for custom visual user objects opens. It looks like the Window painter, but the empty box that displays in the Layout view is the new custom visual user object.

Building a custom visual user object is similar to building a window, described in Chapter 10, "Working with Windows." The views available in the Window painter and the User Object painter for custom visual user objects are the same.

❖ **To build the custom visual user object:**

1 Place the controls you want in the custom visual user object.

2 Work with the custom visual user object as you would with a window in the Window painter:

- Define the properties of the controls
- Declare functions, structures, or variables as necessary
- Declare any events needed for the user object or its controls

  For information about user events, see “Communicating between a window and a user object” on page 327.
- In the Script view, create and compile the scripts for the user object or its controls

  You can write scripts for each control in a custom visual user object.

  For more information on events associated with custom visual user objects, see “Events in user objects” on page 319.

3 Save the user object.

  See “Saving a user object” on page 320.

Building an external visual user object

On the PB Object tab page of the New dialog box, if you select External Visual and click OK, the User Object painter for external visual user objects opens.

❖ **To build an external visual user object:**

1 In the Properties view, click the Browse button next to the LibraryName box.

2 In the Select Custom Control DLL dialog box, select the DLL that defines the user object and click OK.

3 In the Properties view, enter the following information, as necessary, and click OK:

- The class name registered in the DLL

  Information about the class name is usually provided by the vendor of the purchased DLL.
- Text in the Text box

  This will be displayed only if the object has a text style property.

- Display properties (border and scrollbars)
- Decimal values for the style bits associated with the class

  Information about style bits is usually provided by the vendor of the purchased DLL. PocketBuilder will OR these values with the values selected in the display properties for the control.

4 Declare any functions, structures, or variables you need to declare for the user object.

  You can declare functions, structures, and variables for the user object in the Script view. Information about functions is usually provided by the vendor of the purchased DLL.

5 Declare any needed events for the user object.

  For information about user events, see "Communicating between a window and a user object" on page 327.

6 In the Script view, create and compile the scripts for the user object.

  For more information on events associated with external visual user objects, see "Events in user objects" on page 319.

7 Save the user object.

  See "Saving a user object" on page 320.

Building a standard visual user object

On the PB Object tab page of the New dialog box, if you select Standard Visual and click OK, the Select Standard Visual Type dialog box displays.

❖ **To build a standard visual user object:**

1 In the Select Standard Visual Type dialog box, select the PocketBuilder control you want to use to build your standard visual user object, and click OK.

  The selected control displays in the workspace. Your visual user object will have the properties and events associated with the PocketBuilder control you are modifying.

2 Work with the control as you do in the Window painter:

- Review the default properties and make any necessary changes

- Declare functions, structures, or variables as necessary

  You can declare these in the Script view.

- Declare any user events needed for the user object

  For information about user events, see "Communicating between a window and a user object" on page 327.

- Create and compile the scripts for the user object

  Standard visual user objects have the same events as the PocketBuilder control you modified to create the object.

3 Save the user object.

  See "Saving a user object" on page 320.

Using MOP views for custom visual user objects

The User Object painter includes a MOPView Manager that lets you assign runtime layouts which change automatically when an application user changes the orientation of a handheld device or emulator. You can use the View>Runtime MOP Views>MOPViewManager menu item in the User Object or DataWindow painter to create multiple views for a custom visual user object that have the same orientations available to PocketBuilder windows.

For more information on the available views, see "Multiple Orientation Painter for windows" on page 211.

When you open a window containing a custom visual user object at runtime, PocketBuilder automatically selects the view of the user object that correlates with the current orientation of the device or emulator. If you do not create a specific view for the selected orientation, the last view saved in the painter is displayed at runtime. In this case, the custom user object does not necessarily display with optimal size and position for the user-selected orientation.

Property specificity

For a user object with multiple views, the position (x and y) and dimension (width and height) properties of the user object and any visual controls on the user object are view-specific. All other properties of the user object and its controls apply to all of the views defined for the user object.

The source code in the PKL file includes only the last position and dimension properties saved in the painter for the user object and the controls on the user object. Therefore, when you export a user object with MOP views, the MOP views are not included in the exported code. The exported code contains only the most recent position and dimension properties from the original user object.

However, when you copy or inherit from a user object, the copied or inherited user object retains all the MOP views set in the original user object. You can also move a user object from one PKL to another without losing its MOP views.

Events in user objects

When you build a user object, you can write scripts for any event associated with that user object.

Events in class user objects

Most custom class user objects have only constructor and destructor events.

Table 14-1: Events for custom class user objects

Event	Occurs when
Constructor	The user object is created
Destructor	The user object is destroyed

Standard class user objects have the same events as the PocketBuilder system object from which they inherit.

Events in visual user objects

Standard visual user objects have the same events as the PocketBuilder control from which they inherit. Custom and external visual user objects have a common set of events.

Table 14-2: Events for custom and external visual user objects

Event	Occurs when
Constructor	Immediately before the Open event of the window and when the user object is dynamically placed in a window
Destructor	Immediately after the Close event of the window and when the user object is dynamically removed from a window
DragDrop	A dragged object is dropped on the user object
DragEnter	A dragged object enters the user object
DragLeave	A dragged object leaves the user object
DragWithin	A dragged object is moved within the user object

Event	Occurs when
Other	A Windows message occurs that is not a PocketBuilder event
RButtonDown	The right mouse button is pressed (desktop), or a tap-and-hold action is executed on the control (Pocket PC device)

For more about drag and drop events, see the chapter in the *Resource Guide* on using drag and drop in a window.

Saving a user object

When you save an object, you must give it a name.

Naming conventions

You should adopt naming conventions to make it easy to understand a user object's type and purpose. A user object name can be any valid PowerBuilder identifier of up to 40 characters. For information about valid identifiers, see "identifier names" in the online Help.

One convention you could follow is the use of u_ as the prefix for visual user objects and n_ as the prefix for class (nonvisual) user objects. For standard classes, include in the name the standard prefix for the object or control from which the class inherits. For external user objects, include ex_ in the name, and for custom class user objects, include cst_ in the name.

Table 14-3 shows some examples of this convention.

Table 14-3: Suggested naming conventions for user objects

Type of user object	Format	Example
Standard visual	u_*control_purpose*	u_cb_close, a CommandButton that closes a window
Custom visual	u_*purpose*	u_toolbar, a toolbar
External visual	u_ex_*purpose*	u_ex_sound, outputs sound
Standard class	n_*systemobject_purpose*	n_trans_test, derived from the Transaction object and used for testing
Custom class	n_cst_*purpose*	n_cst_commission, calculates commissions

❖ **To save a user object:**

1 In the User Object painter, select File>Save from the menu bar or click the Save button in the painter bar.

If you have previously saved the user object, PocketBuilder saves the new version in the same library and returns you to the User Object painter.

If you have not previously saved the user object, PocketBuilder displays the Save User Object dialog box.

2 Enter a name in the User Objects box.

To determine a useful name for the user object, see "Naming conventions" on page 320.

3 Enter comments to describe the user object.

These display in the Select User Object dialog box and in the Library painter, and document the purpose of the user object.

4 Specify the library in which to save the user object.

To make a user object available to all applications, save it in a common library and include the library in the library search path for each application.

5 Click OK to save the user object.

Using inheritance to build user objects

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

For example, suppose your application has a user object u_file_view that has three CommandButtons:

- List—displays a list of files in a list
- Open—opens the selected file and displays the file in a MultiLineEdit control
- Close—displays a message box and then closes the window

If you want to build another user object that is exactly like the existing u_file_view except that it has a fourth CommandButton, you can use inheritance to build the new user object, and then all you have to do is add the fourth CommandButton.

❖ **To use inheritance to build a descendent user object:**

1. Click the Inherit button in the PowerBar, or select File>Inherit from the menu bar.
2. In the Inherit From Object dialog box, select User Objects from the Objects of Type drop-down list.
3. Select the target as well as the library or libraries you want to look in.

Displaying user objects from many libraries
To find a user object more easily, you can select more than one library in the Libraries list. Use Ctrl+Click to toggle selected libraries and Shift+Click to select a range.

4. Select the user object you want to use to create the descendant, and click OK.

   The selected object displays in the User Object painter and the title bar indicates that the object is a descendant.
5. Make any changes you want to the user object.
6. Save the user object with a new name.

Using the inherited information

When you build and save a user object, PocketBuilder treats the object as a unit that includes:

- The object (and any controls within the object if it is a custom visual user object)
- The object's properties, events, and scripts
- Any variables, functions, or structures declared for the object

When you use inheritance to build a new user object, everything in the ancestor user object is inherited in the direct descendant, and in its descendants in turn.

Ancestor's instance variables display

Suppose that you create a user object by inheriting it from a custom class or standard class user object that has public or protected instance variables with simple datatypes. In this case, the instance variables display and can be modified in the descendent user object's Properties view.

All public instance variables with simple datatypes, such as integer, boolean, character, date, string, and so on, display in the descendant. Instance variables with the any or blob datatype or instance variables that are objects or arrays do not display.

What you can do in the descendant

You can do the following in a descendent user object:

- Change the values of the properties and the variables
- Build scripts for events that do not have scripts in the ancestor
- Extend or override the inherited scripts
- Add controls (in custom visual user objects)
- Reference the ancestor's functions and events
- Reference the ancestor's structures if the ancestor contains a public or protected instance variable of the structure data type
- Access ancestor properties, such as instance variables, if the scope of the property is public or protected
- Declare variables, events, functions, and structures for the descendant

What you cannot do in the descendant

In a descendent user object, you cannot delete controls inherited from a custom visual user object. If you do not need a control in a descendent user object, you can make it invisible.

Understanding inheritance

The issues concerning inheritance with user objects are the same as the issues concerning inheritance with windows and menus. See Chapter 12, "Understanding Inheritance," for more information.

Using user objects

Once you have built a user object, you are ready to use it in an application. This section describes how to use:

- Visual user objects
- Class user objects

Using visual user objects

You use visual user objects by placing them in a window or in a custom visual user object. The techniques are similar whether you are working in the Window painter or the User Object painter.

❖ **To place a user object:**

1 Open the window or custom visual user object in which you want to place the visual user object.

2 Click the User Object button in the PainterBar, or select Insert>Control from the menu bar and then select User Object.

3 Select the user object you want to use and click the location where you want the user object to display.

PocketBuilder creates a descendent user object that inherits its definition from the selected user object and places it in the window or user object.

What you can do

After you place a user object in a window or a custom visual user object, you can name it, size it, position it, write scripts for it, and do anything else you can do with a control.

When you place the user object in a window, PocketBuilder assigns it a unique name, just as it does when you place a control. The name is a concatenation of the default prefix for a user object control (initially, uo_) and a default suffix, which is a number that makes the name unique.

You should change the default suffix to a suffix that has meaning for the user object in your application.

For more information about naming, see "Naming controls" on page 230.

Writing scripts

When you place a user object in a window or a custom user object, you are actually creating a descendant of the user object. All scripts defined for the ancestor user object are inherited. You can choose to override or extend those scripts.

For more information, see "Using inherited scripts" on page 280.

You place a user object *as a unit* in a window (or another user object). You cannot write scripts for individual controls in a custom user object after placing it in a window or custom user object; you do that only when you are defining the user object itself.

Placing a user object during execution

You can add a user object to a window during execution using the PowerScript functions OpenUserObject and OpenUserObjectWithParm in a script. You can remove a user object from a window using the CloseUserObject function.

Using class user objects

How you insert a nonvisual object

There are two ways to use a class user object when the user object is not autoinstantiating: you can create an instance of it in a script, or you can insert the user object in a window or user object using the Insert menu.

For more information on autoinstantiation, see "Using AutoInstantiate" on page 314.

The nonvisual object you insert can be a custom class user object or a standard class user object of most types.

❖ **To instantiate a class user object:**

1 In the window or user object in which you want to use the class user object, declare a variable of the user object type and create an instance of it using the CREATE statement. For example:

```
// declared instance variable:
// n_myobject invo_myobject
invo_myobject = CREATE n_myobject
```

2 Use the user object's properties and functions to do the processing you want.

3 When you have finished using the user object, destroy it using the DESTROY statement.

If you select Autoinstantiate in the properties of the class user object, you cannot use the CREATE and DESTROY statements.

❖ **To insert a class user object:**

1 Open the window or user object in which you want to insert the class user object.

2 Select Insert>Object from the menu bar.

3 Select User Object (at the bottom of the list) and then select the class user object you want to insert.

PocketBuilder inserts the selected class user object.

4 Modify the properties and code the events of the nonvisual object as needed.

When the user object is created in an application, the nonvisual object it contains is created automatically. When the user object is destroyed, the nonvisual object is destroyed automatically.

Using the Non-Visual Object List view

You can use the same technique to insert standard class user objects. Since all class user objects are nonvisual, you cannot see them, but if you look at the Non-Visual Object List view, you see all the class user objects that exist in your user object.

Using the Non-Visual Object List view's pop-up menu, you can display a class user object's properties in the Properties view, display the Script view for the object to code its behavior, or delete the object.

Using global standard class user objects

Several standard class user object types are inherited from predefined global objects used in all PocketBuilder applications:

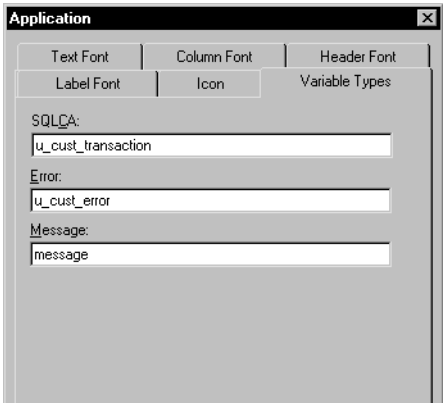
Transaction (SQLCA)
Error
Message

Replacing the built-in global object

If you want your standard class user object to replace the built-in global object, you tell PocketBuilder to use your user object instead of the built-in system object that it inherits from. You will probably use this technique if you build a user object that inherits from the Error or Message object.

In Figure 14-3, user objects have been assigned to replace the default Transaction and Error objects.

Figure 14-3: Replacing default global objects



❖ **To replace the built-in global object with a standard class user object:**

1 Open the Application object.

2 In the Properties view, click the Additional Properties button on the General tab page.

3 In the Application properties dialog box, select the Variable Types tab.

4 Specify the standard class user object you defined in the corresponding field and click OK.

After you have specified your user object as the default global object, it replaces the built-in object and is created automatically when the application starts up. You do not create it (or destroy it) yourself.

The properties and functions defined in the user object are available anywhere in the application. Reference them using dot notation, just as you access those of other PocketBuilder objects such as windows.

Supplementing the built-in global object

You can use a user object inherited from one of the built-in global objects by inserting one in your user object as described in "Using class user objects" on page 325. If you do, your user object is used in addition to the built-in global object variable. Typically you use this technique with user objects inherited from the Transaction object. You now have access to two Transaction objects: the built-in SQLCA and the one you defined.

For more information

For more information about using the Error object, see "Using the Error object" on page 671.

For information about using the Message object, and about creating your own Transaction object, see the *Resource Guide*.

Communicating between a window and a user object

Sometimes you might need to exchange information between a window and a visual user object in the window. Consider these situations:

- You have a set of buttons in a custom user object. Each of the buttons acts upon a file that is listed in a SingleLineEdit control in the window (but not in the user object).

  You need to pass the contents of the SingleLineEdit control from the window to the user object.

- You have a user object color toolbar. When the user clicks one of the colors in the user object, a control in the window changes to that color.

  You need to pass the color from the user object to the window control.

This section discusses two techniques for handling this communication and presents a simple example.

Table 14-4: Techniques for communicating information in a window

Technique	Advantages	Disadvantages
Functions	Easy to use Supports parameters and return types, so is not prone to errors Supports data encapsulation and information hiding Best for complex operations	Creates overhead, might be unnecessary for simple operations
User events	Very flexible and powerful	Uses no type checking, so is prone to error

Communication with both techniques can be either synchronous (using Send for functions and the EVENT keyword for events) or asynchronous (using Post for functions and the POST keyword for events).

Directly referencing properties

Instead of using functions or user events, it is possible to directly reference properties of a user object. If you have a user object control, uo_1, associated with a custom user object that has a SingleLineEdit, sle_1, you can use the following in a script for the window:

```
uo_1.sle_1.Text = "new text"
```

However, it is better to communicate with user objects through functions and user events, as described below, in order to maintain a clean interface between the user object and the rest of your application.

The functions technique

Exchanging information using functions is straightforward. After a user object calls a function, any return value is available to any control within that object.

For how to use this technique, see “Example 1: using functions” on page 330.

❖ **To pass information from a window to a user object:**

1 Define a public, user-object-level function that takes as arguments the information needed from the window.

2 Place the user object in the window.

3 When appropriate, call the function from a script in the window, passing the needed information as arguments.

❖ **To pass information from a user object to a window:**

1 Define a public, window-level function that takes as parameters the information needed from the user object.

2 Place the user object in the window.

3 When appropriate, call the function from a script in the user object, passing the needed information as parameters.

The user events technique

You can define user-defined events, also called user events, to communicate between a window and a user object. You can declare user events for any PocketBuilder object or control.

A custom visual user object often requires a user event. After you place a custom visual user object in a window or in another custom user object, you can write scripts for events that occur only in the user object itself. You cannot write scripts for events in the controls in the user object.

You can, however, define user events for the user object, and trigger those events in scripts for the controls contained in that user object. In the Window painter, you write scripts for the user events, referencing components of the window as needed.

For more information about user events, see Chapter 8, "Working with User Events." For instructions for using this technique, see "Example 2: using user events" on page 331.

❖ **To define and trigger a user event in a visual user object:**

1 In the User Object painter, select the user object.

Make sure no control in the user object is selected.

2 In the Event List view, select Add from the pop-up menu.

3 In the Prototype window that displays, define the user event.

For how to do so, see "Defining user events" on page 183.

4 Use the Event keyword in scripts for a control to trigger the user event in the user object:

userobject.**Event** *eventname* ( )

For example, the following statement in the Clicked event of a CommandButton contained in a custom visual user object triggers the Max_requested event in the user object:

```
Parent.Event Max_requested()
```

This statement uses the pronoun Parent, referring to the custom visual user object itself, to trigger the Max_requested event in that user object.

5 Implement these user events in the Window painter.

❖ **To implement the user event in the window:**

1 Open the window.

2 In the Window painter, select Insert>Control from the menu bar and place the custom visual user object in the window.

3 Double-click the user object, and then in the Script view, write scripts for the user events you defined in the User Object painter.

Examples of user object controls affecting a window

To illustrate these techniques, consider a simple custom visual user object, uo_backcolor, that contains two buttons, Green and Blue.

Figure 14-4: User object with two buttons

If the user clicks the Green button in an application window containing this user object, the current window background turns green. If the user clicks Blue, the window background color changes to blue.

Because the user object can be associated with any window, the scripts for the buttons cannot reference the window that has the user object. The user object must get the name of the window so that the buttons can reference the window.

"Example 1: using functions" next shows how PocketBuilder uses functions to pass a window name to a user object, allowing controls in the user object to affect the window the user object is in.

"Example 2: using user events" on page 331 shows how PocketBuilder uses unmapped user events to allow controls in a user object to affect the window the user object is in.

Example 1: using functions

1 In the Script view in the User Object painter, define an instance variable, *mywin*, of type window.

```
window mywin
```

This variable will hold the name of the window that has the user object.

2 Define a user object-level function, f_setwin, with:

- Public access

- No return value
- One argument, win_param, of type window and passed by value

3 Type the following script for the function:

```
mywin = win_param
```

When f_setwin is called, the window name passed in win_param will be assigned to *mywin*, where user object controls can reference the window that has the user object.

4 Write scripts for the two buttons:

- cb_green: `mywin.BackColor = RGB (64,128,64)`
- cb_blue: `mywin.BackColor = RGB (64,64,255)`

5 Save the user object as uo_backcolor and close the User Object painter.

6 Open the window, select Insert>Control>User Object, select uo_backcolor, and click in the window in the Layout view to add the user object.

7 In the Properties view, change the name for the user object control to uo_func.

8 In the Open event for the window, call the user object-level function, passing the name of the window:

```
uo_func.f_setwin(This)
```

The pronoun This refers to the window's name, which will be passed to the user object's f_setwin function.

What happens When the window opens, it calls the user object-level function f_setwin, which passes the window name to the user object. The user object stores the name in its instance variable *mywin*. When the user clicks a button control in the user object, the control references the window through *mywin*.

Example 2: using user events

1 In the Script view in the User Object painter, define two unmapped user events for the user object: Green_requested and Blue_requested.

Leave the Event ID fields blank to define them as unmapped.

2 Trigger user events of the user object in the scripts for the Clicked event of each CommandButton:

- cb_green: `Parent.Event Green_requested()`
- cb_blue: `Parent.Event Blue_requested()`

3 Save the user object and name it uo_event and close the User Object painter.

4 Open the window and, in the Window painter, select Insert>Object from the menu bar and then place uo_event in the window.

5 Double-click uo_event to display its Script view.

The two new user events display in the second drop-down list in the Script view.

6 Write scripts for the two user events:

- green_requested: `Parent.BackColor = RGB (64,128,64)`
- blue_requested: `Parent.BackColor = RGB (64,64,128)`

These scripts reference the window containing the user object with the pronoun Parent.

What happens When a user clicks a button, the Clicked event script for that button triggers a user event in its parent, the user object. The user object script for that event modifies its parent, the window.

CHAPTER 15

Working with Native Objects and Controls for Windows CE Devices

About this chapter

PocketBuilder includes support for objects and controls that are specific to Windows CE platforms. This chapter describes these objects and controls.

Contents

You can also add a custom Today item for the Start page of a Pocket PC. For information about adding a custom Today item, see "Specifying application and Today item properties" on page 64.

Bar code scanner objects

Description

The IntermecBarcodeScanner, SocketBarcodeScanner, and SymbolBarcodeScanner nonvisual objects inherit from the BarcodeScanner base class. They interface, respectively, with the Intermec, Socket, and Symbol bar code scanners. Table 15-1 displays bar code scanners supported by PocketBuilder.

Table 15-1: Bar code scanners supported by PocketBuilder

Manufacturer	Device types
Intermec	CN2 and 1700 series
Socket Communications	In-Hand Scan Card laser scanner devices
Symbol Technologies	PPT 2800 and 8800 Series Pocket PC terminals

Usage

You can add a bar code scanner object to your PocketBuilder application by selecting the IntermecBarcodeScanner, SocketBarcodeScanner, or SymbolBarcodeScanner menu item from the Insert>Object menu in the window painter. After you add this object to your application, you must call the Open function on the object to load the scanner DLLs and connect to the scanner firmware.

You start a synchronous scan by calling ScanWait, passing in a scan timeout period in seconds. You can run continuous scans by calling the ScanNoWait function from the ScanTriggered event. You retrieve scan data by calling RetrieveData and assigning instance members of the SymbolBarcodeScanner object to variables of the appropriate datatype, or by displaying the values of the instance members in text boxes of an application window or user object.

The following is example code for reading a single bar code scan using a SymbolBarcodeScanner object that is assigned to l_scanner:

```
Integer li_ret
li_ret = l_scanner.Open()
li_ret = l_scanner.ScanWait( 30 )
li_ret = l_scanner.RetrieveData()
sle_symbology.text=string(l_scanner.ScannedSymbology)
sle_data.text = l_scanner.ScannedData
```

For more information

Properties and functions of the BarcodeScanner base class (implemented by the IntermecBarcodeScanner, SocketBarcodeScanner, and SymbolBarcodeScanner objects) are described in the *PowerScript Reference* and in the online Help.

Digital camera objects

Specifying a camera device

The Camera object provides the interface for a PocketBuilder application and a digital camera device. PocketBuilder supports the HP Photosmart, VEO 130S, and HTC cameras.

To capture an image, you must first specify the camera you want to use by selecting the camera type in the Properties view for a Camera object or by setting a specifier for the camera type in script. You must also set the Port or the Folder property on the Camera object before opening a communication channel to the camera device.

Table 15-2 shows integer values that have been defined for supported camera types. It also shows the additional property you must set depending on your camera type selection.

Table 15-2: Specifier and required Port or Folder property setting for supported camera type

Camera type	Specifier	Port or Folder property
VEO 130S	11	Port (set to "SIO1:")
HP Photosmart	71	Port (set to "SIO1:")
HTC using the IA Camera Wizard	81	Folder (set to the path on the Windows CE device)

Camera attributes structure properties

For the HP Photosmart and VEO 130S cameras, the CameraImageAttributes structure object stores the configurations allowed by the camera device for capturing images, and lets you select those attributes when previewing or snapping a photographic image. The CameraImageAttributes object is a read-only structure containing valid configuration settings for a particular camera device. You call GetAllowedImageAttributes to retrieve an array of legal configuration settings for the device.

HTC cameras

For HTC cameras, you must use the IA Camera Wizard to set configurations, and to preview and capture images. After setting the CameraType and Folder properties for the Camera object, then calling Open to open a communication channel to the camera, you can launch the IA Camera Wizard by calling BeginPreview.

Some cameras have different allowable configurations for preview and capture modes. (Some cameras do not even permit preview mode.) For example, the Hewlett-Packard Photosmart Mobile Camera has four different configurations for picture size in capture mode, but only two in preview mode.

Typically, a structure object you use for the preview mode will be different from the structure object for the capture mode of the same device, so there are separate functions to set the configuration for each mode, SetPreviewImageAttributes and SetCaptureImageAttributes.

Camera session example

The following example opens a camera session where cam_1 is an object of type camera. It uses the CameraType property to define the camera as an HP Photosmart device before calling the Open function. The example sets preview and capture configurations, previews the current image in a preview window, and captures the image to a file:

```
integer iRet
CameraImageAttributes   AllowedConfigs[]
cam_1.CameraType = 71
cam_1.Port = "SIO:"
iRet = cam_1.Open(w_myphoto_main)
// Get allowed configurations for this specific camera.
cam_1.GetAllowedImageAttributes(AllowedConfigs[])

// Assume presented to the user, and the user
// makes some selections…
// Assume the user selects the first configuration for
// preview purposes and the third configuration for
// capture purposes
cam_1.SetPreviewImageAttributes &
     (AllowedConfigs[1])
cam_1.SetCaptureImageAttributes &
     (AllowedModes[3])
// set some other options, such as fluorescent
// white balance and center weighting for the
// AE meter
cam_1.SetOption(CamOptWhiteBalance, 3)
cam_1.SetOption(CamOptAEMetering, 1)

// When the user presses a button to begin preview mode,
// preview the image in the picture control "p_preview"
cam_1.BeginPreview( w_main.p_preview )
// When the user presses a button to capture the current
// picture, save the picture to the file "my_pic.jpg"
if cam_1.isReadyToCapture() then
      cam_1.CaptureImage( "\my_pic.jpg" )
end if
// In the application Close event, end the preview and
// close the camera connection
cam_1.EndPreview()
cam_1.Close()
```

HPBiometricScanner object

Description

The HPBiometricScanner nonvisual object inherits from the BiometricScanner base class to interface with the Hewlett-Packard biometric scanner. The scanner software is designed to work with the hp IPAQ h5500 and h5550 Pocket PC terminals that include an integrated biometric fingerprint reader.

Usage

You can add an HPBiometricScanner object to your PocketBuilder application by selecting the Insert>Object>HPBiometricScanner menu item. After you add this object to your application, you must call the Open function on the object to load the scanner DLLs and connect to the scanner firmware.

You start a synchronous scan by calling ScanCapture, passing in a scan timeout period in seconds and an enumerated value for the scan purpose. You can call the VerifyMatch function to compare the scanned minutiae with a template scan stored in a database.

The following is example code for comparing a single fingerprint scan using an HPBiometricScanner object that is assigned to l_scanner:

```
Integer li_ret
Blob lblob_MinutiaeFromDatabase
Blob lblob_MinutiaeFromScan

li_ret = l_scanner.Open()
li_ret = l_scanner.ScanCapture(30, &
   EnrollForVerification!)
sle_quality.text = string(l_scanner.ScannedQuality())
li_ret = &
  l_scanner.ScannedMinutiae(lblob_MinutiaeFromScan)
li_ret = &
   l_scanner.VerifyMatch(lblob_MinutiaeFromScan, &
   lblob_MinutiaeFromDatabase)
```

For more information

Properties and functions of the BiometricScanner base class (implemented by the HPBiometricScanner object) are described in the *PowerScript Reference* and in the online Help.

NotificationBubble object

Description

The NotificationBubble is a Windows CE control that lets you send notifications to a user.

Usage

You can add a NotificationBubble object to a window or user object by using the Insert>Object>NotificationBubble menu item or creating the object in a script. For example, you might add the following PowerScript code in the Script view of a command button Clicked event:

```
String ls_body
NotificationBubble myBubble
Integer li_return

ls_body = "<html><body><form method=~"POST~" action=>"&
+"<p>This is an <font color=~"#0000FF~">"&
+"<b>HTML</b></font> notification coming from "

ls_body += "<font color=~"#FF0000~"><i>Pocket</i>"&
+"PowerBuilder</font></p><p align=right>"&
+"<input type=button name='cmd:10' value="&
+"'My Ok'> <input type=button name='cmd:2'"&
+"value='Cancel'></p></body></html>"

myBubble = CREATE NotificationBubble
myBubble.caption = "My Title"
myBubble.notificationID = 123
myBubble.body = ls_body
myBubble.duration = 10   // seconds
li_return = myBubble.SetMessageSink( parent )
li_return =myBubble.icon( "foo.ico" )
li_return =myBubble.Update()
```

You could code another control to update the message body of the notification bubble with a different message, making sure to create a NotificationBubble with the same NotificationID and to call the Update function again.

Since there is no guarantee that a user will acknowledge a notification and thereby remove it, an application must provide a separate means to remove the notification. Typically you would code the NotificationBubble Remove function in the Close or Destructor event of the visual object that you assigned to receive the notification. The following code would ensure the removal of the NotificationBubble with the NotificationID 123:

```
NotificationBubble myBubble
Integer li_return
myBubble = CREATE NotificationBubble
myBubble.notificationID = 123
li_return = myBubble.Remove()
```

You can add a user event to capture a user response to the notification bubble. The user event must implement the pbm_command event ID. The pbm_command arguments hwndChild and childID are useful for capturing the NotificationID of the NotificationBubble control and the command ID of the HTML input element that acknowledges the notification. You could write these values to a ListBox control, as in the following example:

```
if( hwndChild = 123 ) then
 // display only the messages from the Notification
lb_result.addItem("NotificationID="+string(hwndChild))
lb_result.addItem("cmd:###= " + string(childID))
end if
```

In this example, hwndChild adds the value 123 to the lb_result ListBox. If the notification is acknowledged from an HTML input element with the attribute name = “cmd:10”, as in the example at the beginning of this section, childID would add the value 10 to the ListBox control.

For more information

Properties and functions of a NotificationBubble control are described in the *PowerScript Reference* and in the online Help.

Phone-related objects

The PhoneCall nonvisual object provides an interface that allows a PocketBuilder application user to place or receive phone calls on a Smartphone or Pocket PC-Phone Edition platform. Other nonvisual objects track call history and provide access to dialing directories. Table 15-3 lists nonvisual user objects that support telephone functionality in PocketBuilder.

Table 15-3: Phone-related objects

Object	Provides the interface to
PhoneCall object	The telephone account on a device
CallLog and CallLogEntry objects	Entries in the call log on a device
DialingDirectory and DialingDirectoryEntry objects	Entries in the address books on a device

PhoneCall object

Instantiating the PhoneCall object

At design time, you can add PhoneCall object to a window by selecting Insert>Object>PhoneCall from the PocketBuilder menu. The default name for the first PhoneCall object is pcall_1. If you use the PocketBuilder UI to add the object, PocketBuilder automatically instantiates the object at runtime. If you do not use the PocketBuilder UI to add a PhoneCall object, you must instantiate the object in code (and optimally destroy it after a user finishes placing the call or closes the application).

PhoneCall object example

The following example makes a voice call to a local take-out restaurant after setting some properties on the pcall_1 object:

```
Integer li_ret
//set properties of phone call object
pcall_1.VoiceCall = true
pcall_1.PhoneNumber = "1-617-123-4567"
pcall_1.CalledParty = "pizza order"
//place a phone call
li_ret = pcall_1.MakeCall()
```

CallLog and CallLogEntry objects

Using the call history objects

The CallLog and CallLogEntry objects provide an interface to the entries in the call log on Smartphone and PocketPC - Phone Edition platforms. The call log lists information regarding all incoming and outgoing calls for a device, allowing a user to track and return missed calls, manage phone billing charges, and perform additional tasks. It is a read-only data store.

At design time, you can add a CallLog object by selecting Insert>Object>CallLog from the PocketBuilder menu. If you do not use the PocketBuilder UI to add a CallLog object, you must instantiate a CallLog object in code. You can call the getEntry method on the CallLog object to return a CallLogEntry object.

CallLog example

The following example places information from a single phone call into SingleLineEdit boxes:

```
Integer li_ret
CallLogEntry l_entry
l_entry = clog_1.getEntry(1)
//display call log entry values in text boxes
sle_name.text = l_entry.Name
sle_number.text = l_entry.PhoneNumber
sle_StartTime.text = l_entry.StartTime
sle_EndTime.text = l_entry.EndTime
```

DialingDirectory and DialingDirectoryEntry objects

Using the phone directory objects

The DialingDirectory and DialingDirectoryEntry objects provide an interface to the entries of phone books on Smartphone and PocketPC - Phone Edition platforms. You can use these objects to merge multiple sources of phone numbers into a single logical entity.

At design time, you can add a DialingDirectory object by selecting Insert>Object>DialingDirectory from the PocketBuilder menu. If you do not use the PocketBuilder UI to add a DialingDirectory object, you must instantiate a DialingDirectory object in code.

You can call the getEntry method on the DialingDirectory object to return a DialingDirectoryEntry object:

```
Integer l_idx = 1
DialingDirectoryEntry l_mydirectoryentry
l_mydirectoryentry = l_myphonebook.getEntry (l_idx)
```

For more information

Properties and functions of phone-related objects are described in the *PowerScript Reference* and in the online Help.

POOM object

Description

The Pocket Outlook Object Manager (POOM) object provides an interface to the object store for the Pocket PC contact manager, appointment calendar, and task manager.

The POOM object interacts with other POOM-related objects listed in Table 15-4.

Table 15-4: POOM and POOM-related objects

Object	Description
POOM object	Main POOM object that provides an interface to other POOM-related objects, and lets you add and remove appointments, contacts, and tasks
POOMAppointment object	Gets and sets appointment recipients and recurrences
POOMContact object	Copies and displays contacts
POOMTask object	Copies and displays tasks, and gets and sets task recurrences
POOMRecurrence object	Used by the POOMAppointment and POOMTask objects to define recurring appointments and tasks
POOMRecipient object	Used by the POOMAppointment object to define the recipients of appointment notifications

The POOM object is the root object that lets you access all the other objects in the system. You use it to attach to the Pocket Outlook object manager running on the Pocket PC, to create and delete contacts, appointments, and tasks, and to receive and dispatch these objects (contacts, appointments, and tasks) using infrared queues.

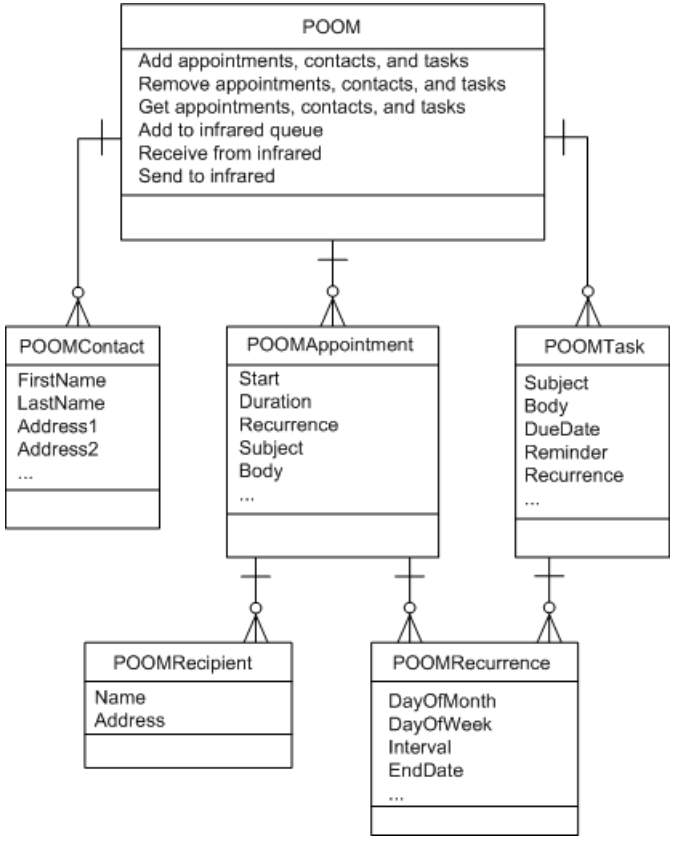
You use the POOMContact, POOMAppointment, and POOMTask objects to specify the properties of objects, such as the e-mail address of a contact or the time of an appointment. All of these objects have functions that display the object on the Pocket PC and update any changes in the Pocket Outlook object manager's repository.

The POOMAppointment and POOMTask objects also have functions that let you specify the details of recurring appointments and tasks, using the properties of the POOMRecurrence object.

POOMAppointment also lets you add and remove the names of recipients of an appointment (the individuals invited to a meeting).

Figure 15-1 displays the hierarchical relationships among the POOM-related objects.

Figure 15-1: Overview of POOM-related objects



Usage

You can add a POOM object to a window or user object by using the Insert>Object>POOM menu item or creating the object in a script.

Creating a POOM object To perform any POOM-related task, such as getting an appointment or creating a task, you must create a POOM object and attach to the Pocket Outlook object manager. You attach to the repository by calling the Login function. To ensure that the POOM object is removed from the device or emulator's memory, call the Logoff function when you have finished working with the object manager.

When you call the Login function, you can optionally specify a window that will serve as the parent window for the Outlook session and will be used when you call the Display function. If you do not specify a window, the Pocket PC Contacts, Calendar, or Tasks window displays.

The following script creates a POOM object, attaches to the Pocket Outlook object manager, retrieves an array of appointments, and then detaches from the object manager:

```
// Global variable: POOM g_poom
POOMAppointment  myAppts[]
POOMTask         myTasks[]
POOMContact      myContacts[]
Integer li_rtn

g_poom = CREATE POOM
li_rtn = g_poom.login()

// Get appointments and display first in edit boxes
li_rtn = g_poom.getAppointments( myAppts )
sle_1.text = myAppts[1].subject
sle_2.text = STRING( myAppts[1].start, "[datetime]" )
sle_3.text = STRING( myAppts[1].end,   "[datetime]" )
sle_4.text = myAppts[1].location
…
g_poom.logoff()
DESTROY g_poom
```

The examples in the rest of this section assume that you have already attached to the Pocket Outlook object manager.

Modifying an item in the POOM repository To modify an existing item, you first obtain the item using one of the Get functions of the POOM object. For POOMAppointment, POOMContact, and POOMTask, the POOM object has four Get functions. For example, for contacts:

- GetContact (*index*) gets a contact using its index in the object manager. It takes an integer and returns a POOMContact object.
- GetContactFromOID (*oid*) gets a contact using its Windows CE object identifier (OID). It takes an unsigned long and returns a POOMContact object.

OIDs are not persistent
You cannot use OIDs across POOM sessions. They are reassigned when the repository is backed up and at other times.

- GetContacts (ref *poomtask[]*) gets an array of all the contacts in the repository and returns an integer.
- GetContacts (*matchcriteria*, *poomtask[]*) gets an array of all the contacts in the repository that satisfy the match criteria and returns an integer.

Checking for valid objects
When you use a function, such as GetContact or GetContactFromOID, that returns an object, use the IsValid function to ensure that a valid object was returned.

The following example changes the second e-mail address of the contact with index 23:

```
POOMContact mycontact

mycontact = g_poom.GetContact(23)
If IsValid(mycontact) then
   mycontact.email2address = "janedoe@netscape.net"
   mycontact.Update()
end if
```

Creating new objects You use the CREATE statement to create new objects. This example creates a new appointment, sets its subject, location, and start and end times, and specifies that the user should be reminded 15 minutes before the start time. The type of reminder is not specified in this example, so the system default will be used. You can specify the type of reminder with the ReminderOptions property.

```
integer li_rc
POOMAppointment appt
DateTime  dt

appt = CREATE POOMAppointment
appt.Subject = "Quick Team Meeting"
appt.Location = "Terry's Office"

// start the meeting 30 minutes from now and
// end it 15 minutes later
dt = datetime(today(), RelativeTime(now(), 30*60) )
appt.appointmentStart = dt
dt = datetime(today(), RelativeTime(now(), 45*60) )
appt.appointmentEnd = dt
// set a reminder
appt.IsReminderSet = true
appt.ReminderMinutesBeforeStart = 15
```

```
// save the appointment in the repository and display it
li_rc = g_poom.Add( appt )

appt.display()POOMAppointment appt
```

You can add detailed notes to the object description only after it has been saved in the repository. The Body property sets a text annotation, and the BodyInk property sets an annotation in Pocket Word Ink (PWI) format. Setting either or both properties automatically updates the object. For example, if you add the following line to the previous example after the call to Add and before the call to Display, the text displays in the calendar:

```
appt.body = "Quick update on status"
```

Cloning an existing object When you modify an object, as described in "Modifying an item in the POOM repository" on page 344, you call the Update function to save your changes to the repository. If you call the Add function instead, a new object is added to the repository with the properties you change and all the properties of the original object. For example, if Mary Smith changes her last name to Smythe, and you want both names to be listed as contacts, you could use the following code, assuming Mary's index in the contact list is 27:

```
POOMContact contact
contact = g_poom.GetContact(27)
If IsValid(contact) then
   contact.LastName = "Smythe"
   g_poom.Add( contact )
end if
```

Sending a cancellation notice You can use the Cancel function on an appointment to send a cancellation notice to the appointment's recipients.

```
appt = g_poom.GetAppointment( 1 )
iRet = appt.Cancel()
```

Removing an existing object You can delete appointments, contacts, and tasks from the repository by calling the Remove function:

```
appt = g_poom.GetAppointment( 1 )
iRet = appt.Remove()
```

Creating and clearing a recurring appointment or task You create a POOMRecurrence object to set recurring properties for a valid appointment or task. The following code causes a task to be entered in the client Outlook calendar for 23 consecutive days:

```
POOMTask task
POOMRecurrence  recur
integer iRet
task = g_poom.GetTask( 1 )// BY INDEX
if isValid(task) then
    // add the recurrence
    recur = CREATE POOMRecurrence
    recur.recurrencetype = RecursDaily!
    recur.Occurrences = 23
    iRet = task.SetRecurrence( recur )
    iRet = task.Update()
end if
```

You clear a recurrence pattern by calling ClearRecurrencePattern on the POOMRecurrence object for the repeating task or appointment:

```
task = g_poom.GetTask( 1 )// BY INDEX
if isValid(task) then
    // clear the recurrence
    iRet = task.ClearRecurrencePattern()
    iRet = task.Update()
end if
```

Adding recipients to an appointment You add a recipient to an appointment by calling AddRecipient on a POOMAppointment object. The AddRecipient function can take string arguments for the name and address of a recipient, or can be passed a POOMRecipient object containing a recipient name and address. You must add recipients to an appointment one by one. However, if you use POOMRecipient objects exclusively to add appointment recipients, you can call GetRecipients on the POOMAppointment object to obtain an array of all the names on the recipient list and all the addresses where appointment notices are sent.

For more information

Properties and functions of the POOM object and its related objects are described in the *PowerScript Reference* and in the online Help.

For more information about the Pocket Outlook Object Model, see the Microsoft Web site at http://msdn2.microsoft.com/en-us/library/aa454225.aspx.

SerialGPS object

Supported hardware

PocketBuilder applications can interface with global positioning system (GPS) devices through nonvisual user objects. The SerialGPS object provides an interface to the Bluetooth unit GPS devices manufactured by Pharos and TomTom, and to Windows Mobile 5 or later devices that use a COM port. The SerialGPS object inherits from a GPS base class.

Windows Mobile 5 devices
You can instantiate the GPS base class, instead of the SerialGPS object, to interface directly with GPS devices using a WM 5 platform. To use the GPS base class, you must set the ConfigParams property to "driver=WMNative".

In addition to the GPS and SerialGPS classes, the PocketBuilder GPS interface uses several structure objects to store information.

For information about the properties, events, and functions of the SerialGPS object, and the properties of the related GPSCoordinate, GPSFix, GPSHeading, GPSSatellitePosition, and GPSSatellitesInView structure objects, see *Objects and Controls* in the online Help.

For more information on the supported GPS devices, see the Pharos Web site at http://www.pharosgps.com and the TomTom Web site at http://www.tomtom.com. For information about the NMEA-0183 specification used by the supported GPS devices, see the NMEA Web site at http://www.nmea.org/.

GPS Example

The following example captures the time, latitude, longitude, and altitude of a GPS reading in SingleLineEdit text boxes:

```
GPSFix myFix
GPSCoordinate myCoord
Integer fixMinutes
Real fixSeconds
Integer rc

gps_1.SerialPort = "COM5:" //override default com8 port
gps_1.ConfigParams = "buffersize=2000,refresh=2000,"&
 + "timeout=5000,multithread=0"
rc = gps_1.Open()
IF rc = 1 THEN
 rc = gps_1.GetFix(myFix)
 IF rc = 1 THEN
 //Data received, test if valid
  IF myFix.isFixValid THEN
   //Display current fix
```

```
      sle_time.text = string(myFix.FixTime, "h:mm:ss") &
         + " UTC"
      myCoord = myFix.Longitude
      fixMinutes = INTEGER (myCoord.minute)
      fixSeconds =(myCoord.minute &
        - INTEGER(myCoord.minute))*60.0
      sle_long.text = "Longitude: " &
        + string(myCoord.degree) + " degrees " &
        + string(fixMinutes) + " minutes " &
        + string(fixSeconds) + " seconds " &
        + string(myCoord.hemisphere)

      myCoord = myFix.Latitude
      fixMinutes = INTEGER (myCoord.minute)
      fixSeconds =(myCoord.minute &
        - INTEGER(myCoord.minute))*60.0
      sle_lat.text = "Latitude: " &
       + string(myCoord.degree) + " degrees " &
       + string(fixMinutes) + " minutes " &
       + string(fixSeconds) + " seconds " &
       + string(myCoord.hemisphere)

      sle_height.text ="Altitude: " + &
       string(myFix.Altitude) + " meters"
     ELSE
      sle_message.text = "Fix is not valid"
     END IF
    ELSE
     //Call user function to display error
     sle_message.text = uf_display_error("GetFix", rc)
    END IF
   ELSE
    sle_message.text = uf_display_error("Open Serial " + &
     "GPS", rc)
   END IF
   rc = gps_1.Close()
```

The previous example uses the user function, uf_display_error, which provides a convenient way to interpret error codes from GPS objects. The following code defines the user function:

```
public function string uf_display_error (string
msg_prefix, long errcode);

stringerrmsg, err_description
err_description = " RC=" + string(errcode)
```

```
choose case errcode
    case 1
       err_description += " Success"
    case 100
       err_description += " End of Buffer"
    case -1
       err_description += " General Error"
    case -10
       err_description += " GPS Object Error"
    case -11
       err_description += " No Raw Data. Set " + &
        "configparams before Open"
    case -12
       err_description += " Open Failure. " + &
        "Check filename argument on Open"
    case -13
       err_description += " Open object before " + &
        "calling other methods."
    case -14
       err_description += " Read timeout. " + &
        "SerialGPS reciever may not be sending data."
    case -15
       err_description += " Read Failure. "
    case -16
       err_description += " Parser Error. " + &
          "Unexpected data recieved."
    case -17
       err_description += " Checksum incorrect. " + &
          "Possible data corruption in this sentence."
    case -100
       err_description += " Method not implemented."
end choose

errmsg = msg_prefix + err_description + EOL
return errmsg

end function
```

For more information

For more information on the supported GPS devices, see the Pharos Web site at http://www.pharosgps.com and the TomTom Web site at http://www.tomtom.com. For information about the NMEA-0183 specification, see the NMEA Web site at http://www.nmea.org/.

Signature control

Description

The Signature control lets you capture a user's signature or a drawing and save it. You can also set data in the control.

Usage

To add a Signature control to a window or user object, use the Insert>Control>Signature menu item, or select the Signature icon on the Controls drop-down toolbar on the PainterBar and click inside the window or user object.

The GetDataAsInk and SetDataAsInk functions get and set blob data, including graphical information such as signature or drawing, in Pocket Word Ink (PWI) format, which is compatible with Pocket Word. You can also save blob data as a bitmap using the GetDataAsBitmap function. The bitmap is compatible with the Picture control and Windows desktop applications.

The GetDataAsRTF and SetDataAsRTF functions get and set data in RTF format in a blob or Unicode string. However, due to a Microsoft limitation, these functions can currently get and set only the text data in the control, such as data typed in using the SIP. They cannot get and set graphical data.

The GetDataAsText and SetDataAsText can also get and set text data in a Unicode string.

After saving the data in the control, use the Clear function to clear all data in the control.

Properties and functions of a Signature control are described in the *PowerScript Reference* and in the online Help.

SMS messaging objects

SMSSession object interface

Nonvisual user objects in PocketBuilder provide support for sending Short Message Service (SMS) messages from applications that you deploy to devices that use the SMS messaging protocol. This includes the "Phone Editions" of Pocket PC devices, as well as all Smartphone platforms. SMS involves the combination of text-based e-mail (although other types of data can be sent) and a paging mechanism.

The SMSSession object provides the interface for a PocketBuilder application and the SMS messaging system on a Pocket PC or Smartphone device. It also provides access to the SMSAddress, SMSMessage, and SMSProtocol objects.

Receiving SMS messages

You can also receive SMS messages in PocketBuilder applications running on Windows Mobile 2003 platforms. PocketBuilder provides a shim DLL, *PKSMS20.DLL*, that contains a COM object that hooks into the SMS processor. The DLL should be copied to the *\Windows* directory of a Pocket PC device or the *\Storage\Windows* directory of a Smartphone device. You must register the DLL with the operating system, then perform a soft reset on the device.

You can deploy the DLL along with registry settings in the CAB file you create for a customer application. The DLL is not part of the standard CAB file that you generate from the PocketBuilder Project painter. You can use the Enhanced CAB Generation tool to add the *PKSMS20.DLL* file to the CAB file, or you can add it manually.

Registering the shim DLL

To register the DLL, you can use the Enhanced CAB Generation tool (see "Support for SMS receiving in generated CAB files" next) or either of the following approaches:

- Call DLLRegisterServer from your PocketBuilder application.
- Modify the INI for the CAB file to add the required registry keys manually.

  First you register the server:

  ```
  [HKEY_CLASSES_ROOT\CLSID\{CA08D891-1E24-4c69-A313-
  453B1120E558}\InProcServer32] @="PKSMS20.dll"
  ```

  ```
  [HKEY_CLASSES_ROOT\CLSID\{CA08D891-1E24-4c69-A313-
  453B1120E558}]ReadOnly = dword: #
  ```

  The ReadOnly value (*#*) can be 0 or 1. If you register the DLL by calling DLLRegisterServer, this value is automatically set to 0 (false).

  When `ReadOnly = dword: 0` and you return true for an SMSSession IncomingMessage event, PocketBuilder attempts to prevent an incoming SMS message from displaying in the SMS inbox. PocketBuilder can fail in this attempt when other phone-related activity interferes with the request for nondisplay of the incoming message. In some configuations, it is possible that the shim DLL will be prevented from operating unless the ReadOnly attribute is set to 1.

  When `ReadOnly = dword: 1`, the application will be notified of incoming SMS messages, but PocketBuilder will not be able to delete the incoming messages from the SMS inbox regardless of what the IncomingMessage event returns.

Next you register the inbox processor:

```
[HKEY_LOCAL_MACHINE\Software\Microsoft\Inbox\Svc\
SMS\Rules]{CA08D891-1E24-4c69-A313-453B1120E558} =
dword:1
```

In order to receive SMS messages in an application, you must set the second argument in the Open call for the SMSSession object to 2 or 3.

Unregistering a DLL
You can unregister the DLL by calling DLLUnregisterServer from your application. You must perform a soft reset after you unregister a DLL.

Support for SMS receiving in generated CAB files

The Other Options tab page in the Enhanced CAB Generation tool has check boxes that let you include the following in a CAB file that you deploy to a device:

- The SMS reception DLL, *PKSMS20.DLL*

  You must select the check box to include PocketBuilder support DLLs. This enables the check box that lets you include the SMS reception DLL.

- Registry entries for SMS reception in a PocketBuilder application

  If you select this option and generate a CAB file with the Enhanced CAB Generation tool, unzipping the CAB file on the deployment device automatically inserts the required entries in the device registry.

- A modification to the `ReadOnly` string of the SMS reception DLL registry entry

  If the check box labeled SMS Reception is Read Only is selected when you generate a CAB file with the Enhanced CAB Generation tool, unzipping the CAB file on the deployment device automatically assigns a `ReadOnly` value of 1. This makes PocketBuilder applications unable to delete incoming messages from the SMS inbox.

You access the Enhanced CAB Generation tool on the Tool tab of the New dialog box.

For more information

Properties and functions of the SMSSession, SMSAddress, SMSMessage, and SMSProtocol objects are described in the *PowerScript Reference* and in the online Help.

Toolbar control

The Toolbar control lets you add picture buttons to a menu for your PocketBuilder applications. You can control picture size in runtime toolbars by setting the PictureWidth and PictureHeight properties on the Toolbar control. The toolbar shrinks or grows to fit the height, and the toolbar buttons shrink or grow to fit the width and the height you set. Pictures you select for the buttons are automatically scaled to the picture width and height settings.

You add a Toolbar control to a window or user object using the Insert>Control>Toolbar menu item, or by selecting the Toolbar icon in PainterBar1 and clicking inside the window or user object.

You can add toolbar items to the Toolbar control on the Items page of the Properties view for the control. You can also create ToolbarItem objects in script:

```
ToolbarItem myItem
Integer li_rtn

li_rtn = tlbr_myToolBar.AddPicture ("pic1.bmp")
li_rtn = tlbr_myToolBar.AddPicture ("addwatch!")
tlbr_myToolBar.visible = true
myitem.itemstate = 4
myitem.itemgroup = 0
myitem.itempictureindex = 1
myitem.itemstyle = stylebutton!
li_rtn = tlbr_mytoolbar.AddItem(myItem)
myitem.itempictureindex = 2

myitem.itemstyle = stylecheck!

li_rtn = tlbr_mytoolbar.AddItem(myItem)
```

Typically you would use the PocketBuilder UI to add a visual user object, because events that are associated with a visual user object are not available when the object is instantiated in script. Since there are no events on a ToolbarItem object, the only disadvantage to using script to add a toolbar item object is one of convenience; however, tapping an item in a Toolbar control triggers a Clicked event on the control. To take advantage of this Toolbar control event, you must create the Toolbar control in the UI, even if you add toolbar items in script.

Properties and functions of the Toolbar control and ToolbarItem objects are described in the *PowerScript Reference* and in the online Help.

PART 5

Working with Databases and DataWindows

This part describes how to use PocketBuilder to manage your database. It also describes how to build DataWindow objects to retrieve, present, and manipulate data in your PocketBuilder applications

CHAPTER 16

Managing the Database

About this chapter

This chapter describes how you can manage the database from within PocketBuilder and database synchronization from within PocketBuilder applications.

Contents

Working with database components

A database is an electronic storage place for data. Databases are designed to ensure that data is valid and consistent, and that it can be accessed, modified, and shared.

A database management system (DBMS) governs the activities of a database and enforces rules that ensure data integrity. A relational DBMS stores and organizes data in tables.

How you work with databases in PocketBuilder

You can use PocketBuilder to work with the following database components:

- Tables and columns
- Keys
- Indexes
- Database views
- Extended attributes
- Additional database components

Tables and columns

A database usually has many tables, each of which contains rows and columns of data. Each row in a table has the same columns, but a column's value for a particular row could be empty or NULL if the column's definition allows it.

Tables often have relationships with other tables. For example, in the ASADemo_10 database, the Department table has a dept_id column, and the Employee table also has a dept_id column that identifies the department in which the employee works. When you work with the Department table and the Employee table, the relationship between them is specified by a join of the two tables.

About the demo database

The ASADemo_10 database is a migrated version of the Adaptive Server Anywhere 9 Sample database that you can use with SQL Anywhere 10. The PocketBuilder setup program installs this database in the *Code Examples\SA10DemoData* directory. You must create an ODBC data source name for this database in the ODBC Administrator before you can use it. Table and column names in this database are different than the names in the SQL Anywhere 10 Demo database that installs with SQL Anywhere.

Keys

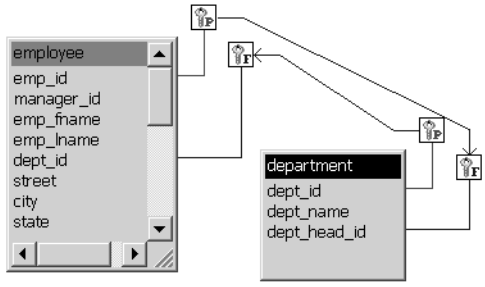
Relational databases use keys to ensure database integrity.

Primary keys A primary key is a column or set of columns that uniquely identifies each row in a table. For example, two employees might have the same first and last names, but they have unique ID numbers. The emp_id column in the Employee table is the primary key column.

Foreign keys A foreign key is a column or set of columns that contains primary key values from another table. For example, the dept_id column is the primary key column in the Department table and a foreign key in the Employee table.

Key icons In PocketBuilder, columns defined as keys are displayed with key icons that include a P for primary or an F for foreign. PocketBuilder automatically joins tables that have a primary/foreign key relationship, with the join on the key columns.

Figure 16-1: Links between tables in Object Layout view



For more information, see "Working with keys" on page 380.

Indexes

An index is a column or set of columns you identify to improve database performance when searching for data specified by the index. You index a column that contains information you will need frequently. Primary and foreign keys are special examples of indexes.

You specify a column or set of columns with unique values as a unique index, represented by an icon with a single key.

You specify a column or set of columns with values that are not unique as a duplicate index, represented by an icon with two keys.

For more information, see "Working with indexes" on page 384.

Database views

If you often select data from the same tables and columns, you can create a database view of the tables. You give the database view a name, and each time you refer to it, the associated SELECT command executes to find the data.

Database views are listed in the Objects view of the Database painter and can be displayed in the Object Layout view, but a database view does not physically exist in the database in the same way that a table does. Only its definition is stored in the database, and the view is re-created whenever the definition is used.

Database administrators often create database views for security purposes. For example, a database view of the Employee table that is available to users who are not in Human Resources might show all columns except Salary.

For more information, see "Working with database views" on page 385.

Extended attributes

Extended attributes enable you to store information about a table's columns in special system tables. Unlike tables, keys, indexes, and database views (which are DBMS-specific), extended attributes are specific to PocketBuilder. The most powerful extended attributes determine the edit style, display format, and validation rules for the column.

For more information about extended attributes, see "Specifying column extended attributes" on page 371. For more information about the extended attribute system tables, see Appendix A, "Extended Attribute System Tables."

Using the Database painter

To open the Database painter, click the Database button in the PowerBar.

About the painter

Like the other PocketBuilder painters, the Database painter contains a menu bar, a customizable PainterBar, and several views. All database-related tasks that you can do in PocketBuilder can be done in the Database painter.

UltraLite limitations
UltraLite databases do not support database owners, groups, stored procedures, views, system tables, and extended attributes, and you cannot modify tables and primary and foreign keys in the database directly. As a result, some of the Database painter views are read only, some tree view items are not present or empty, and some menu items are disabled when you are connected to an UltraLite database. Use the UltraLite Schema Painter, available in the UltraLite Utilities folder in the Database painter's Objects view, to modify tables.

Views in the Database painter

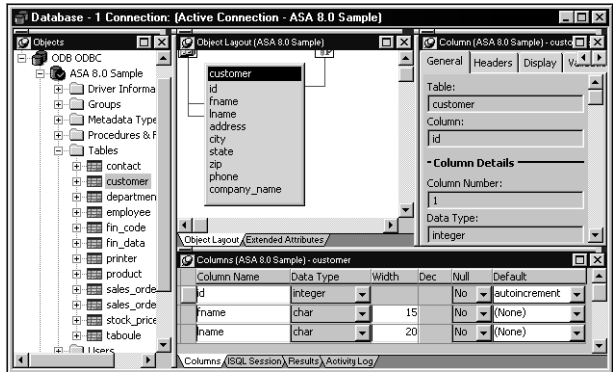
Table 16-1 lists the views available in the Database painter.

Table 16-1: Database painter views

View	Description
Activity Log	Displays the SQL syntax generated by the actions you execute.
Columns	Used to create and/or modify a table's columns. The Columns view is read only for UltraLite databases.
Extended Attributes	Lists the display formats, edit styles, and validation rules defined for the selected database connection. Extended attributes are not supported in UltraLite databases.
Interactive SQL (ISQL Session)	Used to build, execute, or explain SQL.

View	Description
Object Details	Displays an object's properties. For some objects, its properties are read-only; for others, properties can be modified. This view is analogous to the Properties view in other painters.
Object Layout	Displays a graphical representation of tables.
Objects	Lists the database interfaces and profiles. For an active database connection, might also list all or some of the following objects associated with that database: groups, metadata types, procedures and functions, tables, columns, primary and foreign keys, indexes, users, views, driver information, and utilities (the database components listed depend on the database and your user privileges).
Results	Displays data in a grid, table, or freeform format.

Figure 16-2: Default layout of the Database painter



Dragging and dropping

You can select certain database objects from the Objects view and drag them to the Object Details, Object Layout, Columns, and/or ISQL views. Position the pointer on the database object's icon and drag it to the appropriate view.

Table 16-2: Using drag and drop in the Database painter

Object	Can be dragged to
Driver, group, metadata type, procedure or function, table, column, user, primary or foreign key, index	Object Details view
Table or view	Object Layout view
Table or column	Columns view
Procedure or view	ISQL view

Database painter tasks

Table 16-3 describes how to do some basic tasks in the Database painter. Most of these tasks begin in the Objects view. Many can be accomplished by dragging and dropping objects into different views. If you prefer, you can use buttons or menu selections from the main bar or from pop-up menus.

Table 16-3: Common tasks in the Database painter

To do this	Do this
Modify a database profile	Highlight a database profile and select Properties from the Object or pop-up menu, or use the Properties button. You can use the Import and Export Profiles menu selections to copy profiles. For more information, see the chapter on connecting to a database in the *Resource Guide*.
Connect to a database	Highlight a database profile and then select Connect from the File or pop-up menu or use the Connect button. With File>Recent Connections you can review and return to earlier connections. Database connections can also be made using the Database Profile button.
Create new profiles, tables, views, columns, keys, indexes, or groups	Highlight the database object and select New from the Object or pop-up menu or use the Create button.
Modify database objects	Drag the object to the Object Details view.
Graphically display tables	Drag the table icon from the list in the Objects view to the Object Layout view, or highlight the table and select Add To Layout from the Object or pop-up menu.
Manipulate data	Highlight the table and select Grid, Tabular, or Freeform from the Object>Data menu or the pop-up menu Edit Data item, or use the appropriate Data Manipulation button.
Build, execute, or explain SQL	Use the ISQL view to build SQL statements. Use the Paste SQL button to paste SELECT, INSERT, UPDATE, and DELETE statements, or type them directly into the view's scripting area. To execute or explain SQL, select Execute SQL and Explain SQL from the Design or pop-up menu.
Define or modify extended attributes	Select from the Object>Insert menu the type of extended attribute you want to define or modify, or highlight the extended attribute from the list in the Extended Attributes view and select New or Properties from the pop-up menu.
Specify extended attributes for a column	Drag the column to the Object Details view and select the Extended Attributes tab.

To do this	Do this
Access database utilities	Double-click a utility in the Objects view to launch it.
Log your work	Select Design>Start Log from the menu bar. To see the SQL syntax generated, display the Activity Log view.

Modifying database preferences

To modify database preferences, select Design>Options from the menu bar. Some preferences are specific to the database connection; others are specific to the Database painter.

Preferences on the General properties page

The Connect To Default Profile, Shared Database Profiles, Keep Connection Open, Use Extended Attributes, and Read Only preferences are database-connection-specific preferences. For information about modifying these preferences, see the *Resource Guide*.

The remaining preferences are specific to the Database painter, and are shown in Table 16-4.

Table 16-4: Database painter preferences

Database preference	What PocketBuilder does with the specified preference
Columns in the Table List	When PocketBuilder displays tables graphically, eight table columns display unless you change the number of columns.
SQL Terminator Character	PocketBuilder uses the semicolon as the SQL statement terminator unless you enter a different terminator character in the box.
Refresh Table List	When PocketBuilder first displays a table list, it retrieves the table list from the database and displays it. To save time, PocketBuilder saves this list internally for reuse to avoid regenerating very large table lists. The table list is refreshed every 30 minutes (1800 seconds) unless you specify a different refresh rate.

Preferences on the Object Colors property page

You can set colors separately for each component of the Database painter's graphical table representation: the table header, columns, indexes, primary key, foreign keys, and joins. Set a color preference by selecting a color from a drop-down list.

You can design custom colors for use when you select color preferences. To design custom colors, select Design>Custom Colors from the menu bar and work in the Custom Colors dialog box.

Logging your work

As you work with your database, you generate SQL statements. As you define a new table, for example, PocketBuilder builds a SQL CREATE TABLE statement internally. When you click the Create button, PocketBuilder sends the SQL statement to the DBMS to create the table. Similarly, when you add an index, PocketBuilder builds a CREATE INDEX statement.

You can see all SQL generated in a Database painter session in the Activity Log view. You can also save this information to a file. This allows you to have a record of your work and makes it easy to duplicate the work if you need to create the same or similar tables in another database.

❖ **To start logging your work:**

1 Open the Database painter.

2 Select Start Log from the Design menu or the pop-up menu in the Activity Log view.

PocketBuilder begins sending all generated syntax to the Activity Log view.

❖ **To stop the log:**

- Select Stop Log from the Design menu or the pop-up menu in the Activity Log view.

PocketBuilder stops sending the generated syntax to the Activity Log view. Your work is no longer logged.

❖ **To save the log to a permanent text file:**

1 Select Save or Save As from the File menu or the pop-up menu in the Activity Log view.

2 Name the file and click Save. The default file extension is *SQL*, but you can change that if you want to.

Submitting the log to your DBMS
You can open a saved log file and submit it to your DBMS in the ISQL view. For more information, see “Building and executing SQL statements” on page 397.

Creating databases

In PocketBuilder you typically work within an existing database, but you can create a new local SQL Anywhere (Adaptive Server Anywhere) or UltraLite database from within PocketBuilder. You can also delete existing SQL Anywhere databases.

Creating a SQL Anywhere database

You can create a SQL Anywhere database in the Database painter or the Database Profiles dialog box.

❖ **To create a local SQL Anywhere database:**

1. From the Objects view in the Database painter or the Database Profiles dialog box, launch the Create ASA Database utility.

   The Create Adaptive Server Anywhere Database dialog box displays. (Adaptive Server Anywhere is the name for previous versions of SQL Anywhere.)

2. Select the version of SQL Anywhere (ASA Version) that you want to use to create the database, and enter a user ID and password for the new database.

3. In the Database Name box, specify the file name and path of the database you are creating.

   If you do not provide a file extension, the database file name is given the extension *DB*.

4. Define other properties of the database as needed.

   If you are using a non-English database, you can specify a code page in the Collation Sequence box.

   For complete information about filling in the dialog box, click the Help button in the dialog box.

5. Click OK.

   When you click OK, PocketBuilder does the following:

   - Creates a database with the specified name in the specified directory or folder. If a database with the same name exists, you are asked whether you want to replace it.
   - Adds a data source to the *ODBC.INI* key in the registry. The data source has the same name as the database unless one with the same name already exists, in which case a suffix is appended.

- Creates a database profile and adds it to the registry. The profile has the same name as the database unless one with the same name already exists, in which case a suffix is appended.
- Connects to the new database.

Deleting a SQL Anywhere database

You can delete a SQL Anywhere database in the Database painter or the Database Profiles dialog box.

❖ **To delete a local SQL Anywhere database:**

1. From the Objects view in the Database painter or the Database Profiles dialog box, launch the Delete ASA Database utility.

   The Delete Local Database dialog box displays.

2. Select the database you want to delete and select Open.
3. Click Yes to delete the database.

   When you click Yes, PocketBuilder deletes the specified database.

Creating an UltraLite database

The following procedure shows how to create a new UltraLite database using the UltraLite Schema Painter and Create UltraLite Database utilities. You can also create an UltraLite schema and database from an existing SQL Anywhere database. For an example, see the DWExam example in the *Code Examples\DataWindows* directory and the chapter on MobiLink synchronization in the *Resource Guide*.

For more information about creating UltraLite databases, see the *UltraLite Database Management Reference* book in the online book collection for SQL Anywhere Studio.

❖ **To create a local UltraLite database:**

1. From the Objects view in the Database painter or the Database Profiles dialog box, launch the UltraLite Schema Painter utility.
2. Select File>New>UltraLite Schema.
3. Click Browse to navigate to the location where you want to create the schema (.USM) file, type a name for the schema file, and click Open.
4. If necessary, select a different collation sequence, select the check box if you want the database to be case sensitive, and click OK.

   You do not need to change the collation sequence for most languages based on the Roman alphabet.

5 Expand the schema and its Tables folder in the left pane and click Add Table.

The New Table dialog box opens.

6 Complete the dialog box to add a new table, click OK, and add additional tables as needed.

For complete information about filling in the dialog box, click the Help button.

7 Click Save and close the UltraLite Schema Painter.

8 From the Objects view, launch the Create UltraLite Database utility.

The Create UltraLite Database utility runs ulconv, the UltraLite database converter command-line tool.

9 Click the Browse button next to the UL Schema box and browse to select the schema file you just created.

10 Click the Browse button next to the New UL DB box, browse to select the path to the new database, enter a name, and click OK.

Working with tables

When you open the Database painter, the Object view lists all tables in the current database that you have access to (including tables that were not created using PocketBuilder). You can create a new table or alter an existing table. You can also modify table properties and work with indexes and keys.

Creating and modifying tables in an UltraLite database
To create and modify tables in an UltraLite database, use the UltraLite Schema Painter utility.

Creating a new table from scratch

In PocketBuilder, you can create a new table in a database to which PocketBuilder is connected.

❖ **To create a table in the current database:**

1 Do one of the following:

- Click the Create Table button
- Right-click in the Columns view and select New Table from the pop-up menu
- Right-click Tables in the Objects view and select New Table from the pop-up menu
- Select Insert>Table from the Object menu

The new table template displays in the Columns view. You use this template to specify each column in the table. The insertion point is in the Column Name box for the first column.

2 Enter the required information for this column.

For what to enter in each field, see "Specifying column definitions" on page 369.

As you enter information, use the Tab key to move from place to place in the column definition. After defining the last item in the column definition, press the Tab key to display the work area for the next column.

3 Repeat step 2 for each additional column in your table.

4 (Optional) Select Object>Pending SQL from the menu bar or select Pending SQL from the pop-up menu to see the pending SQL syntax.

If you have not already named the table, you must provide a name in the dialog box that displays. To hide the SQL syntax and return to the table columns, select Object>Pending Syntax from the menu bar.

5 Click the Save button or select Save from the File or pop-up menu, then enter a name for the table in the Create New Table dialog box.

PocketBuilder submits the pending SQL syntax statements it generated to the DBMS, and the table is created. The new table is displayed in the Object Layout view.

About saving the table
If you make changes after you save the table and before you close it, you see the pending changes when you select Pending SQL again. When you click Save again, PocketBuilder submits a DROP TABLE statement to the DBMS, recreates the table, and applies all changes that are pending. Clicking Save many times can be time consuming when you are working with large tables, so you might want to save only when you have finished.

6 Specify extended attributes for the columns.

For what to enter in each field, see "Specifying column extended attributes" on page 371.

Creating a new table from an existing table

You can very quickly create a new table that is similar to an existing table by using the Save Table As menu option.

❖ **To create a new table from an existing table:**

1 Open the existing table in the Columns view by dragging and dropping it or by selecting Alter Table from the pop-up menu.

2 Right-click in the Columns view and select Save Table As from the pop-up menu.

The Create New Table dialog box displays.

3 Enter a name for the new table and then the owner's name and click OK.

The new table appears in the Objects Layout view and the Columns view.

4 Make whatever changes you want to the table definition.

5 Save the table.

6 Make changes to the table's properties in the Object Details view.

For more information about modifying table properties, see "Specifying table and column properties" on page 370.

Specifying column definitions

When you create a new table, you must specify a definition for each column.

Table 16-5: Defining columns in the Columns view

Field	What you enter
Column Name	(Required) The name by which the column will be identified.
Data Type	(Required) Select a datatype from the drop-down list. All datatypes supported by the current DBMS are displayed in the list.
Width	For datatypes with variable widths, the number of characters in the field.
Dec	For numeric datatypes, the number of decimal places to display.
Null	Select Yes or No from the Null drop-down list to specify whether NULLs are allowed in the column. Specifying No means the column cannot have NULL values; users must supply a value. No is the default in a new table.
Default	The value that will be placed in a column in a row that you insert into a DataWindow object. The drop-down list has built-in choices, but you can type any other value. For an explanation of the built-in choices, see your DBMS documentation.

Specifying table and column properties

After a table has been created and saved, you can specify the properties of a table and of any column in a table. Table properties include the fonts used for headers, labels, and data, and a comment that you can associate with the table. Column properties include the text used for headers and labels, display formats, validation rules, and edit styles used for data (also known as a column's extended attributes), and a comment you can associate with the column.

Extended attribute system tables not supported for UltraLite databases
Comments, font properties, and extended attributes are stored in the PocketBuilder extended attribute system tables. System tables are not supported in UltraLite databases, but you can still take advantage of extended attributes in your DataWindow objects. For more information, see "Using extended attributes with an UltraLite application" on page 452.

Specifying table properties

Besides adding a comment to associate with the table, you can choose the fonts that will be used to display information from the table in a DataWindow object. You can specify the font, point size, color, and style.

❖ **To specify table properties:**

1 Do one of the following:

- Highlight the table in either the Objects view or the Object Layout view and select Properties from the Object or pop-up menu
- Click the Properties button
- Drag and drop the table to the Object Details view

The properties for the table display in the Object Details view.

2 Select a tab and specify properties:

Select this tab	To modify this property
General	Comments associated with the table
Data Font	Font for data retrieved from the database and displayed in the Results view by clicking a Data Manipulation button
Heading Font	Font for column identifiers used in grid, tabular, and n-up DataWindow objects displayed in the Results view by clicking a Data Manipulation button
Label Font	Font for column identifiers used in freeform DataWindow objects displayed in the Results view by clicking a Data Manipulation button

3 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you made in the Object Details view are immediately saved to the table definition.

Specifying column extended attributes

Besides adding a comment to associate with a column, you can specify extended attributes for each column. An extended attribute is information specific to PocketBuilder that enhances the definition of the column.

❖ **To specify extended attributes:**

1 Do one of the following:

- Highlight the column in either the Objects view or the Object Layout view and select Properties from the Object or pop-up menu
- Click the Properties button
- Drag and drop the column to the Object Details view

2 Select a tab and specify extended attribute values:

Select this tab	To modify these extended attributes
General	Column comments.
Headers	Label text used in freeform DataWindow objects and header text used in tabular, grid, or n-up DataWindow objects.
Display	How the data is formatted in a DataWindow object as well as display height, width, and position. For example, you can associate a display format with a Revenue column so that its data displays with a leading dollar sign and negative numbers display in parentheses.
Validation	Criteria that a value must pass to be accepted in a DataWindow object and as an initial value for the column. For example, you can associate a validation rule with a Salary column so that users can enter only a value within a particular range. For the initial value, you can select a value from the drop-down list. The initial value must be the same datatype as the column, must pass validation, and can be NULL only if NULL is allowed for the column.
Edit Style	How the column is presented in a DataWindow object. For example, you can display column values as radio buttons or in a drop-down list.

3 Right-click on the Column property sheet and select Save Changes from the pop-up menu.

Any changes you made in the property sheet are immediately saved to the table definition.

Overriding definitions
In the DataWindow painter, you can override the extended attributes specified in the Database painter for a particular DataWindow object.

How the information is stored

Extended attributes are stored in the PocketBuilder system tables in the database. PocketBuilder uses the information to display, present, and validate data in the Database painter and in DataWindow objects. When you create a view in the Database painter, the extended attributes of the table columns used in the view are used by default.

About display formats, edit styles, and validation rules

In the Database painter, you create display formats, edit styles, and validation rules. Whatever you create is then available for use with columns in tables in the database. You can see all the display formats, edit styles, and validation rules defined for the database in the Extended Attributes view.

For more information about defining, maintaining, and using these extended attributes, see Chapter 21, "Displaying and Validating Data."

About headings and labels

By default, PocketBuilder uses the column names as labels and headings, replacing any underscore characters with spaces and capitalizing each word in the name. For example, the default heading for the column Dept_name is `Dept Name`. To define multiple-line headings, press Ctrl+Enter to begin a new line.

Specifying additional properties for character columns

You can set two additional properties for character columns on the Display property page: Case and Picture.

Specifying the displayed case

You can specify whether PocketBuilder converts the case of characters for a column in a DataWindow object.

❖ **To specify how character data should be displayed:**

- On the Display property page, select a value in the Case drop-down list:

Value	Meaning
Any	Characters are displayed as they are entered
UPPER	Characters are converted to uppercase
lower	Characters are converted to lowercase

Specifying a column as a picture

You can specify that a character column can contain names of picture files (*BMP* files).

❖ **To specify that column values are names of picture files:**

1 On the Display property page, select the Picture check box.

When the Picture check box is selected, PocketBuilder expects to find picture file names in the column, but displays the contents of the picture file—not the name of the file—in reports and DataWindow objects.

Because PocketBuilder cannot determine the size of the image until runtime, it sets both display height and display width to 0 when you select the Picture check box.

2 Enter the size and the justification for the picture (optional).

Altering a table

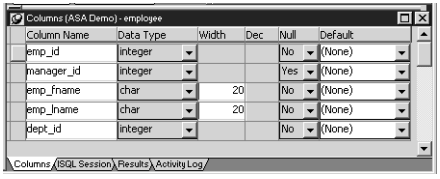
After a table is created, you can make the following modifications if the table is not in an UltraLite database (you need to modify an UltraLite database schema to modify a table):

- Add or modify extended attributes for columns
- Delete an index and create a new index
- Append columns that allow NULLs
- Increase or decrease the number of characters allowed for data in an existing column

You cannot:

- Insert a column between two existing columns
- Prohibit NULL values for an appended column
- Prohibit NULLs in a column that allowed NULLs
- Allow NULLs in a column that did not allow NULLs
- Alter an existing index

Figure 16-3: Table definition in Columns view of Database painter



❖ **To alter a table:**

1 Highlight the table and select Alter Table from the pop-up menu.

> **Opening multiple instances of tables**
> You can open another instance of a table by selecting Columns from the View menu. Doing this is helpful when you want to use the Database painter's cut, copy, and paste features to cut or copy and paste between tables.

The table definition displays in the Columns view.

2 Make the changes you want in the Columns view or in the Object Details view.

3 Select Save Table or Save Changes.

PocketBuilder submits the pending SQL syntax statements it generated to the DBMS, and the table is modified.

Cutting, copying, and pasting columns

In the Database painter, you can use the Cut, Copy, and Paste buttons in the PainterBar (or Cut, Copy, and Paste from the Edit or pop-up menu) to cut, copy, and paste one column at a time within a table or between tables.

❖ **To cut or copy a column within a table:**

1 Put the insertion point anywhere in the column you want to cut or copy.

2 Click the Cut or Copy button in the PainterBar.

❖ **To paste a column within a table:**

1 Put the insertion point in the column you want to paste to.

If you are changing an existing table, put the insertion point in the last column of the table. If you try to insert a column between two columns, you get an error message. You can append a column only to an existing table. If you are defining a new table, you can paste a column anywhere.

2 Click the Paste button in the PainterBar.

❖ **To paste a column to a different table:**

1 Open another instance of the Columns view and use Alter Table to display an existing table or click New to create a new table.

2 Put the insertion point in the column you want to paste to.

3 Click the Paste button in the PainterBar.

Closing a table

You can remove a table from a view by selecting Close or Reset View from its pop-up menu. This action removes the table only from the Database painter view. It does not drop (remove) the table from the database.

Dropping a table

Dropping removes the table from the database.

❖ **To drop a table:**

1 Select Drop Table from the table's pop-up menu or select Object>Delete from the menu bar.

2 Click Yes.

Deleting orphaned table information

If you drop a table outside PocketBuilder, information remains in the system tables about the table, including extended attributes for the columns.

❖ **To delete orphaned table information from the extended attribute system tables:**

1 Select Design>Synch Extended Attributes from the menu bar.

If you try to delete orphaned table information and there is none, a message tells you that synchronization is not necessary.

2 Click Yes.

Viewing pending SQL changes

As you create or alter a table definition, you can view the pending SQL syntax changes that will be made when you save the table definition.

Figure 16-4: SQL syntax for pending changes to a table

❖ **To view pending SQL syntax changes:**

- Right-click the table definition in the Columns view and select Pending Syntax from the pop-up menu.

  PocketBuilder displays in SQL syntax the pending changes to the table definition.

The SQL statements execute only when you save the table definition or reset the view and then tell PocketBuilder to save changes.

Copying, saving, and printing pending SQL changes

When you are viewing pending SQL changes, you can:

- Copy pending changes to the clipboard
- Save pending changes to a file
- Print pending changes

To copy, save, or print only part of the SQL syntax
Select the part of the SQL syntax you want before you copy, save, or print.

❖ **To copy the SQL syntax to the clipboard:**

- In the Pending Syntax view, click the Copy button or select Copy from the pop-up menu.

❖ **To save SQL syntax for execution at a later time:**

1 In the Pending Syntax view, select File>Save As.

The Save Syntax to File dialog box displays.

2 Navigate to the folder where you want to save SQL, name the file, and then click the Save button.

At a later time, you can import the SQL file into the Database painter and execute it.

❖ **To print pending table changes:**

- While viewing the pending SQL syntax, click the Print button or select Print from the File menu.

❖ **To display columns in the Columns view:**

- Select Object>Pending Syntax from the menu bar.

Printing the table definition

You can print a report of the table's definition at any time, whether or not the table has been saved. The Table Definition Report contains information about the table and each column in the table, including the extended attributes for each column.

❖ **To print the table definition:**

- Select Print or Print Definition from the File or pop-up menu or click the Print button.

Exporting table syntax

You can export the syntax for a table to the log. This feature is useful when you want to create a backup definition of the table before you alter it, or when you want to create the same table in another DBMS.

❖ PocketBuilder

❖ **To export the syntax of an existing table to a log:**

1. Select the table in the painter workspace.
2. Select Export Syntax from the Object menu or the pop-up menu.

   If you selected a view, PocketBuilder immediately exports the syntax to the log.
3. Specify a data source in the Data Sources dialog box.
4. Supply any information you are prompted for.

   PocketBuilder exports the syntax to the log. Extended attribute information (such as validation rules used) for the selected table is also exported.

   For more information about the log, see “Logging your work” on page 364.

About system tables

Two kinds of system tables exist in the database:

- System tables provided by your DBMS

  For information about the SQL Anywhere system tables, see the chapter on system tables in the *SQL Anywhere Server SQL Reference* manual in the online books for SQL Anywhere Studio.
- PocketBuilder extended attribute system tables

No system table support in UltraLite
UltraLite databases do not support system tables.

About PocketBuilder system tables

PocketBuilder stores extended attribute information you provide when you create or modify a table (such as the text to use for labels and headings for the columns, validation rules, display formats, and edit styles) in system tables. These system tables contain information about database tables and columns. Extended attribute information extends database definitions. Table 16-6 lists the extended attribute system tables.

Table 16-6: Extended attribute system tables

This system table	Stores this extended attribute information
PBCatCol	Column data such as name, header, and label for reports and DataWindow objects, and header and label positions
PBCatEdt	Edit style names and definitions
PBCatFmt	Display format names and definitions
PBCatTbl	Table data such as name, fonts, and comments
PBCatVld	Validation rule names and definitions

In the Employee table, for example, one column name is Emp_lname. A label and a heading for the column are defined for PocketBuilder to use in DataWindow objects. The column label is defined as `Last Name:`. The column heading is defined as `Last Name`. The label and heading are stored in the PBCatCol table in the extended attribute system tables.

The extended attribute system tables are maintained by PocketBuilder. Only PocketBuilder users can enter information into the extended attribute system tables. For more information, see Appendix A, “Extended Attribute System Tables.”

Opening and displaying system tables

You can open system tables in the Database painter, just like other tables.

By default, PocketBuilder shows only user-created tables in the Objects view. If you highlight Tables and select Show System Tables from the pop-up menu, PocketBuilder also shows system tables.

Working with keys

Why you should use keys

You use primary and foreign keys to enforce the referential integrity of your database. That way you can rely on the DBMS to make sure that only valid values are entered for certain columns instead of having to write code to enforce valid values.

For example, suppose you have two tables called Department and Employee. The Department table contains the column Dept_Head_ID, which holds the ID of the department's manager. You want to make sure that only valid employee IDs are entered in this column. The only valid values for Dept_Head_ID in the Department table are values for Emp_ID in the Employee table.

To enforce this kind of relationship, you define a foreign key for Dept_Head_ID that points to the Employee table. With this key in place, the DBMS disallows any value for Dept_Head_ID that does not match an Emp_ID in the Employee table.

For more about primary and foreign keys, consult a book about relational database design.

What you can do in the Database painter

You can work with keys in the following ways:

- Look at existing primary and foreign keys
- Open all tables that depend on a particular primary key
- Open the table containing the primary key used by a particular foreign key
- Create, alter, and drop keys

Working with keys in an UltraLite database
You cannot create, alter, or drop keys in an UltraLite database.

Viewing keys

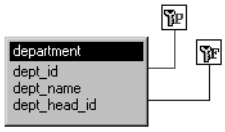
Keys can be viewed in several ways:

- In the expanded tree view of a table in the Objects view
- As icons connected by lines to a table in the Object Layout view

In the following picture, the Department table has two keys:

- A primary key (on dept_id)
- A foreign key (on dept_head_id)

Figure 16-5: Primary and foreign keys in Object Layout view



If you cannot see the lines
If the color of your window background makes it hard to see the lines for the keys and indexes, you can set the colors for each component of the Database painter's graphical table representation, including keys and indexes. For information, see "Modifying database preferences" on page 363.

Opening related tables

When working with tables containing keys, you can easily open related tables.

❖ **To open the table that a particular foreign key references:**

1 Display the foreign key pop-up menu.

2 Select Open Referenced Table.

❖ **To open all tables referencing a particular primary key:**

1 Display the primary key pop-up menu.

2 Select Open Dependent Table(s).

PocketBuilder opens and expands all tables in the database containing foreign keys that reference the selected primary key.

Defining primary keys

You can define primary keys for database tables with PocketBuilder. However, you cannot define a primary key for a table that already has one, unless you first drop the existing primary key.

❖ **To create a primary key:**

1 Do one of the following:

- Highlight the table for which you want to create a primary key and click the Create Primary Key drop-down toolbar button in PainterBar1
- Select Object>Insert>Primary Key from the Database painter menu or New>Primary Key from the pop-up menu
- Expand the table's tree view, right-click Primary Key, and select New Primary Key from the pop-up menu

The Primary Key properties display in the Object Details view.

2 Select one or more columns for the primary key.

Columns that are allowed in a primary key
Only a column that does not allow NULLs can be included as a column in a primary key definition. If you choose a column that allows NULLs, you get a DBMS error when you save the table.

3 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you made in the view are immediately saved to the table definition.

Defining foreign keys

You can define foreign keys for database tables in PocketBuilder.

❖ **To create a foreign key:**

1 Do one of the following:

- Highlight the table and click the Create Foreign Key drop-down toolbar button in PainterBar1
- Select Object>Insert>Foreign Key from the Database painter menu or New>Foreign Key from the pop-up menu
- Expand the table's tree view and right-click on Foreign Keys and select New Foreign Key from the pop-up menu

The Foreign Key properties display in the Object Details view.

2 Name the foreign key in the Foreign Key Name box.

3 Select the columns for the foreign key.

4 On the Primary Key tab page, select the table and column containing the Primary key referenced by the foreign key you are defining.

Key definitions must match exactly
The definition of the foreign key columns must match the primary key columns, including datatype, precision (width), and scale (decimal specification).

5 On the Rules tab page, specify the rule that you want applied on delete of primary table row.

The default rule applied is "Disallow if Dependent Rows Exist (RESTRICT)".

6 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you make in the view are immediately saved to the table definition.

Modifying keys

You can modify a primary key in PocketBuilder.

❖ **To modify a primary key:**

1 Do one of the following:

- Highlight the primary key listed in the table's expanded tree view and click the Properties button
- Select Properties from the Object or pop-up menu
- Drag the primary key icon and drop it in the Object Details view

2 Select one or more columns for the primary key.

3 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you make in the view are immediately saved to the table definition.

Dropping a key

You can drop keys (remove them from the database) from within PocketBuilder.

❖ **To drop a key:**

1 Highlight the key in the expanded tree view for the table in the Objects view or right-click the key icon for the table in the Object Layout view.

2 Select Drop Primary Key or Drop Foreign Key from the key's pop-up menu.

3 Click Yes.

Working with indexes

Creating an index

You can create as many single- or multi-valued indexes for a database table as you need, and you can drop indexes that are no longer needed.

Update limitation
You can update a table in a DataWindow object only if it has a unique index or primary key.

In SQL Anywhere databases, you should not define an index on a column that is defined as a foreign key, because foreign keys are already optimized for quick reference.

In UltraLite databases, the ascending or descending attribute is applied to each column in the index. UltraLite supports the ability to have an ascending index column and a descending index column in the same index.

❖ **To create an index:**

1. Do one of the following:
   - Highlight the table for which you want to create an index and click the Create Index drop-down toolbar button in PainterBar1
   - Select Object>Insert>Index from the Database painter menu or New>Index from the pop-up menu
   - Expand the table's tree view and right-click on Indexes and select New Index from the pop-up menu

   The Index's properties display in the Object Details view.
2. Enter a name for the index in the Index box.
3. Select whether or not to allow duplicate values for the index.
4. Specify any other information required for your database

   For SQL Anywhere, specify the order of the index.
5. Click the names of the columns that make up the index.
6. Select Save Changes from the pop-up menu.
7. Right-click on the Object Details view and select Save Changes from the pop-up menu.

   Any changes you made in the view are immediately saved to the table definition.

Modifying an index

You can modify an index.

❖ **To modify an index:**

1 Do one of the following:

- Highlight the index listed in the table's expanded tree view and click the Properties button
- Select Properties from the Object or pop-up menu
- Drag the index icon and drop it in the Object Details view

2 In the Object Details view, select or deselect columns as needed.

3 Right-click on the Object Details view and select Save Changes from the pop-up menu.

Any changes you made in the view are immediately saved to the table definition.

Dropping an index

Dropping an index removes it from the database.

❖ **To drop an index from a table:**

1 In the Database painter workspace, display the pop-up menu for the index you want to drop.

2 Select Drop Index and click Yes.

Working with database views

A database view gives a different (and usually limited) perspective of the data in one or more tables. Although you see existing database views listed in the Objects view, a database view does not physically exist in the database as a table does. Each time you select a database view and use the view's data, PocketBuilder executes a SQL SELECT statement to retrieve the data and creates the database view.

Views are not supported in UltraLite databases.

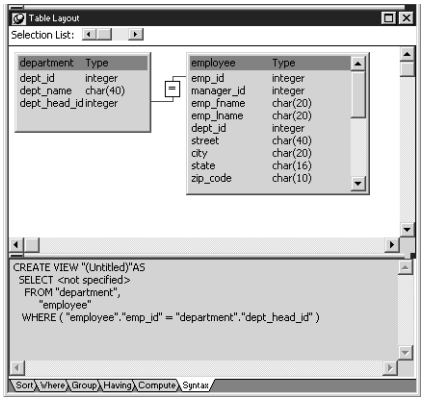
Using database views in PocketBuilder

You can define and manipulate database views in PocketBuilder. Typically you use database views for the following reasons:

- To give names to frequently executed SELECT statements.
- To limit access to data in a table. For example, you can create a database view of all the columns in the Employee table except Salary. Users of the database view can see and update all information except the employee's salary.
- To combine information from multiple tables for easy access.

In PocketBuilder, you can create single- or multiple-table database views. You can also use a database view when you define data to create a new database view.

Figure 16-6: Creating a multiple-table view in the View painter



❖ **To create a database view:**

1 Click the Create View drop-down toolbar button, select Object>Insert>View from the Database painter menu, or select New View from the pop-up menu on the Views folder in the Objects view.

The Select Tables dialog box displays, listing all tables and views that you can access in the database.

2 Select the tables and views from which you will create the view by doing one of the following:

- Click the name of each table or view you want to open in the list displayed in the Select Tables dialog box, then click the Open button to open them. The Select Tables dialog box closes.
- Double-click the name of each table or view you want to open. Each object is opened immediately. Then click the Cancel button to close the Select Tables dialog box.

Representations of the selected tables and views display in the View painter workspace.

3 Select the columns to include in the view and include computed columns as needed.

4 Join the tables if there is more than one table in the view.

For information, see "Joining tables" on page 389.

5 Specify criteria to limit rows retrieved (Where tab), group retrieved rows (Group tab), and limit the retrieved groups (Having tab) if appropriate.

For information about using selection and grouping criteria, see "Specifying selection, sorting, and grouping criteria" on page 440.

6 When the view has been completed, click the Return button.

7 Name the view.

Include "view" or some other identifier in the view's name so that you will be able to distinguish it from a table in the Select Tables dialog box.

8 Click the Create button.

PocketBuilder generates a CREATE VIEW statement and submits it to the DBMS. The view definition is created in the database. You return to the Database painter workspace with the new view displayed in the workspace.

Opening a database view

You define, open, and manipulate database views in the View painter, which is similar to the Select painter. For more information about the Select painter, see "Selecting a data source" on page 419.

❖ **To open a database view:**

1 In the Objects view of the database painter, expand the list of Views for your database.

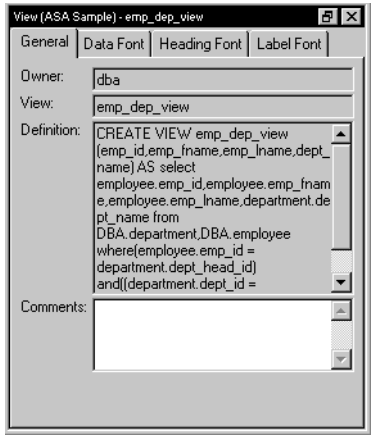
2 Highlight the view you want to open and select Add To Layout from the pop-up menu, or drag the view's icon to the Object Layout view.

Updating database views
Some database views are logically updatable and others are not. Views are not supported in UltraLite. For SQL Anywhere you cannot update views containing aggregate functions, such as COUNT(*), or a GROUP BY clause in the SELECT statement, or views containing a UNION operation.

Displaying a database view's SQL statement

You can display the SQL statement that defines a database view. How you do it depends on whether you are creating a new view in the View painter or want to look at the definition of an existing view.

Figure 16-7: Displaying a view definition in the Database painter



You cannot alter the view definition in the Object Details view. To alter a view, drop it and create another view.

❖ **To display the SQL statement from the View painter:**

- Select the Syntax tab in the View painter

  PocketBuilder displays the SQL it is generating. The display is updated each time you change the view.

❖ **To display the SQL statement from the Database painter:**

- Highlight the name of the database view in the Objects view and select Properties from the pop-up menu, or drag the view's icon to the Object Details view.

  The completed CREATE statement used to create the database view displays in the Definition field on the General page. The view definition in the Object Details view is read-only.

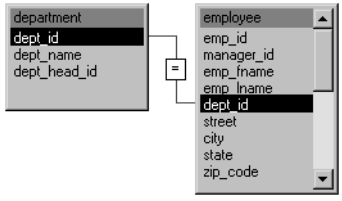
Joining tables

If the database view contains more than one table, you should join the tables on their common columns. When the View painter is first opened for a database view containing more than one table, PocketBuilder makes its best guess as to the join columns, as follows:

- If there is a primary/foreign key relationship between the tables, PocketBuilder automatically joins them.
- If there are no keys, PocketBuilder tries to join tables based on common column names and types.

In the following screen, the Employee and Department tables are joined on the dept_id column:.

Figure 16-8: Equality join between two tables



❖ **To join tables:**

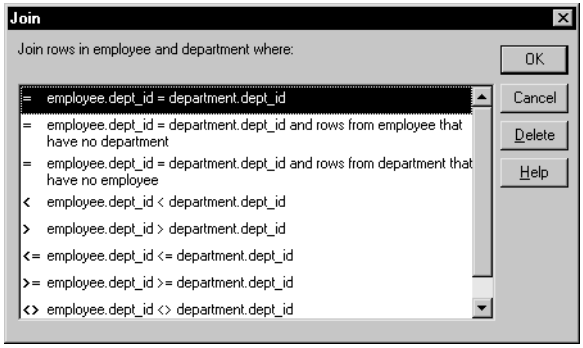
1 Click the Join button.

2 Click the columns on which you want to join the tables.

3 To create a join other than the equality join, click the join representation in the workspace.

  The Join dialog box displays.

Figure 16-9: Join dialog box showing types of join allowed



4 Select the join operator you want from the Join dialog box.

You can select outer joins for SQL Anywhere databases. For example, in the preceding dialog box (which uses the Employee and Department tables), you can choose to include rows from the Employee table where there are no matching departments, or rows from the Department table where there are no matching employees.

For more about outer joins, see "Using ANSI outer joins" on page 437.

Dropping a database view

Dropping a database view removes its definition from the database.

❖ **To drop a view:**

1 In the Objects view, select the database view you want to drop.

2 Click the Drop Object button in PainterBar1 or select Drop View from the pop-up menu.

PocketBuilder prompts you to confirm the drop, then generates a DROP VIEW statement and submits it to the DBMS.

Exporting view syntax

You can export the syntax for a view to the log. This feature is useful when you want to create a backup definition of the view before you alter it or when you want to create the same view in another DBMS.

❖ **To export the syntax of an existing view to a log:**

1 Select the view in the painter workspace.

2 Select Export Syntax from the Object menu or the pop-up menu.

For more information about the log, see "Logging your work" on page 364.

Manipulating data

As you work on the database, you often want to look at existing data or create some data for testing purposes. You might also want to test display formats, validation rules, and edit styles on real data.

PocketBuilder provides data manipulation for such purposes. With data manipulation, you can:

- Retrieve and manipulate database information
- Save the contents of the database in a variety of formats (such as Excel, HTML tables, or SQL)

Retrieving data

When you retrieve data in the Database painter, what you see is actually a DataWindow object. The formatting style you pick for retrieval corresponds to a type of DataWindow object (grid, tabular, or freeform). In a grid display, you can drag the mouse on a column's border to resize the column.

Figure 16-10: Retrieving data in grid format


Results

Customer ID	First Name	Last Name	Address	City	State	Zip Code	Phone Number
101	Michaels	Devlin	3114 Pioneer Avenue	Rutherford	NJ	07070-	(201) 555-8966
102	Beth	Reiser	1033 Whippany Road	New York	NY	10154	(212) 555-8725
103	Erin	Niedringhaus	1990 Windsor Street	Paoli	PA	19301	(215) 555-6513
104	Meghan	Mason	550 Dundas Street East	Knoxville	TN	37919	(615) 555-5463
105	Laura	McCarthy	1210 Highway 36	Carmel	IN	46032	(317) 555-8437
106	Paul	Phillips	2000 Cherry Creek N. Dr.	Middletown	CT	64579-	(203) 555-3464
107	Kelly	Colburn	18131 Vallco Parkway	Raleigh	NC	27695-	(919) 555-5152
108	Matthew	Goforth	11801 Wayzata Blvd.	Chattanooga	TN	37421	(615) 555-8926
109	Jessie	Gagliardo	2800 Park Avenue	Hull	PQ	-	(819) 555-9539
110	Michael	Agliori	13705 North Glebe Road	Columbus	OH	43216	(614) 555-2496
111	Dylan	Ricci	14700 Prosperity Avenue	Syracuse	NY	13202	(315) 555-4486
112	Shawn	McDonough	15175 S Main Street	Brooklyn Park	MN	55428	(612) 555-5603
113	Samuel	Kaiser	404 Bristol Street	Minneapolis	MN	55041	(612) 555-3409

❖ **To retrieve data:**

1 In the Database painter, select the table or database view whose data you want to manipulate.

2 Do one of the following:

- Click one of the three Data Manipulation buttons (Grid, Tabular, or Freeform) in the PainterBar.
- Select Data or Edit Data from the Object or pop-up menu and choose one of the edit options from the cascading menu that displays.

All rows are retrieved and display in the Results view. Exactly what you see in the Results view depends on the formatting style you picked.

As the rows are being retrieved, the Retrieve button in PainterBar3 changes to a Cancel button. You can click the Cancel button to stop the retrieval.

When results are retrieved, only a few rows of data display at a time. In freeform format, a single row of data displays. You can use the First, Prior, Next, and Last buttons in PainterBar3 to page through the rows of data.

Modifying data

You can add, modify, or delete rows. When you have finished manipulating the data, you can apply the changes to the database.

If looking at data from a view
Some views are logically updatable and others are not. For SQL Anywhere you cannot update views containing aggregate functions, such as COUNT(*), or a GROUP BY clause in the SELECT statement, or views containing a UNION operation.

❖ **To modify data:**

1 Do one of the following:

- To modify existing data, tab to a field and enter a new value.
- To add a row, click the Insert Row button and enter data in the new row.
- To delete a row, click the delete Row button.

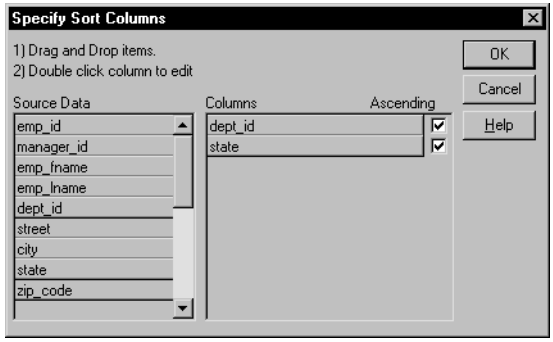
When you add or modify data, the data uses the validation rules, display formats, and edit styles that you or others have defined for the table in the Database painter.

2 Click the Save Changes button or select Rows>Update to apply changes to the database.

Sorting rows

You can sort the data, but any sort criteria you define are for testing only and are not saved with the table or passed to the DataWindow painter.

Figure 16-11: Specifying sort criteria in the Database painter



The order in which the columns display in the Columns box determines the precedence of the sorting. For example, in the preceding dialog box, rows will be sorted by department ID. Within department ID, rows will be sorted by state.

❖ **To sort the rows:**

1 Select Rows>Sort from the menu bar.

The Specify Sort Columns dialog box displays.

2 Drag the columns you want to sort on from the Source Data box to the Columns box:

A check box with a check mark in it displays under the Ascending heading to indicate that the values will be sorted in ascending order. To sort in descending order, clear the check box.

3 Change the sort order precedence by dragging the column names in the Column box to achieve the sort order you want.

4 (Optional) Double-click an item in the Columns box to specify an expression to sort on.

The Modify Expression dialog box displays.

5 Specify the expression.

For example, if you have two columns, Revenues and Expenses, you can sort on the expression `Revenues - Expenses`.

6 Click OK.

You return to the Specify Sort Columns dialog with the expression displayed.

If you change your mind
You can remove a column or expression from the sorting specification by simply dragging it and releasing it outside the Columns box.

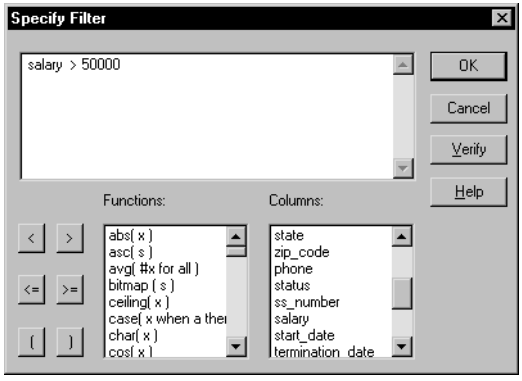
7 When you have specified all the sort columns and expressions, click OK.

Filtering rows

You can limit which rows are displayed by defining a filter.

The filters you define are for testing only and are not saved with the table or passed to the DataWindow painter.

Figure 16-12: Specifying a boolean expression as a filter for testing



❖ **To filter the rows:**

1 Select Rows>Filter from the menu bar.

The Specify Filter dialog box displays.

2 Enter a boolean expression that PocketBuilder will test against each row.

If the expression evaluates to TRUE, the row is displayed. You can paste functions, columns, and operators in the expression.

3 Click OK.

PocketBuilder filters the data. Only rows meeting the filter criteria are displayed.

❖ **To remove the filter:**

1 Select Rows>Filter from the menu bar.

The Specify Filter dialog box displays, showing the current filter.

2 Delete the filter expression, then click OK.

Filtered rows and updates
Filtered rows are updated when you update the database.

Viewing row information

You can display information about the data you have retrieved.

❖ **To display row information:**

- Select Rows>Described from the menu bar.

  The Describe Rows dialog box displays, showing the number of:

  - Rows that have been deleted in the painter but not yet deleted from the database
  - Rows displayed in Preview
  - Rows that have been filtered
  - Rows that have been modified in the painter but not yet modified in the database

All row counts are zero until you retrieve the data from the database or add a new row. The count changes when you modify the displayed data or test filter criteria.

Importing data

You can import data from an external source and then save the imported data in the database.

❖ **To import data:**

1 Select Rows>Import from the menu bar.

The Select Import File dialog box displays.

2 Specify the file from which you want to import the data.

The types of files that you can import into the painter are shown in the Files of Type drop-down list.

3 Click Open.

PocketBuilder reads the data from the file. You can click the Save Changes button or select Rows>Update to add the new rows to the database.

Printing data

You can print the data displayed by selecting File>Print from the menu bar. Before printing, you can also preview the output on the screen.

❖ **To preview printed output before printing:**

1 Select File>Print Preview from the menu bar.

Preview displays the data as it will print. To display rulers around the page borders in Print Preview, select File>Print Preview Rulers.

2 To change the magnification used in Print Preview, select File>Print Preview Zoom from the menu bar.

The Zoom dialog box displays.

3 Select the magnification you want and click OK.

Preview zooms in or out as appropriate.

4 When you have finished looking at the print layout, select File>Print Preview from the menu bar again.

Saving data

You can save the displayed data in an external file using one of the formats supported by PocketBuilder.

❖ **To save the data in an external file:**

1 Select Rows>Save Rows As from the menu bar.

The Save Rows As dialog box displays.

2 Choose a format for the file.

   If you want the column headers saved in the file, select a file format that includes headers, such as Excel With Headers. When you select a "with headers" format, the names of the database columns (not the column labels) are also saved in the file.

   For more information, see "Saving data in an external file" on page 474.

3 Name the file and save it.

   PocketBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Creating and executing SQL statements

The Database painter's Interactive SQL view is a SQL editor in which you can enter and execute SQL statements. The view provides all editing capabilities needed for writing and modifying SQL statements. You can cut, copy, and paste text; search for and replace text; and paint SQL statements. You can also set editing properties to make reading your SQL files easier.

Building and executing SQL statements

You can use the Interactive SQL view to build SQL statements and execute them immediately. The view acts as a notepad in which you can enter SQL statements.

Creating stored procedures

You can use the Interactive SQL (ISQL) view to create stored procedures or triggers, but make sure that the painter's SQL statement terminator character is not the same as the terminator character used in the stored procedure language of your DBMS. Stored procedures are not supported in UltraLite databases.

About the statement terminator

By default, PocketBuilder uses the semicolon as the SQL statement terminator. You can override the semicolon by specifying a different terminator character in the Database painter. To change the terminator character, select Design>Options from the Database painter's menu bar.

Make sure that the character you choose is not reserved for another use by your database vendor. For example, using the slash character (/) will cause compilation errors with some DBMSs.

Controlling comments

By default, PocketBuilder strips off comments when it sends SQL to the DBMS. You can have comments included by clearing the check mark next to Strip Comments in the pop-up menu of the ISQL view.

Entering SQL

You can enter a SQL statement in four ways:

- Pasting the statement
- Typing the statement in the view
- Opening a text file containing the SQL
- Dragging a procedure or function from the Objects view

Pasting SQL

You can paste SELECT, INSERT, UPDATE, and DELETE statements to the view. Depending on which kind of statement you want to paste, PocketBuilder displays dialog boxes that guide you through painting the full statement.

❖ **To paste a SQL statement to the workspace:**

1 Display the ISQL view in the Database painter.

2 Click the Paste SQL button in PainterBar2, or select Paste Special>SQL from the Edit or pop-up menu.

The SQL Statement Type dialog box displays, listing the types of SQL statements you can use.

3 Double-click the appropriate icon to select the statement type.

The Select Table dialog box displays.

4 Select the table(s) you will reference in the SQL statement.

You go to the Select, Insert, Update, or Delete painter, depending on the type of SQL statement you are pasting.

5 Do one of the following:

- For a SELECT statement, define the statement exactly as in the Select painter when building a view.

  You choose the columns to select. You can define computed columns, specify sorting and joining criteria, and WHERE, GROUP BY, and HAVING criteria. For more information, see "Working with database views" on page 385. For more information about the Select painter, see "Selecting a data source" on page 419.

- For an INSERT statement, type the values to insert into each column. You can insert as many rows as you want.

- For an UPDATE statement, specify the new values for the columns in the Update Column Values dialog box. Then specify the WHERE criteria to indicate which rows to update.

- For a DELETE statement, specify the WHERE criteria to indicate which rows to delete.

6 When you have completed painting the SQL statement, click the Return button in the PainterBar in the Select, Insert, Update, or Delete painter.

  You return to the Database painter with the SQL statement pasted into the ISQL view.

Typing SQL

Rather than paste, you can simply type one or more SQL statements directly in the ISQL view.

You can enter most statements supported by your DBMS. This includes statements you can paint as well as statements you cannot paint, such as a database stored procedure or a CREATE TRIGGER statement. You cannot enter certain statements that could destabilize the PocketBuilder development environment. These include the SET statement and the USE *database* statement.

Importing SQL from a text file

You can import SQL from a text file to the Database painter.

❖ **To read SQL from a file:**

1 Put the insertion point where you want to insert the SQL.

2 Select Paste Special>From File from the Edit or pop-up menu.

3 Select the file containing the SQL and click OK.

Dragging a procedure or function from the Objects view

From the tree view in the Objects view, you can select an existing procedure or function that contains a SQL statement you want to enter and drag it to the Interactive SQL view.

Explaining SQL

Sometimes there is more than one way to code SQL statements to obtain the results you want. When this is the case, you can use Explain SQL on the Design menu to help you select the most efficient method. Explain SQL displays information about the path that PocketBuilder will use to execute the statements in the SQL Statement Execution Plan dialog box. This is most useful when you are retrieving or updating data in an indexed column or using multiple tables. This feature is not supported for UltraLite connections.

Executing SQL

When you have typed or otherwise entered the SQL statements you want in the ISQL view of the Database painter, you can submit them to the DBMS.

❖ **To execute the SQL:**

- Click the Execute button, or select Design>Execute SQL from the menu bar

If the SQL retrieves data, the data appears in grid format in the Results view. If there is a database error, you see a message box describing the problem.

For a description of what you can do with the data, see “Manipulating data” on page 391.

Customizing the editor

The ISQL view provides the same editing capabilities as the File editor. It also has Script, Font, and Coloring properties that you can change to make SQL files easier to read. With no change in properties, SQL files have black text on a white background and a tab stop setting of 3 for indentation.

Setting Script and Font properties

Select Design>Options from the menu bar to open the Database Preferences dialog box. The Script and Font properties are the same as those you can set for the File editor.

For more information, see the section on using the File editor in “Using the File editor” on page 25.

Editor properties apply elsewhere
When you set Script and Font properties for the Database painter, the settings also apply to the Script view, the File editor, and the Debugger.

Setting Coloring properties

You can set the text color and background color for SQL styles (such as datatypes and keywords) so that the style will stand out and the SQL code will be more readable. You set Coloring properties on the Coloring tab page.

Enabling syntax coloring
Be sure the Enable Syntax Coloring check box is selected before you set colors for SQL styles. You can turn off all Coloring properties by clearing the check box.

Controlling access to the current database

The Database painter's Design>Table Security menu allows you to control access to the current database. You use the Table Security dialog box to assign table access privileges to different users and groups.

UltraLite databases do not use owners and groups. All users have full permission to update tables, and you cannot use the procedures described in this section to grant or revoke access to tables.

User maintenance in UltraLite databases

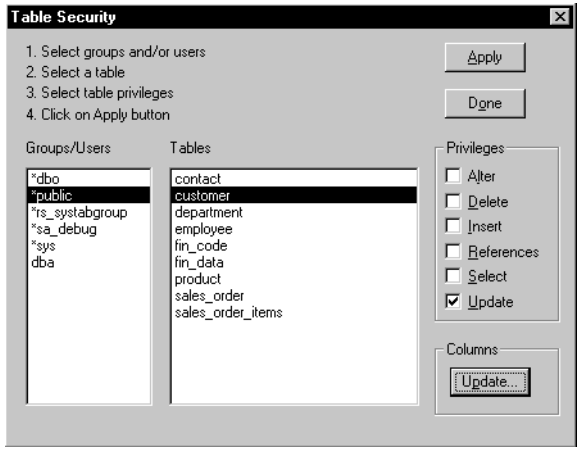
Although you cannot set table-level security for UltraLite databases, PocketBuilder allows you to add or edit multiple users in these databases. UltraLite permits a maximum of four user IDs per database. For UltraLite database profiles, you can use the pop-up menu for the Users item in the Objects view of the Database painter to add or edit users, and to delete users.

When you select Add or Edit User from the User item pop-up menu for an UltraLite database, or when you select Delete User, you should already know which user IDs exist in the database. If you select either of these menu items, the Properties view displays. The Properties view has three text boxes (User, New Password, and Confirm New Password) when you select Add or Edit User, and a single text box (UltraLite User to Delete) when you select Delete User.

After you save changes in the Database painter involving user maintenance for an UltraLite database, the Output view displays a message with the type of change you made.

Figure 16-13 shows the Table Security dialog box that allows you to assign privileges to selected users or groups for specific tables in a database.

Figure 16-13: Assigning privileges in the Table Security dialog box



❖ **To control access to the current database:**

1 Select Design>Table Security from the Database painter menu

2 Select a user or group, or multiple users and groups, from the Groups/Users list box.

3 Select a table from the Tables list box.

4 Select the privileges you want to give the selected users or groups for the table you selected in the previous step, and click Apply.

Update privileges can be restricted to specific columns of the selected table by clicking the Update button, selecting only those columns for which you want to grant the privilege, and clicking OK.

5 Repeat steps 2 – 5 until you have granted all the privileges that you want to assign, then click Done.

Using the MobiLink Synchronization for ASA wizard

What the wizard generates

You use the MobiLink Synchronization for ASA wizard on the Database tab of the New dialog box to create a nonvisual user object and a global external function in your current target, making it easier to add database synchronization capabilities to the target. By default, the wizard also adds two windows, a structure, and a second external (configuration) function.

Table 16-7 shows objects that can be generated by the wizard, listed by their default names, where *appname* stands for the name of the current application.

Table 16-7: Objects generated by MobiLink Synchronization wizard

Default name	Description
nvo_*appname*_sync	Nonvisual user object that starts synchronization from the remote client.
gf_*appname*_sync	Global function that instantiates nvo_*appname*_sync to start the synchronization.This function includes the logic to start the synchronization with or without a feedback window.
w_*appname*_sync or w_sync_default_feedback	Optional feedback window that can be used, instead of the standard MobiLink status window, to display synchronization status to the client. Even if you do not use the optional status window, a generated feedback window must be instantiated by the gf_*appname*_sync function.
gf_*appname*_configure_sync	Optional global function that calls the w_*appname*_sync_options window, which allows the user to configure the dbmlsync client.

Default name	Description
w_*appname*_sync_options	Window that allows the application user to change MobiLink connection arguments at runtime.
s_*appname*_sync_parms	Optional structure used to store runtime information entered by the user in the w_*appname*_sync_options window.

Wizard options

In addition to the names of objects generated by the MobiLink Synchronization for ASA wizard, the wizard includes fields for the options listed in Table 16-8.

Table 16-8: MobiLink Synchronization for ASA wizard options

Option	Description
Destination library	Select the target PKL file where you want to generate the MobiLink synchronization objects.
Desktop database connection	Select a PocketBuilder database profile or proceed without a database connection.
ODBC data source file	This is the DSN file, a copy of which you will deploy, or have already deployed, to remote devices. Your application uses this DSN to connect to the remote database.
Publication name	Lets you select a publication (or multiple publications) if you specified a database profile for a desktop database connection. If you did not, you can type the name of a publication you want to synchronize.
Override registry settings	Lets you override registry settings from a previous deployment.
Build number	Assigns a build number for MobiLink synchronization objects. To override registry settings from a previous deployment, enter a value that is higher than the value you previously used.
Client logging options	Specifies what information gets written to the synchronization log and whether you save the information to a log file.
Additional command line options	Adds the options you specify to the command line for starting the MobiLink synchronization client. You can click the Usage button to see a list of valid options.

Option	Description
Extended options	Adds extended options you specify. You do not need to enter the "-e" switch for extended options in this field. You can click the Usage button to see a list of valid extended options. Single quotes must be used for any extended option values requiring quotation marks. You must separate multiple options with semicolons, for example: `scn=on;adr='host=localhost;port=2439'`
Host	Sets the host information for connecting to the MobiLink Synchronization Server. If you enter a value for this field, it overrides any value set in synchronization subscriptions and in the Extended Options field.
Port	Sets the port for connecting to the MobiLink Synchronization Server. The default port for MobiLink is 2439. The value you enter for this field overrides any value set in synchronization subscriptions and in the Extended Options field.

For more information about a wizard option, click inside the option field and press F1 for online Help.

About the desktop database profile field

The wizard prompts you for a database profile, which it uses to establish a connection to a remote database on the desktop. If you are not testing a connection on the desktop, you can select the option to proceed without a database connection and ignore the database profile field.

Using a remote database on the desktop
The remote desktop database defined by a database connection profile can be the database you plan to deploy to Windows CE devices or a database that you use for testing on the desktop only.

A database profile is required for automatic retrieval of publication names in the database. A publication is a database object describing data to be synchronized. A publication, along with a synchronization user name and a synchronization subscription, is required for MobiLink synchronization.

About the publication name field

The wizard lets you select multiple publication names if they exist in the remote database defined by the connection profile. There must be subscriptions associated with the publication in order for them to display in the publication selection list.

If you selected the option to proceed without a database connection, the wizard prompts you to type a publication name (or a comma-separated list of publication names) in the MobiLink Client Publication wizard page instead of prompting you to select publication names retrieved from the database.

For more information about publications, see the Publishing Data section of the *MobiLink Client Administration* book in the SQL Anywhere Studio online book collection.

❖ **To add objects for MobiLink synchronization**

1. Select File>New from the PocketBuilder menu bar.
2. Click the Database tab, select the MobiLink Synchronization for ASA wizard, and click OK.
3. Follow the instructions in the wizard, providing the information the wizard needs.

   On the last page of the wizard, make sure the Generate To-Do List check box is selected if you want the wizard to add items to the To-Do List to guide and facilitate your development work.
4. When you are satisfied with your choices in the wizard, click Finish.

   The wizard generates objects that you can use for database synchronization.

   For information about using the generated objects to synchronize a target database, see the chapter on MobiLink synchronization in the *Resource Guide*.

Using the UltraLite Synchronization wizard

About the wizard

The UltraLite Synchronization wizard generates objects that make it easier for you to initiate and control MobiLink synchronization requests from an application connection to a remote UltraLite database. Some of the objects created by the wizard are similar to the objects created by the MobiLink Synchronization for ASA wizard, although there are differences in the way MobiLink synchronizes remote SQL Anywhere and remote UltraLite databases.

One of the major differences is that there are no subscriptions in an UltraLite database. Another difference is that the MobiLink synchronization call is made directly on the connection object to the remote UltraLite database, rather than through an outside call to a separate synchronization utility.

UltraLite version
Although PocketBuilder 2.1 supports both UltraLite 10 and MobiLink 10, the objects created by the UltraLite Synchronization wizard still work with UltraLite 9 databases only. The wizard will be upgraded to work with UltraLite 10 databases in future releases of PocketBuilder.

Information required by the wizard

Table 16-9 shows the information required by the UltraLite Synchronization wizard. For information about preparing databases for synchronization, see the chapter on MobiLink synchronization in the *Resource Guide*.

Table 16-9: Information required by UltraLite Synchronization wizard

Option	Description
Application Library	PKL where the objects created by the wizard are stored
Publications	List of comma separated publications you want to synchronize
Script Version	The version of the script you want to use for synchronization. If you have not created scripts for synchronization events, you can leave this blank and check the Send Column Names box.
Send Column Names	Optional check box that is useful for testing if you have not created your own scripts for synchronization events
Communications Stream	Select the communications stream type that you want to use. ActiveSync is not supported as a communications stream type for synchronization from a PocketBuilder application
Host	Specify the numeric IP address for the MobiLink server. If you leave this blank, localhost is used as the server name. The IP address should be entered as a list of four sets of two or three digit numbers separated by periods, such as 199.99.001.01
Port	Specify a port for the MobiLink server. If you leave this blank, 2439 is used as the default port
Additional	Additional parameters can be added for the MobiLink server connection in the format: `keyword=value[;keyword=value...]`

Option	Description
Authentication Parameters	Specify a comma separated list of parameters that you want to pass for authentication purposes to the MobiLink server. PocketBuilder parses the list of parameters you enter, counts them, and passes them as an array in a structure created by the UltraLite Synchronization wizard
Names for the objects	You can specify names for the objects created by the wizard if you do not want to use the default names provided by PocketBuilder
Override Previous Synchronization Settings	Select this check box and assign a build number to the objects generated by the wizard to enable runtime overrides to client registry settings for MobiLink parameters entered by a user
Build Number	Specify a positive numeric value for the build number. To override the registry settings, the build number you assign must be higher than the build number in the registry, if there is one.

What the wizard generates

The UltraLite Synchronization wizard creates a nonvisual user object, a global external function, and three different structure objects for passing parameters, all of which make it easier to add database synchronization capabilities to your PocketBuilder target. By default, the wizard also adds a second global external function and two optional display windows.

Table 16-7 shows objects that can be generated by the wizard, listed by their default names, where *appname* stands for the name of the current application.

Table 16-10: Objects generated by UltraLite Synchronization wizard

Default name	Description
nvo_*appname*_ulsync	Nonvisual user object that starts synchronization from the remote client
gf_*appname*_ulsync	Global function that instantiates nvo_*appname*_ulsync to start the synchronization.This function includes the logic to start the synchronization with or without a feedback window
s_*appname*_ulsync_parms	Structure used to store runtime information, such as connection parameters entered by the user in the w_*appname*_ulsync_options window
s_*appname*_ulsync_info	Structure used in the Synchronize call to pass values of instance variables set in the nvo_*appname*_ulsync nonvisual user object

Default name	Description
s_*appname*_ulsync_results	Structure used to hold summary information from the synchronization process that is returned in the GetSynchronizeResult call
w_*appname*_ulsync or w_ulsync_default_feedback	Optional feedback window that can be used, instead of the standard MobiLink status window, to display synchronization status to the client. Even if you do not use the optional status window, a generated feedback window must be instantiated by the gf_*appname*_ulsync function
w_*appname*_ulsync_options	Optional window that allows the application user to change MobiLink connection arguments at runtime
gf_*appname*_configure_ulsync	Optional global function that calls the w_*appname*_ulsync_options window

Maintaining users and subscriptions in the remote database

Sync User and Subscription Maintenance wizard

A nonvisual object generated by the MobiLink Sync User and Subscription Maintenance wizard creates a database connection (with full or limited administration authority) that allows an application user to update MobiLink users and subscriptions in a remote SQL Anywhere database. A wizard screen allows you to set the database authority that you want to provide to the user for these administrative tasks.

Altogether the wizard generates six objects, which are described in Table 16-11. The *appname* variable in the default name for each object is replaced by the name of your target application. You can change the default names on the Name Generated Objects page of the wizard.

Table 16-11: Default names of objects generated by the wizard

Generated object	Description
w_*appname*_mluser	Window for entering MobiLink user information, including MobiLink server options that you enable for editing
w_*appname*_mlsubscription	Window for entering MobiLink subscription information, including MobiLink server options that you enable for editing

Generated object	Description
nvo_*appname*_mluser	Nonvisual user object with functions that connect to the remote database and open the user or subscription maintenance window
m_*appname*_mluser	Menu assigned to the user and subscription maintenance windows with menu items for propagating changes made in those windows
vo_*appname*_mluser	Visual user object included in the user and subscription maintenance windows to display values for MobiLink options and user and subscription information
d_*appname*_mluser	DataWindow included in the visual user object to display values for MobiLink options and user and subscription information

After you run the wizard, you can assign a global variable to the nonvisual user object generated by the wizard and instantiate the nonvisual object in an application script. You can then call the uf_run_mluser_window function of the instantiated object from the Clicked event of a User Maintenance menu item that you add to one of your application menus. Similarly, you can call the uf_run_mlsub_window function from the Clicked event of a Subscription Maintenance menu item that you add to an application menu.

Calling the functions of the nonvisual user object opens the designated maintenance window and connects to the remote database. Changes that the user makes to either maintenance window are propagated to the remote database by selections in the menu generated by the wizard. The generated menu object is associated with both maintenance windows.

Sample code for performing the post-wizard tasks is included in commented sections of the Script views for the nonvisual object and the two maintenance windows generated by the wizard.

Managing MobiLink synchronization on the server

You can start the MobiLink synchronization server and Sybase Central from the PocketBuilder UI.

Starting the MobiLink synchronization server

Before you synchronize remote databases with the consolidated database, you must start the MobiLink synchronization server. You can start the server from the Database or the Database Profiles dialog box in PocketBuilder.

❖ **To start the MobiLink synchronization server:**

1. From the Objects view of the Database painter or from the Database Profiles dialog box, expand the ODBC Utilities folder, and click MobiLink Synchronization server.

   The MobiLink Synchronization Server Options dialog box displays.

2. Select the MobiLink version and enter the ODBC connection string for your consolidated database.

   The values that populate the MobiLink version drop-down list come from the Adaptive Server Anywhere versions listed in the *hkey_local_machine\software\odbc\odbcinst.ini* registry key.

   The ODBC connection string should not contain any blank spaces that are not part of the data source name. The following is an example of an ODBC connection string for the SalesDB consolidated database:

   ```
   DSN=SalesDB;UID=dba;PWD=sql
   ```

3. Define other options as needed.

   For information about filling in specific fields in the dialog box, click the Help button in the dialog box. The Usage button opens a dialog box with information about command line options.

4. Click OK.

   When you click OK, PocketBuilder starts the MobiLink Synchronization server.

Using Sybase Central

You can use Sybase Central to manage MobiLink synchronization and create synchronization scripts that are held in the consolidated database. You can also use the SQL Anywhere plug-in to Sybase Central to add publications, synchronization users, and synchronization subscriptions to remote databases.

❖ **To start Sybase Central**

- From the Objects view of the Database painter or from the Database Profiles dialog box, expand the ODBC Utilities folder, and click Sybase Central. You can also launch Sybase Central from the UltraLite Utilities folder.

  Sybase Central displays.

❖ **To work with the consolidated database in Sybase Central**

- Right-click MobiLink Synchronization in the left pane of Sybase Central, select Connect from the pop-up menu, enter connection parameters in the Connect dialog box, and click OK.

  You can use Sybase Central to add scripts for database tables and select synchronization events that cause the script to be executed.

❖ **To work with remote databases in Sybase Central**

- Right-click SQL Anywhere 10 in the left pane of Sybase Central, select Connect from the pop-up menu, enter connection parameters in the Connect dialog box, and click OK.

  If you open the Publications and MobiLink Users folders under the MobiLink Synchronization Client folder for the remote database, you can add publications and synchronization users in the right pane of Sybase Central.

  After you add a publication and a synchronization user, you can create a synchronization subscription by right-clicking the publication in the right pane of Sybase Central, selecting Properties from the pop-up menu, clicking the Subscribe button on the Synchronization Subscriptions tab of the Publication Properties dialog box, and clicking OK.

For more information, see the chapter on MobiLink synchronization in the *Resource Guide*, or see the MobiLink books in the SQL Anywhere online Help. You can open the online Help by selecting the Help>SQL Anywhere 10> Help Topics menu in Sybase Central.

CHAPTER 17

Defining DataWindow Objects

About this chapter

The applications you build are centered around your organization's data. This chapter describes how to define DataWindow objects to display and manipulate the data.

Contents

About DataWindow objects

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

DataWindow objects have knowledge about the data they are retrieving. You can specify display formats, presentation styles, and other data properties so the data is used in the most meaningful way by users.

DataWindow object examples

You can display the data in the format that will best present the data to your users.

Edit styles

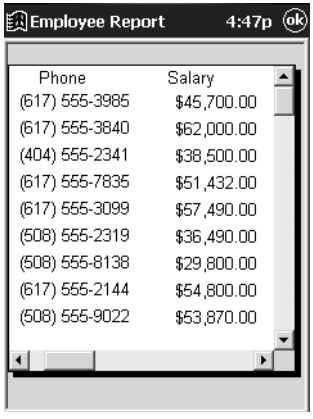
If a column can take only a small number of values, you can have the data appear as radio buttons in a DataWindow object so users know what their choices are.

Figure 17-1: DataWindow example with radio buttons

Display formats

If a column displays phone numbers, salaries, or dates, you can specify the format appropriate to the data.

Figure 17-2: DataWindow example with formatted data



Validation rules

If a column can take numbers only in a specific range, you can specify a simple validation rule for the data, without writing any code, to make sure users enter valid data.

Enhancing DataWindow objects

If you want to enhance the presentation and manipulation of data in a DataWindow object, you can include computed fields, pictures, and graphs that are tied directly to the data retrieved by the object.

How to use DataWindow objects

Before you can use a DataWindow object, you need to build the object. To do that you can go to the DataWindow painter, which lets you create and edit DataWindow objects.

This section describes the overall process for creating and using DataWindow objects. For more information about using DataWindow objects in different kinds of applications and writing code that interacts with DataWindow objects, see the *Resource Guide*.

❖ **To use DataWindow objects in an application:**

1 Create the DataWindow *object* using one of the DataWindow wizards on the DataWindow tab page of the New dialog box.

The wizard helps you define the data source, presentation style, and other basic properties of the object, and the DataWindow object displays in the DataWindow painter. In this painter, you define additional properties for the DataWindow object, such as display formats, validation rules, and sorting and filtering criteria.

For more information about creating a DataWindow object, see “Building a DataWindow object” on page 418.

2 Place a DataWindow *control* in a window or user object.

It is through this control that your application communicates with the DataWindow object you created in the DataWindow painter.

3 Associate the DataWindow control with the DataWindow object.

4 Set properties and write scripts in the Window painter to manipulate the DataWindow control and its contents.

For example, you use the PowerScript Retrieve function to retrieve data into the DataWindow control.

You can write scripts for the DataWindow control to deal with error handling, sharing data between DataWindow controls, and so on.

Choosing a presentation style

The presentation style you select for a DataWindow object determines the format PocketBuilder uses to display the DataWindow object in the Design view. You can use the format as displayed or modify it to meet your needs:

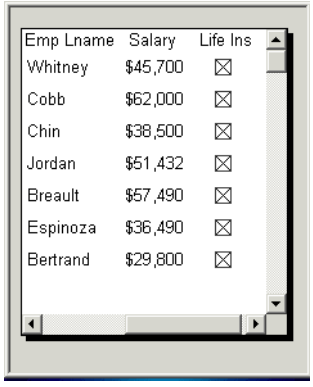
When you create a DataWindow object, you can choose from the following presentation styles:

Tabular
Freeform
Grid
Group
Graph

Using the Tabular style

The Tabular presentation style presents data with data columns going across the page and headers above each column. You can reorganize the default layout any way you want by moving columns and text. The number of rows that can display at one time is limited by the Windows CE screen. You can add scroll bars to the DataWindow control that holds the tabular DataWindow object.

Figure 17-3: DataWindow example showing a tablular report



Using the Freeform style

The Freeform presentation style presents data with data columns going down the page and labels next to each column. You can reorganize the default layout any way you want by moving columns and text. The Freeform style is often used for data entry forms. Figure 17-1 on page 414 is an example of a DataWindow with a Freeform presentation style. Typically you would use this style to present detailed information about a particular record (row) in the database.

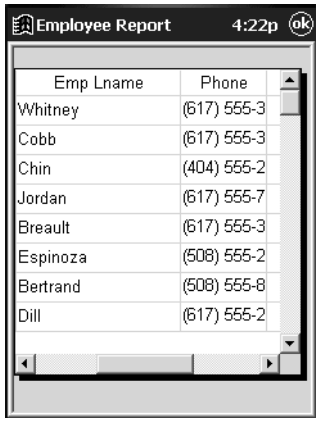
Using the Grid style

The Grid presentation style shows data in row-and-column format with grid lines separating rows and columns. With other styles, you can move text, values, and other objects around freely in designing the report. With the grid style, the grid lines create a rigid structure of cells.

An advantage of the Grid style is that users can reorder and resize columns during execution.

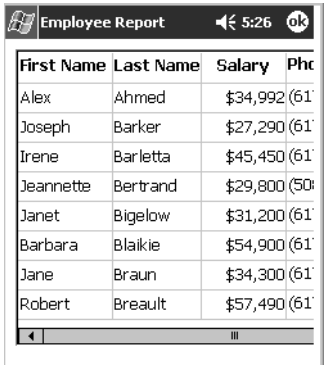
Original Grid report The grid report in Figure 17-4 shows employee information. Several of the columns have a large amount of extra white space.

Figure 17-4: Grid DataWindow



Grid report with modified column widths The grid report in Figure 17-5 was created from the original one by decreasing the width of some columns.

Figure 17-5: Grid DataWindow with column widths adjusted at runtime

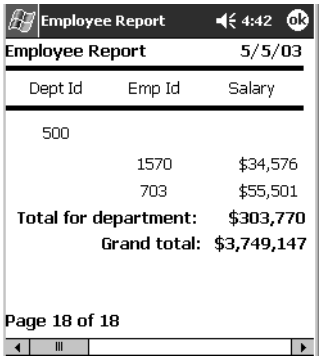


Using the Group presentation style

The Group presentation style provides an easy way to create grouped DataWindow objects, where the rows are divided into groups, each of which can have statistics calculated for it. Using this style generates a tabular DataWindow object that has grouping properties defined.

This report groups employees by department and lists each employee's salary. It also includes a subtotal for all department salaries in the trailer band for each group and a grand total for all salaries in the summary band.

Figure 17-6: Group DataWindow showing salaries by department



For more about the Group presentation style, see Chapter 22, "Filtering, Sorting, and Grouping Rows."

Using the Graph presentation styles

In addition to text-based presentation styles, PocketBuilder allows you to display information graphically using the Graph presentation style.

For more information about this presentation style, see Chapter 24, "Working with Graphs."

Building a DataWindow object

You use a wizard to build a new DataWindow object. To create a DataWindow object or use the DataWindow painter, you must be connected to the database whose data you will be accessing. When you open the DataWindow painter or select a data source in the wizard, PocketBuilder connects you to the DBMS and database you used last. If you need to connect to a different database, do so before working with a DataWindow object.

For information about changing your database connection, see the *Resource Guide*.

❖ **To create a new DataWindow object:**

1 Select File>New from the menu bar and select the DataWindow tab.

2 If there is more than one target in the workspace, select the target where you want the DataWindow to be created from the drop-down list at the bottom of the dialog box.

3 Choose a presentation style for the DataWindow object.

The presentation style determines how the data is displayed. See "Choosing a presentation style" on page 416. When you choose the presentation style, the appropriate DataWindow object wizard starts up.

4 If you want data to be retrieved in the Preview view when the DataWindow object opens, select the Retrieve on Preview check box.

5 Define the data source.

See "Selecting a data source" on page 419.

6 Choose options for the DataWindow object and click Next.

See "Choosing DataWindow object-wide options" on page 450.

7 Review your specifications and click Finish.

The DataWindow object displays in the Design view.

8 Save the DataWindow object in a library.

Selecting a data source

The data source you choose determines how you select the data that will be used in the DataWindow object.

About the term *data source*
The term *data source* used here refers to how you use the DataWindow painter to specify the data to retrieve into the DataWindow object.

If the data is in the database

If the data for the DataWindow object will be retrieved from a database, choose one of the types of data source listed in Table 17-1.

Table 17-1: Types of data source to be used in a DataWindow object

Data source	Use when
Quick Select	The data is from a single table (or from tables that are related through foreign keys) and you need to choose only columns, selection criteria, and sorting.
SQL Select	You want more control over the SQL SELECT statement generated for the data source *or* your data is from tables that are not connected through a key. For example, you need to specify grouping, computed columns, or retrieval arguments within the SQL SELECT statement.
Query	The data has been defined as a query.
Stored Procedure	The data is defined in a stored procedure.

If the data is not in a database

Select the External data source if:

- The DataWindow object will be populated programmatically
- Data will be imported from a DDE application
- Data will be imported from an external file, such as a tab-separated text file (*TXT* file) or a dBASE file (*DBF* file)

PocketBuilder

After you choose a data source in the various DataWindow wizards, you specify the data. The data source you choose determines what displays in the wizards and how you define the data.

Why use a DataWindow if the data is not from a DBMS

Even when the data does not come from the database, there are many times when you want to take advantage of the intelligence of a DataWindow object:

- **Data Validation** You have full access to validation rules for data.
- **Display Formats** You can use any existing display formats to present the data, or create your own.
- **Edit Styles** You can use any existing edit styles, such as radio buttons and edit masks, to present the data, or create your own.

Using Quick Select

The easiest way to define a data source is using Quick Select.

❖ **To define the data using Quick Select:**

1. Click Quick Select in the Choose Data Source dialog box in the wizard and click Next.
2. Select the table that you will use in the DataWindow object.

   For more information, see "Selecting a table" next.
3. Select the columns to be retrieved from the database.

   For more information, see "Selecting columns" on page 423.
4. (Optional) Sort the rows before you retrieve data.

   For more information, see "Specifying sorting criteria" on page 424.
5. (Optional) Select what data to retrieve.

   For more information, see "Specifying selection criteria" on page 424.
6. Click the OK button Quick Select dialog box.

   You return to the wizard to complete the definition of the DataWindow object.

Quick Select limitations

When you choose Quick Select as your data source, you cannot:

- Specify grouping before rows are retrieved
- Include computed columns
- Specify retrieval arguments for the SELECT statement that are supplied during execution.

To use these options when you create a DataWindow object, choose SQL Select as your data source. If you decide later that you want to use retrieval arguments, you can define them by modifying the data source. For more information, see Chapter 18, "Enhancing DataWindow Objects."

Selecting a table

Which tables and views display?

When you choose Quick Select, the Quick Select dialog box displays. The Tables box lists tables and views in the current database.

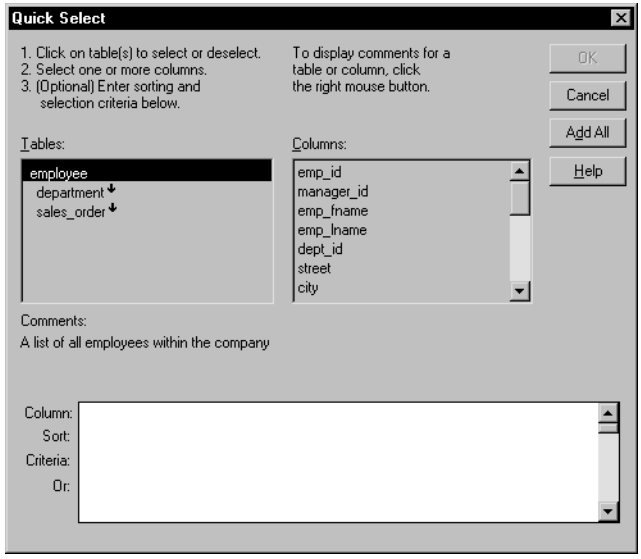
Displaying table comments
To display a comment about a table, position the pointer on the table and click the right mouse button, or select the table. The comment displays in the Quick Select dialog box below the list box for tables.

The DBMS determines what tables and views display. SQL Anywhere does not restrict the display, so all tables and views display, whether or not you have authorization.

Tables with key relationships

When you select a table, the table's column names display in the Columns box, and any tables having a key relationship with the selected table display in the Tables box. These tables are indented and marked with an arrow to show their relationship to the selected table. You can select any of these related tables if you want to include columns from them in the DataWindow object.

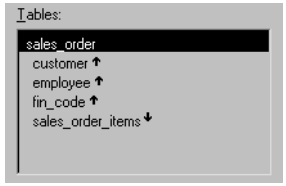
Figure 17-7: Table and column selection using Quick Select



Meaning of the up and down arrows

An arrow displays next to a table to indicate its relationship to the selected table. The arrow always points in the *many* direction of the relationship—toward the selected table (up) if the selected table contains a foreign key in the relationship and away from the selected table (down) if the selected table contains a primary key in the relationship.

Figure 17-8: Arrow direction indicating table key mappings



In Figure 17-8, sales_order is the selected table. The Up arrows indicate that a foreign key in the sales_order table is mapped to the primary key in the customer, employee, and fin_code tables. The Down arrow indicates that the sales_order_items table contains a foreign key mapped to the primary key in the sales_order table.

How columns from additional tables display

The column names of selected tables display in the Columns box. If you select more than one table, the column names are identified as:

tablename.columnname

For example, department.dept_name and employee.emp_id display when the Employee table and the Department table are selected.

To return to the original table list
Click the table you first selected at the top of the table list.

Selecting columns

You can select columns from the primary table and from its related tables. Select the table whose columns you want to use in the Tables box, then add columns from the Columns box:

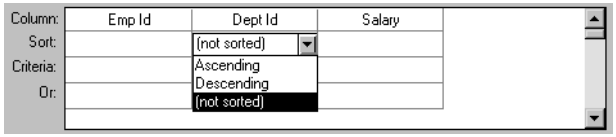
- To add a column, select it in the Columns box.
- To add all the columns that display in the Columns box, click Add All.
- To remove a column, deselect it in the Columns box.
- To view comments that describe a table or column, position the pointer on a table or column name, and press and hold the right mouse button.

As you select columns, they display in the grid at the bottom of the dialog box in the order in which you select them. If you want the columns to display in a different order in the DataWindow object, select a column name you want to move in the grid and drag it to the new location.

Specifying sorting criteria

In the grid at the bottom of the Quick Select dialog box, you can specify if you want the retrieved rows to be sorted. As you specify sorting criteria, PocketBuilder builds an ORDER BY clause for the SELECT statement.

Figure 17-9: Selecting columns for sort criteria



❖ **To sort retrieved rows on a column:**

1 Click in the Sort row for the column you want to sort on.

PocketBuilder displays a drop-down list.

2 Select the sorting order for the rows: Ascending or Descending.

Multilevel sorts

You can specify as many columns for sorting as you want. PocketBuilder processes the sorting criteria left to right in the grid: the first column with Ascending or Descending specified becomes the highest level sorting column, the next column with Ascending or Descending specified becomes the next level sorting column, and so on.

If you want to do a multilevel sort that does not match the column order in the grid, drag the columns to the correct order and then specify the columns for sorting.

Specifying selection criteria

You can enter selection criteria in the grid to specify which rows to retrieve. For example, instead of retrieving data about all employees, you might want to limit the data to employees in Sales and Marketing, or to employees in Sales who make more than $80,000.

As you specify selection criteria, PocketBuilder builds a WHERE clause for the SELECT statement.

❖ **To specify selection criteria:**

1 Click the Criteria row below the first column for which you want to select the data to retrieve.

2 Enter an expression, or if the column has an edit style, select or enter a value.

If the column is too narrow for the criterion, drag the grid line to enlarge the column. This enlargement does not affect the column size in a DataWindow object.

3 Enter additional expressions until you have specified the data you want to retrieve.

About edit styles
If a column has an edit style associated with it in the extended attribute system tables (an association made in the Database painter), the edit style is used in the grid selection criteria where possible. Drop-down list boxes are used for columns with code tables and columns that use the CheckBox and RadioButton edit styles.

SQL operators supported in Quick Select

You can use these SQL relational operators in the retrieval criteria:

Table 17-2: SQL relational operators used in retrieval criteria

Operator	Meaning
=	Is equal to (default operator)
>	Is greater than
<	Is less than
< >	Is not equal to
> =	Is greater than or equal to
< =	Is less than or equal to
LIKE	Matches this pattern
NOT LIKE	Does not match this pattern
IN	Is in this set of values
NOT IN	Is not in this set of values

Because = is the default operator, you can enter the value `100` instead of `= 100`, or the value `New Hampshire` instead of `= New Hampshire`.

Comparison operators

You can use the LIKE, NOT LIKE, IN, and NOT IN operators to compare expressions.

Use LIKE to search for strings that match a predetermined pattern. Use NOT LIKE to find strings that do not match a predetermined pattern. When you use LIKE or NOT LIKE, you can use wildcards:

- The percent sign (%), like the DOS wildcard asterisk (*), matches multiple characters. For example, `Good%` matches all names that begin with `Good`.
- The underscore character (_) matches a single character. For example, `Good _ _ _` matches all seven-letter names that begin with `Good`.

Use IN to compare and include rows with values that fall within the set of values that you define. Use NOT IN to compare and exclude rows with values that do not fall within the set of values that you define. For an example using the IN relational criteria, see "Example 4" on page 428. For an example using the LIKE relational criteria, see "Example 6" on page 429.

Connection operators

You can use the OR and AND logical operators to connect expressions.

Table 17-3: Logical operators for selection criteria expressions

Operator	Meaning
OR	The row is selected if one expression OR another expression is true
AND	The row is selected if one expression AND another expression are true

PocketBuilder makes some assumptions based on how you specify selection criteria. When you specify:

- Criteria for more than one column on one line, PocketBuilder assumes a logical AND between the criteria. A row from the database is retrieved if *all* criteria in the line are met.
- Two or more lines of selection criteria, PocketBuilder assumes a logical OR. A row from the database is retrieved if the criterion in *any* of the lines is met.

To override these defaults, begin an expression with the AND or OR operator. For an example overriding the default logical operators, see "Example 5" on page 428.

This technique is particularly handy when you want to retrieve a range of values in a column.

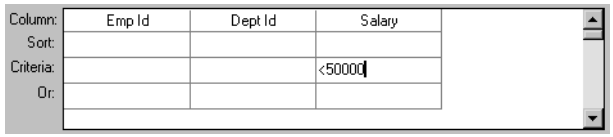
SQL expression examples

The examples in this section all refer to a grid that contains columns from a table of employees.

Example 1

The expression `>50000` in the Criteria row in the Salary column in the grid retrieves information for employees whose salaries are less than $50,000.

Figure 17-10: Expression criteria example using a relational operator



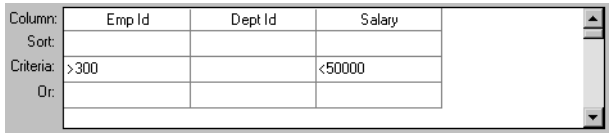
The SELECT statement that PocketBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE salary < 50000
```

Example 2

The expression `>300` in the Criteria row in the EmpId column and the expression `<50000` in the Criteria row in the Salary column in the grid retrieve information for employees whose employee IDs are greater than 300 *and* whose salaries are less than $50,000.

Figure 17-11: Expression criteria example using default logical operator



The SELECT statement that PocketBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE emp_id >300 AND salary <50000
```

Example 3

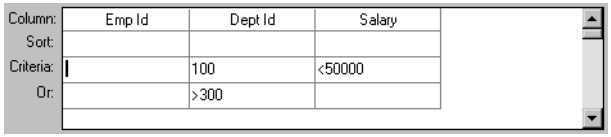
The expressions `100` in the Criteria row and `>300` in the Or row for the DeptId column, together with the expression `<50000` in the Criteria row in the Salary column, retrieve information for employees who belong to:

- Department 100 *and* have a salary less than $50,000

  or

- A department whose ID is greater than 300, regardless of their salary

Figure 17-12: Expression criteria example using two default logical operators



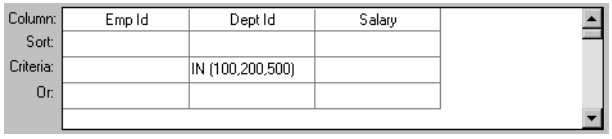
The SELECT statement that PocketBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE (dept_id = 100 AND salary < 50000)
OR dept_id > 300
```

Example 4

The expression `IN(100, 200, 500)` in the Criteria row in the DeptId column in the grid retrieves information for employees who are in department 100 *or* 200 *or* 500.

Figure 17-13: Expression criteria example using the IN relational operator



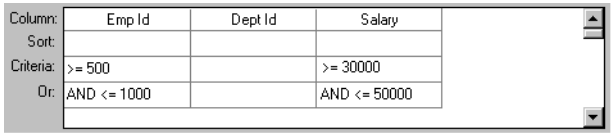
The SELECT statement that PocketBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id IN (100, 200, 500)
```

Example 5

This example shows the use of the word AND in the Or criteria row. In the Criteria row, `>=500` is in the EmpId column and `>=30000` is in the Salary column. In the Or row, `AND <=1000` is in the EmpId column and `AND <=50000` is in the Salary column. These criteria retrieve information for employees who have employee IDs from 500 to 1000 and salaries from $30,000 to $50,000.

Figure 17-14: Expression criteria example overriding the default logical operators



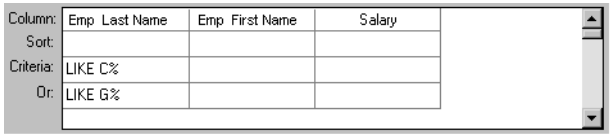
The SELECT statement that PocketBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE (emp_id >= 500 AND emp_id <= 1000)
AND (salary >= 30000 AND salary <= 50000)
```

Example 6

In a grid with three columns: Emp Last Name, Emp First Name, and Salary, the expressions `LIKE C%` in the Criteria row and `LIKE G%` in the Or row in the Emp Last Name column retrieve information for employees who have last names that begin with C or G.

Figure 17-15: Expression criteria example using the LIKE relational operator



The SELECT statement that PocketBuilder creates is:

```
SELECT emp_last_name, emp_first_name, salary
FROM employee
WHERE emp_last_name LIKE 'C%'
OR emp_last_name LIKE 'G%'
```

Providing SQL functionality to users

You can allow your users to specify selection criteria in a DataWindow object using these techniques during execution:

- You can automatically pop up a window prompting users to specify criteria each time just before data is retrieved

  For more information, see Chapter 18, “Enhancing DataWindow Objects.”

- You can place the DataWindow object in query mode using the Modify function

  For more information, see the *Resource Guide*.

Using SQL Select

In specifying data for a DataWindow object, you have more options for specifying complex SQL statements when you use SQL Select as the data source. When you choose SQL Select, you go to the Select painter, where you can paint a SELECT statement that includes the following:

- More than one table
- Selection criteria (WHERE clause)
- Sorting criteria (ORDER BY clause)
- Grouping criteria (GROUP BY and HAVING clauses)
- Computed columns
- One or more arguments to be supplied during execution

Saving your work as a query
While in the Select painter, you can save the current SELECT statement as a query by selecting File>Save Query from the menu bar. Doing so allows you to easily use this data specification again in other reports.

For more information about queries, see "Defining queries" on page 454.

❖ **To define the data using SQL Select:**

1 Click SQL Select in the Choose Data Source dialog box in the wizard and click Next.

The Select Tables dialog box displays.

2 Select the tables and/or views that you will use in the DataWindow object.

For more information, see "Selecting tables and views" next.

3 Select the columns to be retrieved from the database.

For more information, see "Selecting columns" on page 433.

4 Join the tables if you have selected more than one.

For more information, see "Joining tables" on page 436.

5 Select retrieval arguments if appropriate.

For more information, see "Using retrieval arguments" on page 438.

6 Limit the retrieved rows with WHERE, ORDER BY, GROUP BY, and HAVING criteria, if appropriate.

For more information, see "Specifying selection, sorting, and grouping criteria" on page 440.

7 If you want to eliminate duplicate rows, select Distinct from the Design menu. This adds the DISTINCT keyword to the SELECT statement.

8 Click the Return button on the PainterBar.

You return to the wizard to complete the definition of the DataWindow object.

Selecting tables and views

After you have chosen SQL Select, the Select Tables dialog box displays in front of the Table Layout view of the Select painter. What tables and views display in the dialog box depends on the DBMS. SQL Anywhere does not restrict the display, so all tables and views display, whether or not you have authorization.

❖ **To select the tables and views:**

- Do one of the following:
  - Click the name of each table or view you want to open

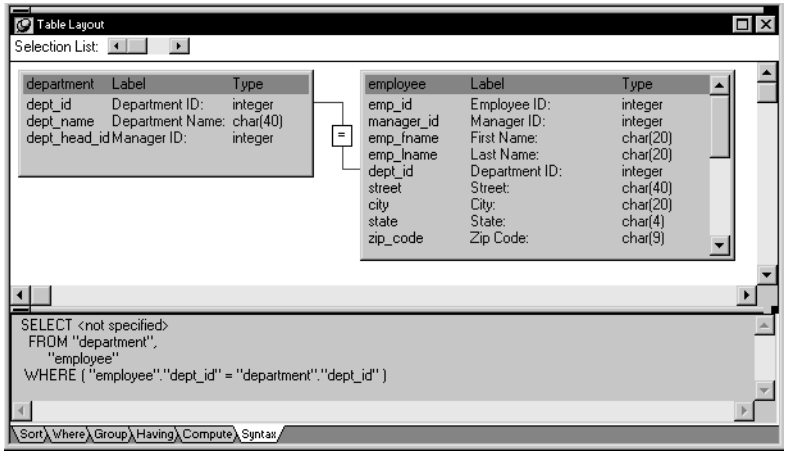
    Each table you select is highlighted. (To deselect a table, click it again.) Click the Open button to close the Select Tables dialog box.
  - Double-click the name of each table or view you want to open

    Each object opens immediately behind the Select Tables dialog box. Click the Cancel button to close the Select Tables dialog box.

Representations of the selected tables and views display. You can move or size each table to fit the space as needed.

Below the Table Layout view, several tabbed views also display by default. You will use the views (for example, Compute, Having, Group) to specify the SQL Select statement in more detail.You can turn the views on and off from the View menu on the menu bar.

Figure 17-16: Table and column selection using SQL Select



Specifying what is displayed

You can display the label and datatype of each column in the tables. (The label information comes from the extended attribute system tables.) If you need more space, you can choose to hide this information.

❖ **To hide or display comments, datatypes, and labels:**

1 Position the pointer on any unused area of the Table Layout view and select Show from the pop-up menu.

A cascading menu displays.

2 Select or clear Datatypes, Labels, or Comments as needed.

Colors in the Select painter

In the Database painter, you select the colors used by the Select painter to display the Table Layout view background and table information. You can also set colors for the text and background components in the table header and detail areas.

For more information about specifying colors in the Database painter, see “Modifying database preferences” on page 363.

Adding and removing tables and views

You can add tables and views to your Table Layout view at any time. You can also remove individual tables and views from the Table Layout view, or clear them all at once and begin selecting a new set of tables.

Table 17-4: Adding tables and views in the Select painter

To do this	Do this
Add tables or views	Click the Tables button in the PainterBar and select tables or views to add
Remove a table or view	Display its pop-up menu and select Close
Remove all tables and views	Select Design>Undo All from the menu bar

How PocketBuilder joins tables

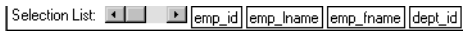
If you select more than one table in the Select painter, PocketBuilder joins columns based on their key relationship.

For information about joins, see "Joining tables" on page 436.

Selecting columns

You can click each column you want to include from the table representations in the Table Layout view. PocketBuilder highlights selected columns and places them in the Selection List at the top of the Select painter.

Figure 17-17: Column selection list in the Table Layout view



❖ **To reorder the selected columns:**

- Drag a column in the Selection List with the mouse. Release the mouse button when the column is in the proper position in the list.

❖ **To select all columns from a table:**

- Move the pointer to the table name and select Select All from the pop-up menu.

❖ **To include computed columns:**

1 Click the Compute tab to make the Compute view available (or select View>Compute if the Compute view is not currently displayed).

   Each row in the Compute view is a place for entering an expression that defines a computed column.

2 Enter one of the following:

- An expression for the computed column. For example:

```
salary/12
```

- A function supported by your DBMS. For example, the following is a SQL Anywhere function:

```
substr("employee"."emp_fname",1,2)
```

You can display the pop-up menu for any row in the Compute view. Using the pop-up menu, you can select and paste columns, functions, and arguments (if you have created any) into the expression.

About these functions
The functions listed in the pop-up menu are functions provided by your DBMS. They are not PocketBuilder functions. This is because you are now defining a SELECT statement that will be sent to your DBMS for processing.

3 Press the Tab key to get to the next row to define another computed column, or click another tab to make additional specifications.

PocketBuilder adds the computed columns to the list of columns you have selected.

About computed columns and computed fields

Computed columns you define in the Select painter are added to the SQL statement and used by the DBMS to retrieve the data. The expression you define here follows the rules of the DBMS.

You can also choose to define computed fields, which are created and processed dynamically by PocketBuilder after the data has been retrieved from the DBMS. There are advantages to doing this. For example, work is offloaded from the database server, and the computed fields update dynamically as data changes in the DataWindow object. If you have many rows, however, this updating can result in slower performance.

For more information, see Chapter 18, “Enhancing DataWindow Objects.”

Displaying the underlying SQL statement

As you specify the data for the DataWindow object in the Select painter, PocketBuilder is generating a SQL SELECT statement. It is this SQL statement that will be sent to the DBMS when you retrieve data into the DataWindow object. You can look at the SQL as it is being generated while you continue defining the data for the DataWindow object.

❖ **To display the SQL statement:**

- Click the Syntax tab to make the Syntax view available, or select View>Syntax if the Syntax view is not currently displayed.

  You may need to use the scroll bar to see all parts of the SQL SELECT statement. This statement is updated each time you make a change.

Editing the SELECT statement syntactically

Instead of modifying the data source graphically, you can directly edit the SELECT statement in the Select painter.

Converting from syntax to graphics
If the SQL statement contains unions or the BETWEEN operator, it may not be possible to convert the syntax back to graphics mode. In general, once you convert the SQL statement to syntax, you should maintain it in syntax mode.

❖ **To edit the SELECT statement:**

1. Select Design>Convert to Syntax from the menu bar.

   PocketBuilder displays the SELECT statement in a text window.

2. Edit the SELECT statement.

3. Do one of the following:

   - Select Design>Convert to Graphics from the menu bar to return to the Select painter.
   - Click the Return button to return to the wizard if you are building a new DataWindow object, or to the DataWindow painter if you are modifying an existing DataWindow object.

Joining tables

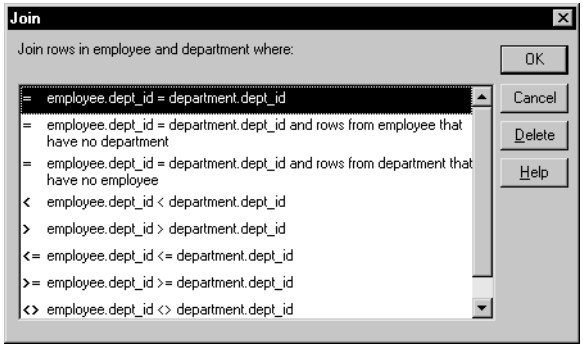
If the DataWindow object will contain data from more than one table, you should join the tables on their common columns. If you have selected more than one table, PocketBuilder joins columns according to whether they have a key relationship:

- Columns with a primary/foreign key relationship are joined automatically
- Columns with no key relationship are joined, if possible, based on common column names and types

Deleting joins

PocketBuilder links joined tables in the Select painter Table Layout view. PocketBuilder joins can differ depending on the order in which you select the tables, and sometimes the PocketBuilder best-guess join is incorrect, so you might need to delete a join and manually define a join.

Figure 17-18: Dialog box for manually deleting and adding table joins



❖ **To delete a join:**

1 Click the join operator connecting the tables.

The Join dialog box displays.

2 Click Delete.

Adding joins manually

You also add joins through the Join dialog box.

❖ **To join tables:**

1 Click the Join button in the PainterBar.

2 Click the columns on which you want to join the tables.

3 To create a join other than an equality join, click the join operator in the Table Layout view.

   The Join dialog box displays.

4 Select the join operator you want and click OK.

Using ANSI outer joins

PocketBuilderPocketBuilder supports both left and right outer joins in graphics mode in the Select painter, and full outer and inner joins in syntax mode.

The syntax for ANSI outer joins is generated according to the following BNF (Backus Naur form):

OUTER-join ::=
table-reference {LEFT | RIGHT} OUTER JOIN *table-reference* ON *search-condition*

table-reference ::=
table_view_name [*correlation_name*] | OUTER-join

Order of evaluation and nesting

In ANSI SQL-92, when nesting joins, the result of the first outer join (determined by order of ON conditions) is the operand of the outer join that follows it. In PocketBuilder, an outer join is considered to be nested if the *table-reference* on the left of the JOIN has been used before within the same outer join nested sequence.

The order of evaluation for ANSI syntax nested outer joins is determined by the order of the ON search conditions. This means that you must create the outer joins in the intended evaluation order and add nested outer joins to the end of the existing sequence, so that the second *table-reference* in the outer join BNF above will always be a *table_view_name*.

Nesting example

For example, if you create a left outer join between a column in Table1 and a column in Table2, then join the column in Table2 to a column in Table3, the product of the outer join between Table1 and Table2 is the operand for the outer join with Table3.

For ODBC connections, the default generated syntax encloses the outer joins in escape notation `{oj ...}` that is parsed by the driver and replaced with DBMS-specific grammar:

```
SELECT Table1.col1, Table2.col1, Table3.col1
FROM {oj {oj Table1 LEFT OUTER JOIN Table2 ON Table1.col1 =
Table2.col1}
LEFT OUTER JOIN Table3 ON Table2.col1 = Table3.col1}
```

Table references

Table references are considered equal when the table names are equal and there is either no alias (correlation name) or the same alias for both. Reusing the operand on the right is not allowed, because ANSI does not allow referencing the *table_view_name* twice in the same statement without an alias.

Determining left and right outer joins

When you create a join condition, the table you select first in the painter is the left operand of the outer join. The table that you select second is the right operand. The condition you select from the Joins dialog box determines whether the join is a left or right outer join.

For example, suppose you select the dept_id column in the employee table, then select the dept_id column in the department table, then choose the following condition:

```
employee.dept_id = department.dept_id and rows from
department that have no employee
```

The syntax generated is:

```
SELECT employee.dept_id, department.dept_id
FROM {oj "employee" LEFT OUTER JOIN "department" ON
"employee"."dept_id" = "department"."dept_id"}
```

If you select the condition with `rows from department that have no employee`, you create a right outer join instead.

Equivalent statements
The syntax generated when you select table A then table B and create a left outer join is equivalent to the syntax generated when you select table B then table A and create a right outer join.

For more about outer joins, see your DBMS documentation.

Using retrieval arguments

If you know which rows will be retrieved into the DataWindow object during execution—that is, if you can fully specify the SELECT statement without having to provide a variable—you do not need to specify retrieval arguments.

Adding retrieval arguments

If you decide later that you need arguments, you can return to the Select painter to define the arguments.

Defining retrieval arguments in the DataWindow painter
You can select View>Column Specifications from the menu bar. In the Column Specification view, a column of check boxes next to the columns in the data source lets you identify the columns users should be prompted for. This, like the Retrieval Arguments prompt, calls the Retrieve function.

See Chapter 18, "Enhancing DataWindow Objects."

If you want the user to be prompted to identify which rows to retrieve, you can define retrieval arguments when defining the SQL SELECT statement. For example, consider these situations:

- Retrieving the row in the Employee table for an employee ID entered into a text box. You must pass that information to the SELECT statement as an argument during execution.
- Retrieving all rows from a table for a department selected from a drop-down list. The department is passed as an argument during execution.

Using retrieval arguments during execution
If a DataWindow object has retrieval arguments, call the Retrieve function of the DataWindow control to retrieve data during execution and pass the arguments in the function.

For more information, see the *DataWindow Reference* in the online Help.

❖ **To define retrieval arguments:**

1. In the Select painter, select Design>Retrieval Arguments from the menu bar.
2. Enter a name and select a datatype for each argument.

   You can enter any valid SQL identifier for the argument name. The position number identifies the argument position in the Retrieve function you code in a script that retrieves data into the DataWindow object.
3. Click Add to define additional arguments as needed and click OK when done.

Specifying an array as a retrieval argument

You can specify an array of values as your retrieval argument. Choose the type of array from the Type drop-down list in the Specify Retrieval Arguments dialog box. You specify an array if you want to use the IN operator in your WHERE clause to retrieve rows that match one of a set of values. For example:

```
SELECT * from employee
WHERE dept_id IN (100, 200, 500)
```

retrieves all employees in department 100, 200, or 500. If you want your user to specify the list of departments to retrieve, you define the retrieval argument as a number array (such as *100, 200, 500*).

In the script that does the retrieval, you declare an array and reference it in the Retrieve function, such as:

```
int x[3]
// Now populate the array with values
// such as x[1] = sle_dept.Text, and so on
// then retrieve the data, as follows.
dw_1.Retrieve(x)
```

PocketBuilder passes the appropriate comma-delimited list to the function (such as *100, 200, 500* if x[1] = 100, x[2] = 200, and x[3] = 500).

When building the SELECT statement, you reference the retrieval arguments in the WHERE or HAVING clause, as described in the next section.

Specifying selection, sorting, and grouping criteria

In the SELECT statement associated with a DataWindow object, you can add selection, sorting, and grouping criteria that are included in the SQL statement and processed by the DBMS as part of the retrieval.

Table 17-5: Adding selection, sorting, and grouping criteria to the SELECT statement

To do this	Use this clause
Limit the data that is retrieved from the database	WHERE
Sort the retrieved data before it is brought into the DataWindow object	ORDER BY
Group the retrieved data before it is brought into the DataWindow object	GROUP BY
Limit the groups specified in the GROUP BY clause	HAVING

Dynamically selecting, sorting, and grouping data
Selection, sorting, and grouping criteria that you define in the Select painter are added to the SQL statement and processed by the DBMS as part of the retrieval. You can also define selection, sorting, and grouping criteria that are created and processed dynamically by PocketBuilder after data has been retrieved from the DBMS.

For more information, see Chapter 22, "Filtering, Sorting, and Grouping Rows."

Referencing retrieval arguments

If you have defined retrieval arguments, you reference them in the WHERE or HAVING clause. In SQL statements, variables (called host variables) are always prefaced with a colon to distinguish them from column names.

For example, if the DataWindow object is retrieving all rows from the Department table where the dept_id matches a value provided by the user during execution, your WHERE clause will look something like this:

```
WHERE dept_id = :Entered_id
```

where Entered_id was defined previously as an argument in the Specify Retrieval Arguments dialog box.

Referencing arrays
Use the IN operator and reference the retrieval argument in the WHERE or HAVING clause.

For example, if you reference an array defined as deptarray, the expression in the WHERE view might look like this:

```
"employee.dept_id" IN (:deptarray)
```

You need to supply the parentheses yourself.

Defining WHERE criteria

You can limit the rows that are retrieved into the DataWindow object by specifying selection criteria that correspond to the WHERE clause in the SELECT statement.

For example, if you are retrieving information about employees, you can limit the employees to those in Sales and Marketing, or to those in Sales and Marketing who make more than $50,000.

❖ **To define WHERE criteria:**

1 Click the Where tab to make the Where view available (or select View>Where if the Where view is not currently displayed).

Each row in the Where view is a place for entering an expression that limits the retrieval of rows.

2 Click in the first row under Column to display columns in a drop-down list, or select Columns from the pop-up menu.

3 Select the column you want to use in the left-hand side of the expression.

The equality (=) operator displays in the Operator column.

Using a function or retrieval argument in the expression
To use a function, select Functions from the pop-up menu and click a listed function. These are the functions provided by the DBMS.

To use a retrieval argument, select Arguments from the pop-up menu. You must have defined a retrieval argument already.

4 (Optional) Change the default equality operator.

Enter the operator you want, or click to display a list of operators and select an operator.

5 Under Value, specify the right-hand side of the expression. You can:

- Type a value.
- Paste a column, function, or retrieval argument (if there is one) by selecting Columns, Functions, or Arguments from the pop-up menu.
- Paste a value from the database by selecting Value from the pop-up menu, then selecting a value from the list of values retrieved from the database. (It may take some time to display values if the column has many values in the database.)
- Define a nested SELECT statement by selecting Select from the pop-up menu. In the Nested Select dialog box, you can define a nested SELECT statement. Click Return when you have finished.

6 Continue to define additional WHERE expressions as needed.

For each additional expression, select a logical operator (AND or OR) to connect the multiple boolean expressions into one expression that PocketBuilder evaluates as true or false to limit the rows that are retrieved.

7 Define sorting (Sort view), grouping (Group view), and limiting (Having view) criteria as appropriate.

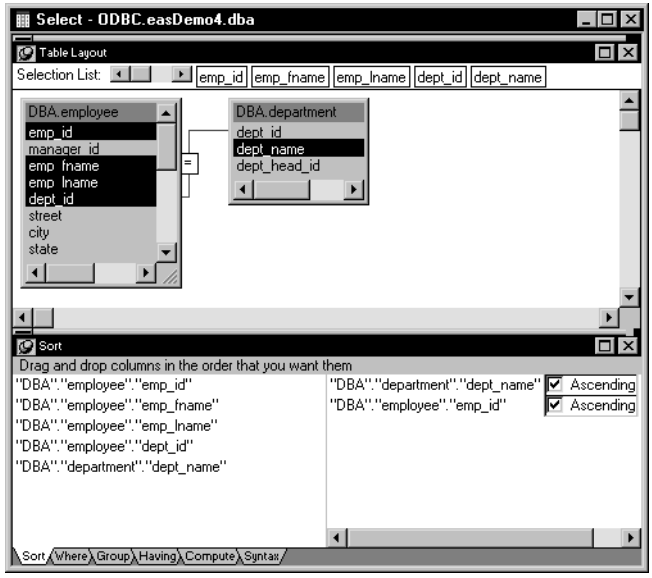
8 Click the Return button to return to the DataWindow painter.

Defining ORDER BY criteria

You can sort the rows that are retrieved into the DataWindow object by specifying columns that correspond to the ORDER BY clause in the SELECT statement.

For example, if you are retrieving information about employees, you can sort on department. Then within each department, you can sort on employee ID.

Figure 17-19: Defining multiple sort criteria for a DataWindow object



❖ **To define ORDER BY criteria:**

1 Click the Sort tab to make the Sort view available (or select View>Sort if the Sort view is not currently displayed).

The columns you selected display in the order of selection. You might need to scroll to see your selections.

2 Drag the first column you want to sort on to the right side of the Sort view.

This specifies the column for the first level of sorting. By default, the column is sorted in ascending order. To specify descending order, clear the Ascending check box.

3 Continue to specify additional columns for sorting in ascending or descending order as needed.

   You can change the sorting order by dragging the selected column names up or down.

4 Define limiting (Where view), grouping (Group view), and limiting by group (Having view) criteria as appropriate.

5 Click the SQL Select button to return to the DataWindow painter.

Defining GROUP BY criteria

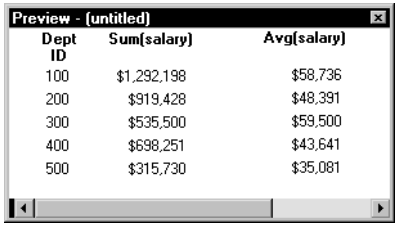
You can group the retrieved rows by specifying groups that correspond to the GROUP BY clause in the SELECT statement. This grouping happens before the data is retrieved into the DataWindow object. Each group is retrieved as one row into the DataWindow object.

For example, if in the SELECT statement you group data from the Employee table by department ID, you will get one row back from the database for every department represented in the Employee table. You can also specify computed columns, such as total and average salary, for the grouped data. This is the corresponding SELECT statement:

```
SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
```

If you specify this with the Employee table in the ASA Sample database, you get five rows back, one for each department.

Figure 17-20: Computed columns grouped by department ID



For more about GROUP BY, see your DBMS documentation.

❖ **To define GROUP BY criteria:**

1 Click the Group tab to make the Group view available (or select View>Group if the Group view is not currently displayed).

   The columns in the tables you selected display in the left side of the Group view. You might need to scroll to see your selections.

2 Drag the first column you want to group on to the right side of the Group view.

This specifies the column for grouping. Columns are grouped in the order they are displayed in the right side of the Group view.

3 Continue to specify additional columns for grouping within the first grouping column as needed.

To change the grouping order, drag the column names in the right side to the positions you want.

4 Define sorting (Sort view), limiting (Where view), and limiting by group (Having view) criteria as appropriate.

5 Click the Return button to return to the DataWindow painter.

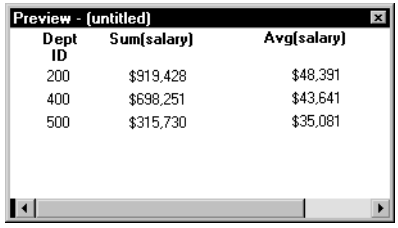
Defining HAVING criteria

If you have defined groups, you can define HAVING criteria to restrict the retrieved groups. For example, if you group employees by department, you can restrict the retrieved groups to departments whose employees have an average salary of less than $50,000. This corresponds to:

```
SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
HAVING avg(salary) < 50000
```

If you specify this with the Employee table in the ASA Sample database, you will get three rows back, because there are three departments that have average salaries less than $50,000.

Figure 17-21: Grouped data restricted by HAVING criteria



❖ **To define HAVING criteria:**

- Click the Having tab to make the Having view available (or select View>Having if the Having view is not currently displayed).

  Each row in the Having view is a place for entering an expression that limits which groups are retrieved. For information on how to define criteria in the Having view, see the procedure in “Defining WHERE criteria” on page 441.

Using Query

When you choose Query as the data source, you select a predefined SQL SELECT statement (a query) as specifying the data for your DataWindow object.

❖ **To define the data using Query:**

1 While using any of the DataWindow wizards, click Query in the Choose Data Source dialog box. Then click Next.

  The Select Query dialog box displays.

2 Type the name of a query or use the Browse button to find the query. Then click Next.

3 Finish interacting with the DataWindow wizard as needed for the presentation style you are using.

To learn how to create queries, see “Defining queries” on page 454.

Using External

If the data for the DataWindow object does not come from a database, specify External as the data source. You then specify the data columns and their types so that PocketBuilder can build the appropriate DataWindow object to hold the data. These columns make up the result set. PocketBuilder places the columns you specified in the result set in the DataWindow object.

❖ **To define the data using External:**

1. Click External in the Choose Data Source dialog box in the wizard and click Next.

   The Define Result Set dialog box displays for you to specify the first column in the result set.

2. Enter the name and type of the column.

   Available datatypes are listed in the drop-down list.

3. Click Add to enter the name and type of any additional columns you want in the result set.

4. Click Next when you have added all the columns you want.

What you do next

In a script, you need to tell PocketBuilder how to get data into the DataWindow object in your application. Typically, you import data during execution using a function (such as ImportFile and ImportString) or do some data manipulation and use the SetItem function to populate the DataWindow.

For more about these functions, see the *DataWindow Reference* in the online Help.

You can also import data values from an external file into the DataWindow object or report.

❖ **To import the data values from an external file:**

1. Make sure the Preview view of the DataWindow object is selected.

2. Select Rows>Import from the menu bar.

   The Select Import File dialog box displays.

3. Select the type of files to list from the List Files of Type drop-down list (either *TXT* or *DBF* files).

4. Select or type the name of the file you want to import and click OK.

Using Stored Procedure

A stored procedure is a set of precompiled and preoptimized SQL statements that performs some database operation. Stored procedures reside where the database resides, and you can access them as needed.

Defining data using a stored procedure

You can specify a stored procedure as the data source for a DataWindow object.

❖ **To define the data using Stored Procedure:**

1 Select Stored Procedure in the Choose Data Source dialog box in the wizard and click Next.

The Select Stored Procedure dialog box displays a list of the stored procedures in the current database.

2 Select a stored procedure from the list.

To list system procedures, select the System Procedure check box.

The syntax of the selected stored procedure displays below the list of stored procedures.

3 Specify how you want the result set description built:

- To build the result set description automatically, clear the Manual Result Set check box and click Next

  PocketBuilder executes the stored procedure and builds the result set description for you

- To define the result set description manually, select the Manual Result Set check box and click Next

  In the Define Stored Procedure Result Set dialog box:

  - Enter the name and type of the first column in the result set
  - To add additional columns, click Add

Your preference is saved
PocketBuilder records your preference for building result set descriptions for stored procedure DataWindow objects in the variable *Stored_Procedure_Build* in the PocketBuilder initialization file. If this variable is set to 1, PocketBuilder automatically builds the result set; if the variable is set to 0, you are prompted to define the result set description.

4 Continue in the DataWindow wizard as needed for the presentation style you are using, and click Finish.

When you have finished interacting with the wizard, you go to the DataWindow painter with the columns specified in the result set placed in the DataWindow object.

For information about defining retrieval arguments for DataWindow objects, see Chapter 18, "Enhancing DataWindow Objects."

For information about using a stored procedure to update the database, see "Using stored procedures to update the database" on page 522.

Editing a result set description

After you create a result set that uses a stored procedure, you can edit the result set description from the DataWindow painter.

❖ **To edit the result set description:**

1 Select Design>Data Source from the menu bar.

This displays the Column Specification view if it is not already displayed.

2 Select Stored Procedure from the Column Specification view's pop-up menu.

The Modify Stored Procedure dialog box displays.

3 Edit the Execute statement, select another stored procedure, or add arguments.

The syntax is:

execute *sp_procname*;*num arg1* = :*arg1*, *arg2* = :*arg2*..., *argn* =:*argn*

where *sp_procname* is the name of the stored procedure, *num* is the stored procedure group suffix, and *arg1*, arg1, and *argn* are the stored procedure's arguments.

The group suffix is an optional integer used to group procedures of the same name so that they can be dropped together with a single DROP PROCEDURE statement.

4 When you have defined the entire result set, click OK.

You return to the DataWindow painter with the columns specified in the result set placed in the DataWindow object.

For information about defining retrieval arguments for DataWindow objects, see Chapter 18, "Enhancing DataWindow Objects."

Choosing DataWindow object-wide options

You can set the default options, such as colors and borders, that PocketBuilder uses in creating the initial draft of a DataWindow object.

DataWindow generation options are for styles that use a layout made up of bands, which include Freeform, Grid, Tabular, and Group. PocketBuilder maintains a separate set of options for each of these styles.

When you first create a DataWindow object with one of these styles, you can choose options in the wizard and save your choices as the future defaults for the style you select.

Table 17-6: Properties for the DataWindow presentation style that you select

Property	Meaning for the DataWindow object
Background color	The default color for the background.
Text border and color	The default border and color used for labels and headings.
Column border and color	The default border and color used for data values.
Wrap Height (Freeform only)	The height of the detail band. When the value is None, the number of columns selected determines the height of the detail band. The columns display in a single vertical line. When the value is set to a number, the detail band height is set to the number specified and columns wrap within the detail band.

❖ **To specify default colors and borders for a style:**

1 Select Design>Options from the menu bar.

The DataWindow Options dialog box displays.

2 Select the Generation tab page if it is not on top.

3 Select the presentation style you want from the Presentation Style drop-down list.

The values for properties shown on the page are for the currently selected presentation style.

4 Change one or more of the presentation style properties and click OK.

PocketBuilder saves your generation option choices as the defaults to use when creating a DataWindow object with the same presentation style.

Generating and saving a DataWindow object

When you have finished interacting with the wizard, PocketBuilder generates the DataWindow object and opens the DataWindow painter.

When generating the DataWindow object, PocketBuilder may use information from a set of tables called the extended attribute system tables. If this information is available, PocketBuilder uses it.

About the extended attribute system tables and DataWindow objects

The extended attribute system tables are a set of tables maintained by the Database painter. They contain information about database tables and columns. Extended attribute information extends database definitions by recording information that is relevant to using database data in screens and reports.

For example, labels and headings you defined for columns in the Database painter are used in the generated DataWindow object. Similarly, if you associated an edit style with a column in the Database painter, that edit style is automatically used for the column in the DataWindow object.

When generating a DataWindow object, PocketBuilder uses the information from the extended attribute system tables listed in Table 17-7.

Table 17-7: Information from the extended attribute system tables

For	PocketBuilder uses
Tables	Fonts specified for labels, headings, and data
Columns	Text specified for labels and headings Display formats Validation rules Edit styles

If there is no extended attribute information for the database tables and columns you are using, you can set the text for headings and labels, the fonts, and the display formats in the DataWindow painter. The difference is that you have to do this individually for every DataWindow object that you create using the data.

If you want to change something that came from the extended attribute system tables, you can change it in the DataWindow painter. The changes you make in the DataWindow painter apply only to the DataWindow object you are working on.

The advantage of using the extended attribute system tables is that it saves time and ensures consistency. You have to specify the information only once, in the database. Since PocketBuilder uses the information whenever anyone creates a new DataWindow object with the data, it is more likely that the appearance and labels of data items will be consistent.

For more information about the extended attribute system tables, see Chapter 16, “Managing the Database,” and Appendix A, “Extended Attribute System Tables.”

Using extended attributes with an UltraLite application

UltraLite databases do not support system tables, so no extended attributes can be defined in an UltraLite database. If you want to use extended attributes in an UltraLite application, follow these steps:

1 Define extended attributes while you are connected to a consolidated SQL Anywhere database through an ODBC connection.

2 Create the DataWindow objects that you need in your UltraLite application while connected to the consolidated SQL Anywhere database. The extended attributes are stored in the definitions of the DataWindow objects, which can be used with any database connection.

3 Use the DataWindow objects created using SQL Anywhere in an application designed for UltraLite deployment.

This technique lets you take advantage of PocketBuilder extended attributes without the overhead of additional tables in the UltraLite database.

Saving the DataWindow object

When you have created a DataWindow object, you should save it. The first time you save it, you give it a name. As you work, you should save your DataWindow object frequently so that you do not lose changes.

The DataWindow object name can be any valid PowerBuilder identifier up to 40 contiguous characters. A common convention is to prefix the name of the DataWindow object with `d_`.

For information about valid identifiers, see “identifier names” in the online Help.

❖ **To save the DataWindow object:**

1. Select File>Save from the menu bar.

   If you have previously saved the DataWindow object, PocketBuilder saves the new version in the same library and returns you to the DataWindow painter.

   If you have not previously saved the DataWindow object, PocketBuilder displays the Save DataWindow dialog box.

2. (Optional) Enter comments in the Comments box to describe the DataWindow object.
3. Enter a name for the DataWindow object in the DataWindows box.
4. Specify the library in which the DataWindow object is to be saved and click OK.

Modifying an existing DataWindow object

❖ **To modify an existing DataWindow object:**

1. Select File>Open from the menu bar.

   The Open dialog displays.

2. Select the object type and the library.

   PocketBuilder lists the DataWindow objects in the current library.

3. Select the object you want.

   PocketBuilder opens the DataWindow painter and displays the DataWindow object.You can also open a DataWindow object by double-clicking it in the System Tree, or, if it has been placed in a DataWindow control window or visual user object, by selecting Modify DataWindow from the control's pop-up menu.

   To learn how you can modify an existing DataWindow object, see Chapter 18, "Enhancing DataWindow Objects."

Defining queries

A query is a SQL SELECT statement created in the Query painter and saved with a name so that it can be used repeatedly as the data source for a DataWindow object.

Queries save time, because you specify all the data requirements just once. For example, you can specify the columns, which rows to retrieve, and the sorting order in a query. Whenever you want to create a DataWindow object using that data, simply specify the query as the data source.

❖ **To define a query:**

1 Select File>New from the menu bar.

2 In the New dialog box, select the Database tab.

3 Select the Query icon and click OK.

4 Select tables in the Select Tables dialog box and click Open.

You can select columns, define sorting and grouping criteria, define computed columns, and so on, exactly as you do when creating a DataWindow object using the SQL Select data source.

For more about defining the SELECT statement, see "Using SQL Select" on page 430.

Previewing the query

While creating a query, you can preview it to make sure it is retrieving the correct rows and columns.

❖ **To preview a query:**

1 Select Design>Preview from the menu bar.

PocketBuilder retrieves the rows satisfying the currently defined query in a grid-style DataWindow object.

2 Manipulate the retrieved data as you do in the Database painter in the Output view.

You can sort and filter the data, but you cannot insert or delete a row or apply changes to the database. For more about manipulating data, see Chapter 16, "Managing the Database."

3 When you have finished previewing the query, click the Close button in the PainterBar to return to the Query painter workspace.

Saving the query

You must supply a name for each query that you save. The query name can be any valid PowerBuilder identifier up to 40 characters. When you name queries, use a unique name to identify each one. A common convention is to use a two-part name: a standard prefix that identifies the object as a query (such as q_) and a unique suffix. For example, you might name a query that displays employee data q_emp_data.

For information about valid identifiers, see "identifier names" in the online Help.

❖ **To save a query:**

1 Select File>Save Query from the menu bar.

If you have previously saved the query, PocketBuilder saves the new version in the same library and returns you to the Query painter.

If you have not previously saved the query, PocketBuilder displays the Save Query dialog box.

2 Enter a name for the query in the Queries box.

3 (Optional) Enter comments to describe the query.

These comments display in the Library painter. It is a good idea to use comments to remind yourself and others of the purpose of the query.

4 Specify the library in which to save the query and click OK.

Modifying a query

❖ **To modify a query:**

1 Select File>Open from the menu bar.

2 Select the Queries object type and the query you want to modify, then click OK.

3 Modify the query as needed.

What's next

After you have generated your DataWindow object, you will probably want to preview it to see how it looks. After that, you might want to enhance the DataWindow object in the DataWindow painter before using it.

PocketBuilder provides many ways for you to make a DataWindow object easier to use and more informative for users.

See Chapter 18, “Enhancing DataWindow Objects,” next.

CHAPTER 18

Enhancing DataWindow Objects

About this chapter

Before you put a DataWindow object into production, you can enhance it to make the data easier to use and interpret. You do that in the DataWindow painter. This chapter describes basic enhancements you can make to a DataWindow object.

Contents

Related topics

Other ways to enhance DataWindow objects are covered in later chapters:

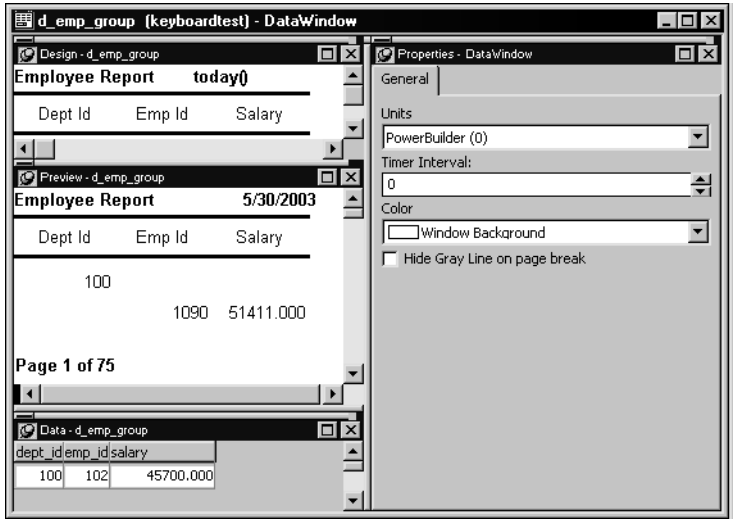
Chapter	Explains how to
Chapter 19, “Working with Controls in DataWindow Objects”	Add controls to a DataWindow object and reorganize, position, and rotate them
Chapter 20, “Controlling Updates in DataWindow Objects”	Control update capabilities
Chapter 21, “Displaying and Validating Data”	Specify display formats, edit styles, and validation rules for column data
Chapter 22, “Filtering, Sorting, and Grouping Rows”	Limit which rows are displayed, the order in which they are displayed, and whether they are divided into groups
Chapter 23, “Highlighting Information in DataWindow Objects”	Highlight data by using conditional expressions to modify the properties of controls in DataWindow objects
Chapter 24, “Working with Graphs”	Use graphs to visually present information retrieved in a DataWindow object

Working in the DataWindow painter

The DataWindow painter provides views related to the DataWindow object you are working on.

Figure 18-1 shows a DataWindow object in the DataWindow painter with the view title bars pinned. (The title bars are not pinned by default.)

Figure 18-1: DataWindow views with the title bars pinned



Design view

The Design view at the top left shows a representation of the DataWindow object and its controls. You use this view to design the layout and appearance of the DataWindow object. Changes you make are immediately shown in the Preview view and the Properties view.

Preview view

The Preview view in the middle on the left shows the DataWindow object with data as it will appear at runtime. If the Print Preview toggle is selected, you see the DataWindow object as it will appear when printed.

Properties view

The Properties view at the top right displays the properties for the currently selected control(s) in the DataWindow object, for the currently selected band in the DataWindow object, or for the DataWindow object itself. You can view and change the values of properties in this view.

Control List view

The Control List view in the stacked pane at the bottom lists all controls in the DataWindow object. Selecting controls in this view selects them in the Design view and the Properties view. You can also sort controls by Control Name, Type, or Tag. Because this view is in a stacked pane, and is not the view on top of the stack, you do not see the Control List view title bar.

Data view

The Data view in the stacked pane displays the data that can be used to populate a DataWindow object and allows manipulation of that data. Because this view is in focus, its title bar is visible at the top of the stack of panes.

Column Specifications view

The Column Specifications view in the stacked pane shows a list of the columns in the data source. For the columns, you can add, modify, and delete initial values, validation expressions, and validation messages. You can also specify that you want a column to be included in a prompt for retrieval criteria during data retrieval. To add a column to the DataWindow object, you can drag and drop the column from the Column Specifications view to the Design view. For external or stored procedure data sources, you can add, delete, and edit columns (column name, type, and length).

Understanding the DataWindow painter Design view

For the Tabular, Freeform, and Grid presentation styles, the DataWindow painter Design view is divided into areas called bands. Each band corresponds to a section of the displayed DataWindow object.

DataWindow objects with these presentation styles are divided into four bands: header, detail, summary, and footer. Each band is identified by a bar containing the name of the band above the bar and an Arrow pointing to the band. The Group presentation style has at least six bands, including a second header band for the group headers and a second detail band for the group details.

These bands can contain any information you want, including text, drawing controls, and computed fields containing aggregate totals.

Figure 18-2 shows the Design view for a DataWindow object with a Group presentation style.

Figure 18-2: Design view for a Group presentation style DataWindow

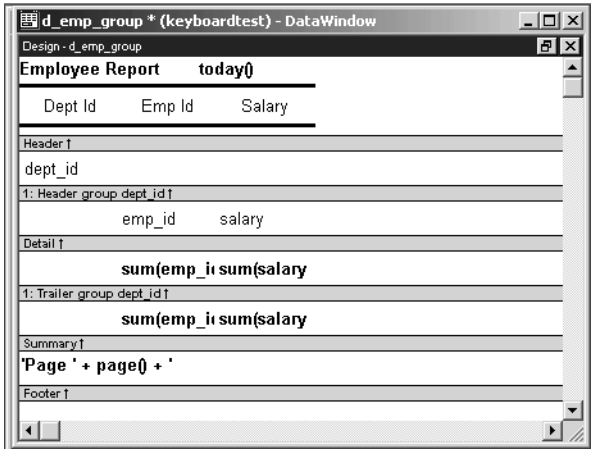


Table 18-1: Bands in the DataWindow painter Design view

Band	Used to display
Header	Information at the top of every screen or page, such as the name of the report or current date
Detail	Data from the database or other data source
Summary	Summary information that displays after all the data, such as totals and counts
Footer	Information displayed at the bottom of every page or screen, such as page number and page count

The header band

The header band contains heading information that is displayed at the top of every screen or page. The presentation style determines the contents of the header band:

- If the presentation style is Tabular or Grid, the headings defined for the columns in the Database painter display in the header band and the columns display on a single line across the detail band
- If the presentation style is Freeform, the header band is empty and labels display in the detail band next to each column

You can specify additional heading information (such as a date) in the header band, and you can include pictures and color to enhance the appearance of the band.

Displaying the current date
To include the current date in the header, you place a computed field that uses the Today DataWindow expression function in the header band. For information, see "Adding computed fields to a DataWindow object" on page 495.

The detail band

The detail band displays the retrieved data. The number of rows of data that display in the DataWindow object at one time is determined by the following expression:

(Height of the DataWindow object – *Height of headers and footers) / Height of the detail band*

The presentation style determines the contents of the detail band:

- If the presentation style is Tabular or Grid, the detail band displays column names, representing the columns
- If the presentation style is Freeform, the labels defined for the columns in the Database painter display in the detail band with boxes for the data to the right

How PocketBuilder names the columns in the Design view
If the DataWindow object uses one table, the names of the columns in the Design view are the same as the names in the table.

If the DataWindow object uses more than one table, the names of the columns in the Design view are *tablename_columnname*. PocketBuilder prefaces the name of the column with the table name to prevent ambiguity, since different tables can have columns with the same name.

When you design the detail band of a DataWindow object, you can specify display information for each column of the DataWindow object and add other controls, such as text, pictures, and drawing controls.

The summary and footer bands

You use the summary and footer bands of the DataWindow object the same way you use summary pages and page footers in a printed report:

- The contents of the summary band display at the end, after all the detail rows; this band often summarizes information in the DataWindow object
- The contents of the footer band display at the bottom of each screen or page of the DataWindow object; this band often displays the page number and name of the report

Using the DataWindow painter toolbars

The DataWindow painter contains three customizable PainterBars and a StyleBar.

For more information about using toolbars, see "Using toolbars" on page 39.

PainterBars

The PainterBars include buttons for standard operations (such as Save, Print, and Undo on PainterBar1), for common formatting operations (such as Currency, Percent, and Tab Order on PainterBar2), and for database operations (such as Retrieve and Insert Row on PainterBar3).

They also include six drop-down toolbars, which are indicated by a small black triangle on the right part of a button. Table 18-2 lists the drop-down toolbars that are available. The Controls toolbar is on PainterBar1. The other drop-down toolbars are on PainterBar2.

Table 18-2: Drop-down toolbars in the DataWindow painter

Toolbar	Used to
Background Color	Specify the background color of one or more selected controls.
Borders	Specify borders for one or more selected controls.
Controls	Specify controls to add to a DataWindow object.
Foreground Color	Specify the foreground color of one or more selected controls. In a text control, the foreground color specifies the color of the text.
Layout	Specify the alignment, sizing, and spacing of selected controls.
Slide	Specify sliding for controls.

StyleBar

The StyleBar includes buttons for applying properties (such as bold) to selected text elements.

Using the Properties view in the DataWindow painter

Each part of the DataWindow object (such as text, columns, computed fields, bands, even the DataWindow object itself) has a set of properties appropriate to the part. The properties display in the Properties view.

You can use the Properties view to modify the parts of the DataWindow object.

❖ **To use the Properties view to modify the parts of the DataWindow object:**

1 Position the mouse over the part you want to modify.

2 Display the part's pop-up menu and select Properties.

If it is not already displayed, the Properties view displays. The view displays the properties of the currently selected control(s), the band, or the DataWindow object itself. The contents of the Properties view change as different controls are selected (made current).

For example, the Properties view for a column has tabbed pages of information with properties specific to the selected column or columns. You access the properties you want to view or modify by clicking the appropriate tab. To choose an edit style for the column, you click the Edit tab. This brings the Edit page to the front of the Properties view.

Selecting controls in the DataWindow painter

The DataWindow painter provides several ways to select controls to act on. Table 18-3 describes the various methods for selecting controls.

Lasso selection
Use lasso selection when possible, because it is fast and easy. Lasso selection is another name for the method described in Table 18-3 for selecting neighboring controls.

When you select multiple controls, you can act on all the selected controls as a unit. For example, you can move all of them or change the fonts used to display text for all of them.

Table 18-3: Selecting controls in a DataWindow object

To select	Do this
A single control in a DataWindow object	Click the control in the Design view or in the Control List view.
Neighboring controls in a DataWindow object (lasso selection)	In the Design view, press and hold the left mouse button at one corner of the area containing the controls, drag the mouse over the neighboring controls you want to select, and release the mouse button. (In the Control List view, use the method for non-neighboring controls in a DataWindow object.)
Non-neighboring controls in a DataWindow object	In the Design or Control List views, click any of the controls that you want to select, then press and hold the Ctrl key while clicking the other controls you want to select.
Controls by type in a DataWindow object	• Select Edit>Select>Select All to select all controls • Select Edit>Select>Select Text to select all text controls • Select Edit>Select>Select Columns to select all columns
Controls by position in a DataWindow object	• Select Edit>Select>Select Above to select all controls above the currently selected control • Select Edit>Select>Select Below to select all controls below it • Select Edit>Select>Select Left to select all controls to the left of it • Select Edit>Select>Select Right to select all controls to the right of it

Displaying information about the selected control

The name, x and y coordinates, width, and height of the selected control are displayed in the MicroHelp bar. If multiple controls are selected, `Group Selected` displays in the Name area and the coordinates and size do not display.

Resizing bands in the DataWindow painter Design view

You can change the size of any band in the DataWindow object.

❖ **To resize a band in the DataWindow painter Design view:**

- Position the pointer on the bar representing the band and drag the bar up or down to shrink or enlarge the band

Using zoom in the DataWindow painter

You can zoom the display in and out in four views in the DataWindow painter: the Design view, Preview view, Data View, and Column Specifications view. For example, if you are working with a large DataWindow object, you can zoom out the Design view so you can see all of it on your screen, or you can zoom in on a group of controls to see their details better.

❖ **To zoom the display in the DataWindow painter:**

1 Click in the view you want to zoom.

You can zoom the Design view, Preview view, Data View, and Column Specifications view.

2 Select Design>Zoom from the menu bar.

3 Select a built-in zoom percentage, or set a custom zoom percentage by typing an integer in the Custom box.

Using the IntelliMouse pointing device
Using the IntelliMouse pointing device, users can also zoom in and out of a view by holding down the Ctrl key while rotating the wheel.

Undoing changes in the DataWindow painter

You can undo your change by pressing Ctrl+Z or selecting Edit>Undo from the menu bar. Undo requests affect all views.

Using the Preview view

You use the Preview view of the DataWindow painter to view a DataWindow object as it will appear with data and to test the processing that takes place in the DataWindow object.

❖ **To display the Preview view of a DataWindow object :**

- If the Preview view is not already displayed, select View>Preview from the menu bar.

In the Preview view, the bars that indicate the bands do not display, and, if you selected Retrieve on Preview in the DataWindow wizard, PocketBuilder retrieves all the rows from the database. You are prompted to supply arguments if you defined retrieval arguments.

As rows of data are being retrieved, the Retrieve button in the PainterBar changes to a Cancel button. You can click the Cancel button to stop the retrieval.

External DataWindow objects If the DataWindow object uses the External data source, no data is retrieved. You can import data, as described in "Importing data into a DataWindow object" on page 470.

DataWindow objects that have stored data If the DataWindow object has stored data in it, no data is retrieved from the database.

UltraLite connections

If you are connected to an UltraLite database and preview fails with a `Select error: SQLE_SYNTAX_ERROR` message, you might have opened a DataWindow object that was created using a different DBMS and that uses qualified SQL. The column names will be qualified with an owner name such as dba. UltraLite does not support table owners. There are two ways to remove the qualifiers:

- Select Design>Data Source and add an additional column to the column list, then remove it again. When you click the Return button on the PainterBar, PocketBuilder displays the message: `SELECT change has forced update specification change`. Click OK to rebuild the DataWindow syntax with unqualified SQL, then click Save.
- Close the DataWindow painter, select the DataWindow object in the System Tree, and select Edit Source from its pop-up menu. Place the edit cursor at the beginning of the editor, then s

elect Edit>Replace to replace all occurrences of the owner name and the period that follows it with an empty string. After you save and close the source, the DataWindow displays correctly in the Preview view.

Retrieving data

Where PocketBuilder gets data

PocketBuilder follows this order of precedence to supply the data in your DataWindow object:

1. If you have saved data in the DataWindow object, PocketBuilder uses the saved rows from the DataWindow object and does not retrieve data from the database.
2. PocketBuilder uses the data in the cache if there is any.
3. If there is no data in the cache yet, PocketBuilder retrieves data from the database automatically, with one exception. If the Retrieve on Preview option is off, you have to request retrieval explicitly, as described next.

Previewing without retrieving data

If you do not want PocketBuilder to retrieve data from the database automatically when the Preview view opens, you can clear the Retrieve on Preview option. The Preview view shows the DataWindow object without retrieving data.

❖ **To be able to preview without retrieving data automatically:**

1. Select Design>Options from the menu bar.

   The DataWindow Options dialog box displays.

2. Clear the Retrieve on Preview check box on the General page.

   When this check box is cleared, your request to preview the DataWindow object does not result in automatic data retrieval from the database.

Retrieve on Preview check box is available in the wizards
During the initial creation of a DataWindow object, you can set or clear the Retrieve on Preview option.

PocketBuilder uses data caching

When PocketBuilder first retrieves data, it stores the data internally. When PocketBuilder refreshes the Preview view, PocketBuilder displays the stored data instead of retrieving rows from the database again. This can save you a lot of time, since data retrieval can be time consuming.

How using data from the cache affects you

Because PocketBuilder accesses the cache and does not automatically retrieve data every time you preview, you might not have what you want when you preview. The data you see in preview and the data in the database can be out of sync.

For example, if you are working with live data that changes frequently, or with statistics based on changing data and you spend time designing the DataWindow object, the data you are looking at might no longer match the database. In this case, retrieve again to obtain the most current data.

Explicitly retrieving data

You can explicitly request retrieval at any time.

❖ **To retrieve rows from the database:**

- Do one of the following:
  - Click the Retrieve button in the PainterBar
  - Select Rows>Retrieve from the menu bar
  - Select Retrieve from the Preview view's pop-up menu

Supplying argument values or criteria
If the DataWindow object has retrieval arguments or is set up to prompt for criteria, you are prompted to supply values for the arguments or to specify criteria.

PocketBuilder retrieves the rows. As PocketBuilder retrieves, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval at any time.

Sharing data with the Data view

The Data view displays data that can be used to populate a DataWindow. When the ShareData pop-up menu item in the Data view is checked, changes you make in the Preview view and vice versa.

Modifying data

You can add, modify, or delete rows in the Preview view. When you have finished manipulating the data, you can apply the changes to the database.

If looking at data from a view or from more than one table
By default, you cannot update data in a DataWindow object that contains a view or more than one table.

For more about updating DataWindow objects, see Chapter 20, "Controlling Updates in DataWindow Objects."

Editing data

You can make changes to data directly in the Preview view. The Preview view uses validation rules, display formats, and edit styles that you have defined for the columns, either in the Database painter or in the current DataWindow object. Changes that you make to the data use the styles and formats selected for the column.

❖ **To modify existing data:**

- Tab to the field that has a value you want to change and enter the new value.

  To save the changes to the database, you must apply them as described in "Applying changes in an application" on page 469.

Adding a row in an application

Clicking the Insert Row button in the Preview view is equivalent to calling the InsertRow function and then the ScrollToRow function during execution.

❖ **To add a row:**

1 Click the Insert Row button.

  PocketBuilder creates a blank row.

2 Enter data for a row.

  To save the changes to the database, you must apply them as described in "Applying changes in an application" on page 469.

Deleting a row in an application

Clicking the Delete Row button in the Preview view is equivalent to calling the DeleteRow function during execution.

❖ **To delete a row:**

- Click the Delete Row button

  PocketBuilder removes the row from the display. To save the changes to the database, you must apply them as described in "Applying changes in an application" next.

Applying changes in an application

Clicking the Update Database button in the Preview view is equivalent to calling the Update function during execution.

❖ **To apply changes to the database:**

- Click the Update Database button.

  PocketBuilder updates the table with all the changes you have made.

Viewing row information

You can display information about the data you have retrieved.

❖ **To display the row information:**

- Select Rows>Described from the menu bar.

  The Describe Rows dialog box displays, showing the number of:

  - Rows that have been deleted in the painter but not yet deleted from the database
  - Rows displayed in the Preview view
  - Rows that have been filtered
  - Rows that have been modified in the painter but not yet modified in the database

All row counts are zero until you retrieve the data from the database or add a new row. The count changes when you modify the displayed data or test filter criteria.

Importing data into a DataWindow object

❖ **To import data into a DataWindow object:**

1. Select Rows>Import from the menu bar.

   Focus must be in the Preview view to enable the Rows>Import menu item.

2. Specify the file from which you want to import the data.

   The types of files that you can import into the painter display in the List Files of Type drop-down list.

3. Click Open.

   PocketBuilder reads the data from the file into the painter.

Data from file must match the DataWindow definition
When you import data from a file, the datatypes of the data must match, column for column, all the columns in the DataWindow definition (the columns specified in the SELECT statement), not just the columns that are displayed in the DataWindow object.

Using print preview

Previewing output for printing

You can print the data displayed in the Preview view. Before printing, you can preview the output on the screen. Your computer must have a default printer specified.

❖ **To preview printed output before printing:**

- Select File>Print Preview from the menu bar.

  Focus must be in the Preview view to enable the File>Print Preview menu item. Print Preview displays the DataWindow object as it will print.

Using the IntelliMouse pointing device
Using the IntelliMouse pointing device, users can scroll a DataWindow object by rotating the wheel. Users can also zoom a DataWindow object larger or smaller by holding down the Ctrl key while rotating the wheel.

Displaying rulers

You can choose whether to display rulers around page borders in the Print Preview.

❖ **To control the display of rulers in Print Preview:**

- Select or clear the File>Print Preview Rulers toggle switch in the menu bar.

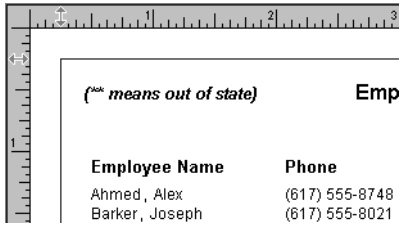
  The File>Print Preview Rulers menu item is enabled only if the File>Print Preview menu item is enabled and selected.

Changing margins

You can dynamically change margins while previewing a DataWindow object.

Figure 18-3 shows the left and top margin boundaries of a page containing a DataWindow object. The Print Preview view also displays boundaries for the right and bottom margins when you display the rulers in the view.

Figure 18-3: Displaying page margins in the Print Preview view



❖ **To change the margins in Print Preview:**

- Drag the margin boundaries on the rulers.

Zooming the page

You can reduce or enlarge the amount of the page that displays in the Print Preview view. This does not affect the printed output.

❖ **To zoom the page on the display screen:**

1 Select File>Print Preview Zoom from the menu bar.

2 Select the magnification you want and click OK.

The display of the page zooms in or out as appropriate. The size of the contents of the page changes proportionately as you zoom. This type of zooming affects your display but does not affect printing.

Zooming the contents

In addition to zooming the display on the screen, you can also zoom the contents, affecting the amount of material that prints on a page.

❖ **To zoom the contents of a DataWindow object with respect to the printed page:**

1 Select Design>Zoom from the menu bar.

2 Select the magnification you want and click OK.

The contents of the page zooms in or out as appropriate. If you enlarge the contents so they no longer fit, PocketBuilder creates additional pages as needed.

Printing data

You can print a DataWindow object at design time while the Preview view is displayed. You can print all pages, a range of pages, only the current page, or only odd or even pages. You can also specify whether you want multiple copies, collated copies, and printing to a file.

Avoiding blank pages and unwanted page breaks

To avoid multiple blank pages and other anomalies in printed reports, no row in the DataWindow object should be larger than the size of the target page. The page boundary is often reached in long text columns with AutoSizeHeight on. It can also be reached when detail rows are combined with page and group headers and trailers, or when they contain a column that has been resized to be larger than the page.

The summary band in a report is always printed on the same page as the last row of data. This means you sometimes find that there is a page break before the last row of data even if there is sufficient space to print the row. If you want the last row to print on the same page as the preceding rows, you must make the summary band small enough to fit on the page as well.

Printing a DataWindow object

You can choose File>Printer Setup from the menu bar to change printers or printer settings before you print a DataWindow object.

❖ **To print a DataWindow object:**

1. Select File>Print Report from the menu bar to display the Print dialog box.
2. Specify the number of copies to print.
3. Specify the pages: select All or Current Page, or type page numbers and page ranges in the Pages box.
4. Specify all pages, even pages, or odd pages in the Print drop-down list.
5. If you want to print to a file rather than to the printer, select the Print to File check box.
6. If you want to change the collating option, clear or select the Collate Copies check box and click OK.

   If you specified print to file, the Print to File dialog box displays.
7. Enter a file name and click OK.

   The extension *PRN* identifies it as a file prepared for the printer. Change the drive and directory if you want.

Working in a grid DataWindow object

If you are viewing a grid-style DataWindow object in the Preview view, you can make the following changes. Whatever you do in the Preview view is reflected in the Design view:

- Resize columns
- Reorder columns
- Copy data to the clipboard

❖ **To resize a column in a grid DataWindow object:**

1. Position the mouse pointer at a column boundary in the header.

   The pointer changes to a two-headed arrow.
2. Press and hold the left mouse button and drag the mouse to move the boundary.
3. Release the mouse button when the column is the correct width.

❖ **To reorder columns in a grid DataWindow object:**

1 Press and hold the left mouse button on a column heading.

PocketBuilder selects the column and displays a line representing the column border.

2 Drag the mouse left or right to move the column.

3 Release the mouse button.

❖ **To copy data to the clipboard from a grid DataWindow object:**

1 Select the cells whose data you want to copy to the clipboard:

- To select an entire column, click its header
- To select neighboring columns, press and hold Shift, then click the headers
- To select non-neighboring columns, press and hold Ctrl, then click the headers
- To select cells, press the left mouse button on the bottom border of a cell and drag the mouse over neighboring cells

Selected cells are highlighted.

2 Select Edit>Copy from the menu bar.

The contents of the selected cells or columns are copied to the clipboard. If you copied the contents of more than one column, the data is separated by tabs.

Saving data in an external file

While previewing, you can save the data retrieved in an external file. Note that the data and headers (if specified) are saved. Information in the footer or summary bands is not saved unless you are saving the data to a Powersoft report.

The file format you select for the external file determines whether PocketBuilder saves the data using ANSI or Unicode character sets.

For information about the relation between the file format you select and the character set in which data is saved, see the chapter on working with Unicode in the *Resource Guide*.

Saving the data in HTML Table format

One of the external file formats you can save data in is the HTML Table format. When you save in HTML Table format, PocketBuilder saves a style sheet along with the data. If you use this format, you can open the saved file in a Web browser. Once you have the file in HTML Table format, you can continue to enhance the file in HTML.

❖ **To save the data in a DataWindow object in an external file:**

1 Select File>Save Rows As from the menu bar.

The Save As dialog box displays.

2 Choose a format for the file from the Save As Type drop-down list.

If you want the column headers saved in the file, select a file format that includes headers (such as Excel With Headers). When you select a format with headers, the names of the database columns (not the column labels) are also saved in the file.

When you choose a format, PocketBuilder supplies the appropriate file extension.

3 Name the file and click Save.

PocketBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Modifying general DataWindow object properties

This section describes the general DataWindow object properties that you can modify.

Changing the DataWindow object style

The general style properties for a DataWindow object include:

- The unit of measure used in the DataWindow object
- A timer interval for events in the DataWindow object
- A background color for the DataWindow object

PocketBuilder assigns defaults when it generates the basic DataWindow object. You can change the defaults.

❖ **To change the default style properties:**

1. Position the pointer in the background of the DataWindow object, display the pop-up menu, and select Properties.

   The Properties view displays with the General page on top.

2. Click the unit of measure you want to use to specify distances when working with the DataWindow object:
   - PowerBuilder units (PBUs)
   - Pixels (smallest element on the display monitor)
   - Thousandths of an inch
   - Thousandths of a centimeter
3. Specify the number of milliseconds you want between internal timer events in the DataWindow object.

   This value determines how often PocketBuilder updates the time fields in the DataWindow object. (Enter 60,000 milliseconds to specify one minute.)
4. Select a background color from the Color drop-down list and click OK. The default color is the window background color.

Setting colors in a DataWindow object

You can set different colors for each element of a DataWindow object to enhance the display of information.

❖ **To set the background color in a DataWindow object:**

1. Position the mouse on an empty spot in the DataWindow object, display the pop-up menu, and select Properties.
2. On the General page in the Properties view for the DataWindow object, select a color from the Color drop-down list.

❖ **To set the color of a band in a DataWindow object:**

1 Position the mouse pointer on the bar that represents the band, display the pop-up menu, then select Properties.

2 On the General page in the band's Properties view, select a color from the Color drop-down list.

The choice you make here overrides the background color for the DataWindow object

❖ **To set colors in controls in a DataWindow object:**

- Position the mouse pointer on the control, display the pop-up menu, then select Properties

For controls that use text, you can set colors for background and text on the Font page in the Properties view. For drawing controls, you can set colors on the General page in the Properties view.

Specifying properties of a grid DataWindow object

In grid DataWindow objects you can specify:

- When grid lines are displayed
- How users can interact with the DataWindow object during execution

❖ **To specify basic grid DataWindow object properties:**

1 Position the mouse pointer on the background in a grid DataWindow object, display the pop-up menu, and select Properties.

2 Select the options you want in the Grid section on the General page in the Properties view as described in Table 18-4.

Table 18-4: Options for grid DataWindow objects

Option	Description
Display	You can select from the following values: • **On** Grid lines always display • **Off** Grid lines never display (users cannot resize columns during execution) • **Display only** This option is equivalent to selecting "On" • **Print only** This option is equivalent to selecting "Off"
Column Moving	When selected, columns can be moved during execution
Mouse Selection	When selected, data can be selected during execution
Row Resize	When selected, rows can be resized during execution

Defining print specifications for a DataWindow object

When you are satisfied with the look of the DataWindow object, you can define the print specifications for the DataWindow object.

❖ **To define print specifications for a DataWindow object:**

1 In the DataWindow painter, select Properties from the DataWindow object's pop-up menu to display the DataWindow object's Properties view.

2 In the Units box on the General page, select a unit of measure.

It is easier to specify the margins when the unit of measure is inches or centimeters.

3 Select the Print Specifications tab.

The Print Specifications page uses the units of measure you specified on the General page.

4 Specify print specifications for the current DataWindow object.

Table 18-5: Setting print specifications for DataWindow objects

Setting	Description
Document Name	Specify a name to be used in the print queue to identify the report.
Printer Name	Specify the name of a printer to which this report should be sent. If this box is empty, the report is sent to the default system printer. If the specified printer cannot be found, the report is sent to the default system printer if the Can Use Default Printer check box is selected. If the specified printer cannot be found and the Can Use Default Printer check box is not selected, an error is returned.
Margins	Specify top, bottom, left, and right margins. You can also change margins in the Preview view while you are actually looking at data. If you change margins in the Preview view, the changes are reflected here on the Print Specifications page.
Paper Orientation	Choose one of the following: • Default: Uses the default printer setup. • Portrait: Prints the contents of the DataWindow object across the width of the paper. • Landscape: Prints the contents of the DataWindow object across the length of the paper.
Paper Size	Choose a paper size or leave blank to use the default.
Paper Source	Choose a paper source or leave blank to use the default.
Prompt Before Printing	Select to display the standard Print Setup dialog box each time users make a print request.
Can Use Default Printer	Clear this check box if a printer has been specified in the Printer Name box and you do not want the report to be sent to the default system printer if the specified printer cannot be found. This box is checked by default if a printer name is specified.
Display Buttons - Print Preview	Select to display Button controls in Print Preview. The default is to hide them.
Display Buttons - Print	Select to display Button controls when you print the report. The default is to hide them.
Clip Text	Select to clip static text to the dimensions of a text field when the text field has no visible border setting. The text is always clipped if the text field has visible borders.

Setting	Description
Override Print Job	When you print a series of reports using the PrintOpen, PrintDataWindow, and PrintClose functions, all the reports in the print job use the layout, fonts, margins, and other print specifications defined for the computer. Select this check box to override the default print job settings and use the print settings defined for this report.
Collate Copies	Select to collate copies when printing. Collating increases print time because the print operation is repeated to produce collated sets.
Newspaper Columns Across and Width	Not applicable to PocketBuilder DataWindows. If you want a multiple-column report where the data fills one column on a page, then the second, and so on, as in a newspaper, select the number and width of the columns in the Newspaper Columns box.

Modifying text in a DataWindow object

You can change the text of a text control in a DataWindow object by selecting the control and typing new text in the first box of the StyleBar or in the Text text box on the General tab of the Properties view. To embed a newline character in the text, type ~n~r.

Changing text properties

When PocketBuilder initially generates the basic DataWindow object, it uses the following attributes and fonts:

- For the text and alignment of column headings and labels, PocketBuilder uses the extended column attributes made in the Database painter.
- For fonts, PocketBuilder uses the definitions made in the Database painter for the table. If you did not specify fonts for the table, PocketBuilder uses the defaults set in the Application painter.

You can override any of these defaults in a particular DataWindow object.

❖ **To change the text properties for a text control in a DataWindow object:**

1 Select the text control.

2 Do one of the following:

- Change the text properties in the StyleBar.
- Select the Font page in the control's Properties view and change the properties there.

Defining the tab order in a DataWindow object

When PocketBuilder generates the basic DataWindow object, it assigns columns a default tab order, the default sequence in which focus moves from column to column when a user presses the Tab key during execution. PocketBuilder assigns tab values in increments of 10 in left-to-right and top-to-bottom order.

Tab order is not used in the Design view
Tab order is used when a DataWindow object is executed, but it is not used in the DataWindow painter Design view. In the Design view, the Tab key moves to the controls in the DataWindow object in the order in which the controls were placed in the Design view.

If the DataWindow object contains columns from more than one table

If you are defining a DataWindow object with more than one table, PocketBuilder assigns each column a tab value of 0, meaning the user cannot tab to the column. This is because, by default, multitable DataWindow objects are not updatable—users cannot modify data in them. You can change the tab values to nonzero values to allow tabbing in these DataWindow objects.

For more about controlling updates in a DataWindow object, see Chapter 20, "Controlling Updates in DataWindow Objects."

Tab order changes have no effect in grid DataWindow objects
In a grid DataWindow object, the tab sequence is always left to right. Changing the tab value to any number other than 0 has no effect.

❖ **To change the tab order:**

1 Select Format>Tab Order from the menu bar or click the Tab Order button on PainterBar2.

The current tab order displays.

2 Use the mouse or the Tab key to move the pointer to the tab value you want to change.

3 Enter a new tab value in the range 0 to 9999.

0 removes the column from the tab order (the user cannot tab to the column). It does not matter exactly what value you use other than 0; all that matters is relative value. For example, if you want the user to tab to column B after column A but before column C, set the tab value for column B so it is between the value for column A and the value for column C.

4 Repeat the procedure until you have the tab order you want.

5 Select Format>Tab Order from the menu bar or click the Tab Order button again.

PocketBuilder saves the tab order.

Each time you select Tab Order, PocketBuilder reassigns tab values to include any columns that have been added to the DataWindow object and to allow space to insert new columns in the tab order.

Changing tab order during execution
To change tab order programmatically in a script, use the SetTabOrder function.

Naming controls in a DataWindow object

You use names to identify columns and other controls in filters, PowerScript functions, and DataWindow expression functions.

The DataWindow painter automatically generates names for all controls in a DataWindow object. To name columns, labels, and headings, the DataWindow painter uses database and extended attribute information. To name all other controls, it uses a system of prefixes. You can control the prefixes used for automatic name generation, and you can specify the name of any control explicitly.

❖ **To specify prefixes for naming controls systematically in a DataWindow object:**

1 Select Design>Options from the menu bar and then select the Prefixes tab.

2 Change prefixes as desired and click OK.

❖ **To specify a name of a control in a DataWindow object:**

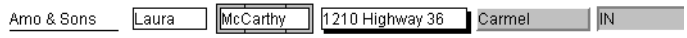
1 Select Properties from the control's pop-up menu and then select the General tab in the Properties view.

2 Type the name in the Name box.

Using borders in a DataWindow object

You can place borders around text and columns to enhance their appearance. PocketBuilder provides six types of borders: Underline, Box, ResizeBorder, ShadowBox, Raised, and Lowered.

Figure 18-4: Border styles for text and columns



❖ **To add a border to a control in a DataWindow object:**

1 Select one or more controls.

2 Select the border you want from the Border drop-down toolbar in the PainterBar.

PocketBuilder places the border around the selected controls.

You can also specify a border for one or more controls in the Properties view on the General page.

Specifying variable-height detail bands in a DataWindow object

Sometimes DataWindow objects contain columns whose data is of variable length. For example, a Memo column in a table might be a character column that can take up to several thousand characters. Reserving space for that much information for the column in the detail band would make the detail band's height very large, meaning you could see few rows at a time.

The detail band can resize based on the data in the Memo column. If the Memo column has only one line of text, the detail band should be one line. If the Memo column has 20 lines of text, the detail band should be 20 lines high.

To provide a detail band that resizes as needed, specify that the variable-length columns and the band have Autosize Height.

❖ **To create a resizable detail band in a DataWindow object:**

1 Select Properties from the pop-up menu of a column that should resize based on the amount of data.

2 Select the Autosize Height check box on the Position page.

3 Clear the Auto Horz Scroll check box on the Edit page.

PocketBuilder wraps text in the Preview view instead of displaying text on one scrollable line.

4 Repeat steps 1 to 3 for any other columns that should resize.

5 Select Properties from the detail band's pop-up menu.

6 Select the Autosize Height check box on the General page.

In the Preview view, the detail band resizes based on the contents of the columns you defined as having Autosize Height.

Clipping columns
You can have Autosize Height columns without an Autosize Height detail band. If such a column expands beyond the size of the detail band in the Preview view, it is clipped.

Modifying the data source of a DataWindow object

When modifying a DataWindow object, you might realize that you have not included all the columns you need, or you might need to define retrieval arguments. You can modify the data source from the DataWindow painter. How you do this depends on the data source.

Modifying SQL SELECT statements

If the data source is SQL (such as Quick Select, SQL Select, or Query), you can graphically modify the SQL SELECT statement.

❖ **To modify a SQL data source:**

1 Select Design>Data Source from the menu bar.

PocketBuilder returns you to the Select painter. (If you used Quick Select to define the data source, this might be the first time you have seen the Select painter.)

2 Modify the SELECT statement graphically using the same techniques as were used when creating it.

For more information, see "Using SQL Select" on page 430.

Modifying the statement syntactically
Select Design>Convert to Syntax from the menu bar to modify the SELECT statement syntactically.

3 Click the Return button to return to the painter.

Changing the table
If you change the table referenced in the SELECT statement, PocketBuilder maintains the columns in the Design view (now from a different table) only if they match the data types and order of the columns in the original table.

Modifying the retrieval arguments

You can add, modify, or delete retrieval arguments when modifying your data source.

❖ **To modify the retrieval arguments:**

1 In the Select painter, select Design>Retrieval Arguments from the menu bar.

The Specify Retrieval Arguments dialog box, displays listing the existing arguments.

2 Add, modify, or delete the arguments.

3 Click OK.

You return to the Select painter, or to the text window displaying the SELECT statement if you are modifying the SQL syntactically.

4 Reference any new arguments in the WHERE or HAVING clause of the SELECT statement.

For more information about retrieval arguments, see Chapter 17, "Defining DataWindow Objects."

Modifying the result set

If the data source is External or Stored Procedure, you can modify the result set description.

❖ **To modify a result set:**

1 If the Column Specification view is not open, select View>Column Specifications from the menu bar.

2 Review the specifications and make any necessary changes.

3 If you are modifying the result set for a DataWindow object whose data source is a stored procedure, do the following:

   - Right-click the Column Specification view, select Stored Procedure from the pop-up menu, then edit the Execute statement, select another stored procedure, or add retrieval arguments in the Modify Stored Procedure Data Source dialog box.

   For more information about editing the Execute statement, see "Using Stored Procedure" on page 448.

Storing data in a DataWindow object

Usually you retrieve data into a DataWindow object from the database, because the data is changeable and you want the latest information. However, sometimes the data you display in a DataWindow object almost never changes (as in a list of states or provinces), and sometimes you need a snapshot of the data at a certain point in time. In these situations, you can store the data in the DataWindow object itself.

Storing data in a DataWindow object is also a good way to share data and the DataWindow definition with other developers. They can simply open the DataWindow object on their computers to get the data and all its properties.

What happens at runtime

Data stored in a DataWindow object is stored within the actual object itself, so when a window opens showing such a DataWindow, the data is already there. There is no need to issue Retrieve to get the data.

PocketBuilder never retrieves data into a drop-down DataWindow that already contains data. For all other DataWindow objects, if you retrieve data into a DataWindow object stored with data, PocketBuilder handles it the same as a DataWindow object that is not stored with data: PocketBuilder gets the latest data by retrieving rows from the database.

Saving data for a drop-down DataWindow

The most common reason to store data in a DataWindow object is for use as a drop-down DataWindow where the data is not coming from a database. For example, you might want to display a list of postal codes for entering values in a State or Province column in a DataWindow object. You can store those codes in a DataWindow object and use the DropDownDataWindow edit style for the column.

For more information about using the DropDownDataWindow edit style, see Chapter 21, "Displaying and Validating Data."

❖ **To store data in a DataWindow object:**

1 If the Data view is not already displayed, select View>Data from the menu bar.

In the default layout for the DataWindow painter, the Data view displays in a stacked pane under the Properties view. All columns defined for the DataWindow object are listed at the top.

2 Do any of the following:

- Click the Insert Row button in the PainterBar to create an empty row and type a row of data. You can enter as many rows as you want.
- Click the Retrieve button in the PainterBar to retrieve all the rows of data from the database. You can delete rows you do not want to save or manually add new rows.
- Click the Delete button in the PainterBar to delete unwanted rows.

Data changes are local to the DataWindow object
Adding or deleting data here does not change the data in the database. It only determines what data will be stored with the DataWindow object when you save it. The Update Database button is disabled.

3 When you have finished, save the DataWindow object.

When you save the DataWindow object, the data is stored in the DataWindow object.

To see changes you make in the Data view reflected in the Preview view, select ShareData from the pop-up menu in the Data view. The Preview view shows data from the storage buffer associated with the Data view.

Saving the DataWindow object without data

After you save the DataWindow object with data to obtain a snapshot, you sometimes need to save it again without data. To do so, select Delete All Rows from the pop-up menu in the Data view before saving.

Prompting for retrieval criteria

You can define your DataWindow object so that it always prompts for retrieval criteria just before it retrieves data. PocketBuilder allows you to prompt for criteria when retrieving data for a DataWindow control, but not for a DataStore object.

❖ **To prompt for retrieval criteria in a DataWindow object:**

1 If the Column Specifications view is not already displayed, select View>Column Specifications from the menu bar.

In the default layout for the DataWindow painter, the Column Specifications view displays in a stacked pane under the Properties view. All columns defined for the DataWindow object are listed in the view.

2 Select the Prompt check box next to each column for which you want to specify retrieval criteria at runtime.

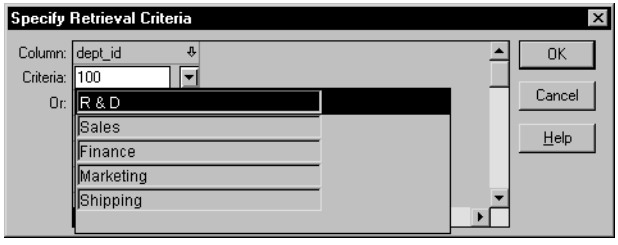
After you select the Prompt check box, PocketBuilder will display the Specify Retrieval dialog box just before a retrieval is to be done (it is the last thing that happens before the SQLPreview event).

Each column you selected in the Column Specification view displays in the grid. Users can specify criteria here exactly as in the grid in the Quick Select dialog box. Criteria specified are added to the WHERE clause for the SQL SELECT statement defined for the DataWindow object.

Testing in PocketBuilder
You can test whether the application containing the DataWindow object will prompt for retrieval criteria by retrieving data in the Preview view of the DataWindow object.

Using edit styles

If a column uses a code table or the RadioButton, CheckBox, or DropDownListBox edit style, an arrow displays in the column header and users can select a value from a drop-down list when specifying criteria.



If you do not want the drop-down list used for a column for specifying retrieval criteria, select the Override Edit check box on the General page of the column's Properties view.

Forcing the entry of criteria

If you have specified that a column should prompt for criteria, you can force the entry of criteria for the column by selecting the Equality Required check box on the General page of the column's Properties view. PocketBuilder underlines the column header in the grid during prompting. Selection criteria for the specified column must be entered, and the = operator must be used.

For more information

The section “Using Quick Select” on page 421 describes in detail how you can specify selection criteria in the grid.

The chapter on dynamic DataWindow objects in the *Resource Guide* describes how to write scripts to dynamically allow users to specify retrieval criteria during execution.

Using MOP views for DataWindows

Assigning multiple views for a DataWindow object

The DataWindow painter includes a Multiple Orientation Painter view manager that lets you assign runtime layouts which change automatically when an application user changes the orientation of a handheld device or emulator. You can use the View>Runtime MOP Views>MOPViewManager menu item in the DataWindow painter to create multiple views for a DataWindow object with the same orientations available to PocketBuilder windows.

For more information on the available views, see “Multiple Orientation Painter for windows” on page 211.

When you open a window containing a DataWindow at runtime, PocketBuilder automatically selects the view of theDataWindow that correlates with the current orientation of the device or emulator. If you do not create a specific view for the selected orientation, the last view saved in the painter is displayed at runtime. In this case, the objects in the DataWindow do not necessarily display with optimal sizes and positions for the user-selected orientation.

Property specificity

For a DataWindow with multiple views, the position (x and y) and dimension (width and height) properties of the DataWindow and its column, text, and control objects are view-specific. However, the height properties of the DataWindow bands are not affected. All other properties are specific to the DataWindow—that is, they apply to all of its defined views.

The source code in the PKL file includes only the last position and dimension properties saved for the column, text, and control objects in the DataWindow. If you export the code for a DataWindow object with MOP views, the MOP views are not included in the exported code. The exported code contains only the most recent position and dimension properties from the original DataWindow. However, if you copy or move the DataWindow object, the copied or moved object contains all the original MOP views.

CHAPTER 19

Working with Controls in DataWindow Objects

About this chapter

One of the ways you can enhance a DataWindow object is to add controls such as columns, drawing objects, buttons, and computed fields. You can also change the layout of the DataWindow object by reorganizing, positioning, and rotating controls. This chapter shows you how.

Contents

Adding controls to a DataWindow object

This section describes adding controls to enhance your DataWindow object.

Adding columns to a DataWindow object

You can add columns that are included in the data source to a DataWindow object. When you first create a DataWindow object, each of the columns in the data source is automatically placed in the DataWindow object. Typically, you would add a column to restore one that you had deleted from the DataWindow object, or to display the column more than once in the DataWindow object.

Adding columns not previously retrieved to the data source
To specify that you want to retrieve a column not previously retrieved (that is, add a column to the data source), you must modify the data source.

See "Modifying the data source of a DataWindow object" on page 484.

❖ **To add a column from the data source to a DataWindow object:**

1 Select Insert>Control>Column from the menu bar.

2 Click where you want to place the column.

The Select Column dialog box displays, listing all columns included in the data source of the DataWindow object.

3 Select the column and click OK.

Adding text to a DataWindow object

When PocketBuilder generates a basic DataWindow object from a presentation style and data source, it places columns and their headings in the workspace. You can add text anywhere you want in order to make the DataWindow object easier to understand.

❖ **To add text to a DataWindow object:**

1 Select Insert>Control>Text from the menu bar.

2 Click where you want the text.

PocketBuilder places the text control in the Design view and displays the word `text`.

3 Type the text you want.

4 (Optional) Change the font, size, style, and alignment for the text using the StyleBar.

Displaying an ampersand character
If you want to display an ampersand character, type a double ampersand in the Text field. A single ampersand causes the next character to display with an underscore because it is used to indicate accelerator keys.

About the default font and style

When you place text in a DataWindow object, PocketBuilder uses the font and style (such as boldface) defined for the application's text in the Application painter. You can override the text properties for any text in a DataWindow object.

For more about changing the application's default text font and style, see Chapter 3, “Working with PowerScript Targets.”

Adding drawing controls to a DataWindow object

You can add the following drawing controls to a DataWindow object to enhance its appearance:

- Rectangle
- RoundRectangle
- Line
- Oval

❖ **To place a drawing control in a DataWindow object:**

1 Select the drawing control from the Insert>Control menu.

2 Click where you want the control to display.

3 Resize or move the drawing control as needed.

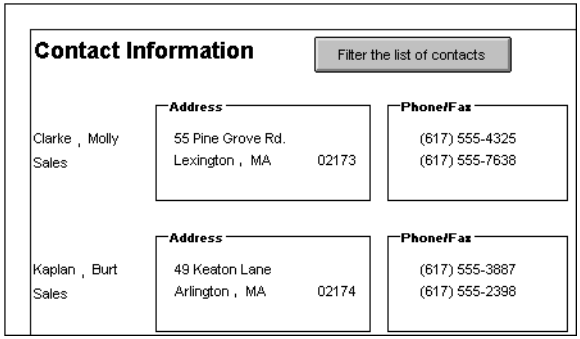
4 Use the drawing control's Properties view to change its properties as needed.

For example, you might want to specify a fill color for a rectangle or thickness for a line.

Adding a group box to a DataWindow object

To visually enhance the layout of a DataWindow object, you can add a group box. The group box is a static frame used to group and label a set of controls in a DataWindow object. The following example shows two group boxes in a report (a nonupdatable DataWindow object). The Address group box groups address information, and the Phone/Fax group box groups telephone numbers.

Figure 19-1: DataWindow with group boxes



❖ **To add a group box to a DataWindow object:**

1. Select Insert>Control>Group Box from the menu bar and click in the Design view.
2. With the group box selected, type the text to display in the frame.
3. Move and resize the group box as appropriate.

Adding pictures to a DataWindow object

You can place pictures, such as your company logo, in a DataWindow object to enhance its appearance. If you place a picture in the header, summary, or footer band of the DataWindow object, the picture displays each time the contents of that band displays. If you place the picture in the detail band of the DataWindow object, it displays in each row.

❖ **To place a picture in a DataWindow object:**

1. Select Insert>Control>Picture from the menu bar.
2. Click where you want the picture to display.

   The Select Picture dialog box displays.
3. Use the Browse button to find the file, or enter a file name in the File Name box. Then click Open.

   The picture can be a bitmap (BMP), Graphics Interchange Format (GIF), or Joint Photographic Experts Group (JPEG) file.

4 Display the pop-up menu and select Original Size to display the bitmap in its original size.

   You can use the mouse to change the size of the bitmap in the DataWindow painter.

5 Click the Invert Image check box on the General page of the Properties view to display the picture with its colors inverted.

Tips for using pictures

To display a different picture for each row of data, retrieve a column containing picture file names from the database.

For more information, see "Specifying additional properties for character columns" on page 373.

To compute a picture name during execution, use the Bitmap function in the expression defining a computed field. If you change the bitmap in the Picture control in a DataWindow object, you need to reset the original size property. The property automatically reverts to the default setting when you change the bitmap.

To use a picture to indicate that a row has focus during execution, use the SetRowFocusIndicator function.

Adding computed fields to a DataWindow object

You can use computed fields in any band of the DataWindow object. Typical uses with examples include:

- Calculations based on column data that change for each retrieved row

  If you retrieve yearly salary, you can define a computed field in the detail band that displays monthly salary: `Salary / 12`.

- Summary statistics of the data

  In a grouped DataWindow object, you can use a computed field to calculate the totals of a column, such as salary, for each group: `sum (salary for group 1)`.

- Concatenated fields

  If you retrieve first name and last name, you can define a computed field that concatenates the values so they appear with only one space between them: `Fname + " " + Lname`.

- System information

  You can place the current date and time in a DataWindow object's header using the built-in functions `Today()` and `Now()` in computed fields.

Computed columns versus computed fields

When creating a DataWindow object, you can define computed columns and computed fields as follows:

- In the Select painter, you can define computed columns when you are defining the SELECT statement that will be used to retrieve data into the DataWindow object.
- In the DataWindow painter, you can define computed fields after you have defined the SELECT statement (or other data source).

The difference between the two ways

When you define the computed column in the Select painter, the value is calculated by the DBMS when the data is retrieved. The computed column's value does not change until data has been updated and retrieved again.

When you define the computed field in the DataWindow painter, the value of the column is calculated in the DataWindow object after the data has been retrieved. The value changes dynamically as the data in the DataWindow object changes.

Example

Consider a DataWindow object with four columns: Part number, Quantity, Price, and Cost. Cost is computed as Quantity * Price.

Part #	Quantity	Price	Cost
101	100	1.25	125.00

If Cost is defined as a computed column in the Select painter, the SELECT statement is as follows:

```
SELECT part.part_num,
part.part_qty,
part.part_price,
part.part_qty * part.part_price
FROM part;
```

If the user changes the price of a part in the DataWindow object in this scenario, the cost does not change in the DataWindow object until the database is updated and the data is retrieved again. The user sees a display with the changed price but the unchanged, incorrect cost.

Part #	Quantity	Price	Cost
101	100	2.50	*125.00*

If Cost is defined as a computed field in the DataWindow object, the SELECT statement is as follows, with no computed column:

```
SELECT part.part_num,
part.part_qty,
part.part_price
FROM part;
```

The computed field is defined in the DataWindow object as `Quantity * Price`.

In this scenario, if the user changes the price of a part in the DataWindow object, the cost changes immediately.

Part #	Quantity	Price	Cost
101	100	2.50	*250.00*

Recommendation

If you want your DBMS to do the calculations on the server before bringing data down and you do not need the computed values to be updated dynamically, define the computed column as part of the SELECT statement.

If you need to have computed values change dynamically, define computed fields in the DataWindow painter Design view, as described next.

Defining a computed field in the DataWindow painter

❖ **To define a computed field in the DataWindow painter:**

1 Select Insert>Control>Computed Field from the menu bar.

2 Click in the Design view at the location where you want the computed field.

If the calculation is to be based on column data that changes for each row, make sure you place the computed field in the detail band.

The Modify Expression dialog box displays, listing:

- DataWindow expression functions you can use in the computed field
- The columns in the DataWindow object
- Operators and parentheses

3 Enter the expression that defines the computed field as described in "Entering the expression" next.

4 (Optional) Click Verify to test the expression.

PocketBuilder analyzes the expression.

5 Click OK.

Entering the expression

You can enter any valid DataWindow expression when defining a computed field. You can paste operators, columns, and DataWindow expression functions into the expression from information in the Modify Expression dialog box. Use the + operator to concatenate strings.

You can use any built-in or user-defined global function in an expression. You cannot use object-level functions.

DataWindow expressions
You are entering a DataWindow expression, not a SQL expression processed by the DBMS, so the expression follows the rules for DataWindow expressions. For complete information about DataWindow expressions, see the *DataWindow Reference* in the online Help.

Referring to next and previous rows

You can refer to other rows in a computed field. Use this syntax:

ColumnName[*x*]

where *x* is an integer. 0 refers to the current row (or first row in the detail band), 1 refers to the next row, –1 refers to the previous row, and so on.

Examples

Table 19-1 shows some examples of computed fields.

Table 19-1: Computed field examples

To display	Enter this expression	In this band
Current date at top of each page	`Today()`	Header
Current time at top of each page	`Now()`	Header
Current page at bottom of each page	`Page()`	Footer
Total page count at bottom of each page	`PageCount()`	Footer

To display	Enter this expression	In this band
Concatenation of Fname and Lname columns for each row	`Fname + " " + Lname`	Detail
Monthly salary if Salary column contains annual salary	`Salary / 12`	Detail
Four asterisks if the value of the Salary column is greater than $50,000	`IF(Salary> 50000, "****", "")`	Detail
Average salary of all retrieved rows	`Avg(Salary)`	Summary
Count of retrieved rows, assuming each row contains a value for EmpID	`Count(EmpID)`	Summary

For complete information about the functions you can use in computed fields in the DataWindow painter, see the *DataWindow Reference* in the online Help.

Menu options and PainterBar buttons for common functions

PocketBuilder provides a quick way to create computed fields that summarize values in the detail band, display the current date, or show the current page number.

❖ **To summarize values:**

1 Select one or more columns in the DataWindow object's detail band.

2 Select Insert>Control from the menu bar.

3 Select one of the options at the bottom of the cascading menu: Average, Count, or Sum.

The same options are available at the bottom of the Controls drop-down toolbar on the PainterBar.

PocketBuilder places a computed field in the summary band or in the group trailer band if the DataWindow object is grouped. The band is resized automatically to hold the computed field. If the computed field you are adding matches an existing computed field, the duplicate field is not generated.

❖ **To insert a computed field for the current date or page number:**

1 Select Insert>Control from the menu bar.

2 Select Today() or Page n of n from the options at the bottom of the cascading menu.

The same options are available at the bottom of the Controls drop-down toolbar on the PainterBar.

3 Click anywhere in the DataWindow object.

If you selected Today, PocketBuilder inserts a computed field containing this expression: `Today()`. For Page n of n, the computed field contains this expression: `'Page ' + page() + ' of ' + pageCount()`.

Adding custom buttons that place computed fields

You can add buttons to the PainterBar in the DataWindow painter that place computed fields using any of the aggregate functions, such as Max, Min, and Median.

❖ **To customize the PainterBar with custom buttons for placing computed fields:**

1. Place the mouse pointer over the PainterBar and select Customize from the pop-up menu.

   The Customize dialog box displays.

2. Click Custom in the Select palette group to display the set of custom buttons.

3. Drag a custom button into the Current toolbar group and release it.

   The Toolbar Item Command dialog box displays.

4. Click the Function button.

   The Function For Toolbar dialog box displays.

5. Select a function and click OK.

   You return to the Toolbar Item Command dialog box.

6. Specify text and microhelp that displays for the button and click OK.

   PocketBuilder places the new button in the PainterBar. You can click it to add a computed field to your DataWindow object the same way you use the built-in Sum button.

Adding buttons to a DataWindow object

Buttons make it easy to provide command button actions in a DataWindow object. No coding is required. The use of Button controls in the DataWindow object instead of CommandButton controls in a window ensures that actions appropriate to the DataWindow object are included in the object itself.

The Button control is a command or picture button that can be placed in a DataWindow object. When clicked at runtime, the button activates either a built-in or user-supplied action.

For example, you can place a button in a report and specify that clicking it opens the Filter dialog box, where users can specify a filter to be applied to the currently retrieved data.

❖ **To add a button to a DataWindow object:**

1 Select Insert>Control>Button from the menu bar.

2 Click where you want the button to display.

You might find it useful to put a Delete button or an Insert button in the detail band. Clicking a Delete button in the detail band deletes the row next to the button clicked. Clicking an Insert button in the detail band inserts a row following the current row.

Be careful when putting buttons in the detail band
Buttons in the detail band repeat for every row of data, which is not always desirable. Buttons in the detail band are not visible during retrieval, so a Cancel button in the detail band would be unavailable when needed.

3 With the button still selected, type the text to display on the button.

4 Display the General page of the Properties view for the button.

5 Select the action you want to assign to the button from the Action drop-down list.

For information about actions, see "Actions assignable to buttons in DataWindow objects" on page 502.

6 If you want to add a picture to the button, select the Action Default Picture check box or enter the name of the Picture file to display on the button.

7 If you want to suppress event processing when the button is clicked at runtime, select the Suppress Event check box.

When this option has been selected for the button and the button is clicked at runtime, only the action assigned to the button and the Clicked event are executed. The ButtonClicking and the ButtonClicked events are not triggered.

8 Click OK.

What happens if Suppress Event is off

If Suppress Event is off and the button is clicked, the Clicked and ButtonClicking events are fired. Code in the ButtonClicking event (if any) is executed. Note that the Clicked event is executed before the ButtonClicking event.

- If the return code from the ButtonClicking event is 0, the action assigned to the button is executed and then the ButtonClicked event is executed
- If the return code from the ButtonClicking event is 1, neither the action assigned to the button nor the ButtonClicked event is executed

Do not use a message box in the Clicked event
If you call the MessageBox function in the Clicked event, the action assigned to the button is executed, but the ButtonClicking and ButtonClicked events are not executed.

Controlling the display of buttons in print preview and on printed output

You can choose whether to display buttons in print preview or in printed output. You control this in the Properties view for the DataWindow object, not the Properties view for the button.

❖ **To control the display of buttons in a DataWindow object in print preview and on printed output:**

1 Display the DataWindow object's Properties view with the Print Specification page on top.

2 Select the Display Buttons – Print check box.

The buttons are included in the printed output when the DataWindow object is printed.

3 Select the Display Buttons – Print Preview check box.

The buttons display on the screen when you view the DataWindow object in print preview.

Actions assignable to buttons in DataWindow objects

Table 19-2 shows the actions you can assign to a button in a DataWindow object. Each action is associated with a numeric value (the Action DataWindow object property) and a return code (the actionreturncode event argument).

The following code in the ButtonClicked event displays the value returned by the action:

```
MessageBox("Action return code", actionreturncode)
```

Table 19-2: Button actions for DataWindow objects

Action	Effect	Value	Action return code
User Defined (default)	Allows the developer to program the ButtonClicked event with no intervening action occurring.	0	The return code from the user's coded event script.
Retrieve (Yield)	Retrieves rows from the database. Before retrieval occurs, the option to yield is turned on; this will allow the Cancel action to take effect during a long retrieve.	1	Number of rows retrieved. -1 if retrieve fails.
Retrieve	Retrieves rows from the database. The option to yield is not automatically turned on.	2	Number of rows retrieved. -1 if retrieve fails.
Cancel	Cancels a retrieval that has been started with the option to yield.	3	0
Page Next	Scrolls to the next page.	4	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Page Prior	Scrolls to the prior page.	5	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Page First	Scrolls to the first page.	6	1 if successful. -1 if an error occurs.
Page Last	Scrolls to the last page.	7	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Sort	Displays Sort dialog box and sorts as specified.	8	1 if successful. -1 if an error occurs.
Filter	Displays Filter dialog box and filters as specified.	9	Number of rows filtered. Number < 0 if an error occurs.
Delete Row	If button is in the detail band, deletes the row associated with button; otherwise, deletes the current row.	10	1 if successful. -1 if an error occurs.
Append Row	Inserts row at the end.	11	Row number of newly inserted row.

Action	Effect	Value	Action return code
Insert Row	If button is in the detail band, inserts a row using row number associated with the button; otherwise, inserts row using the current row.	12	Row number of newly inserted row.
Update	Saves changes to the database. If the update is successful, a Commit will be issued; if the update fails, a Rollback will be issued.	13	1 if successful. -1 if an error occurs.
Save Rows As	Displays Save As dialog box and saves rows in the format specified.	14	Number of rows filtered. Number < 0 if an error occurs.
Print	Prints one copy of the DataWindow object.	15	0
Preview	Toggles between preview and print preview.	16	0
Preview With Rulers	Toggles between rulers on and off.	17	0
Query Mode	Toggles between query mode on and off.	18	0
Query Sort	Allows user to specify sorting criteria (forces query mode on).	19	0
Query Clear	Removes the WHERE clause from a query (if one was defined).	20	0

Adding graphs to a DataWindow object

Graphs are one of the best ways to present information. For example, if your application displays sales information over the course of a year, you can easily build a graph in a DataWindow object to display the information visually.

PocketBuilder offers many types of graphs and provides you with the ability to control the appearance of a graph to best meet your application's needs.

For information on using graphs, see Chapter 24, “Working with Graphs.”

Reorganizing controls in a DataWindow object

This section describes the activities that help you change the layout and appearance of the controls in a DataWindow object.

Displaying boundaries for controls in a DataWindow object

When reorganizing controls in the Design view, it is sometimes helpful to see how large all the controls are. That way you can easily check for overlapping controls and make sure that the spacing around controls is what you want it to be.

❖ **To display control boundaries in a DataWindow object:**

1 Select Design>Options from the menu bar.

The DataWindow Options dialog box displays.

2 Select the Show Edges check box.

PocketBuilder displays the boundaries of each control in the DataWindow object.

Boundaries display only in the Design view
The boundaries displayed for controls are for use only in the Design view. They do not display in a running DataWindow object or in a printed report.

Using the grid and the ruler in a DataWindow object

The DataWindow painter provides a grid and a ruler to help you align controls. You can select from the options in Table 19-3.

Table 19-3: Grid and ruler options for the DataWindow painter

Option	Meaning
Snap to Grid	Make controls snap to a grid position when you place them or move them.
Show Grid	Show or hide the grid when the workspace displays.
X	Specify the size (width) of the grid cells.
Y	Specify the size (height) of the grid cells.
Show Ruler	Show a ruler. The ruler uses the units of measurement specified in the Style dialog box. See "Changing the DataWindow object style" on page 475.

❖ **To use the grid and the ruler:**

1 Select Design>Options from the menu bar.

The DataWindow Options dialog box displays. The Alignment Grid box contains the alignment grid options.

2 Select the grid and ruler options that you want.

Your choices for the grid and the ruler are saved and used the next time you start PocketBuilder.

Deleting controls in a DataWindow object

❖ **To delete controls in a DataWindow object:**

1 Select the controls you want to delete.

2 Select Edit>Delete from the menu bar or press the Delete key.

Moving controls in a DataWindow object

In all presentation styles except Grid

In all presentation styles except Grid, you can move all the controls (such as headings, labels, columns, graphs, and drawing controls) anywhere you want.

❖ **To move controls in a DataWindow object:**

1 Select the controls you want to move.

2 Do one of the following:

- Drag the controls with the mouse
- Press an arrow key to move the controls in one direction

In grid DataWindow objects

You can reorder columns in a grid DataWindow object during execution.

See "Working in a grid DataWindow object" on page 473.

Copying controls in a DataWindow object

You can copy controls within a DataWindow object and to other DataWindow objects. All properties of the controls are copied.

❖ **To copy a control in a DataWindow object:**

1 Select the control.

2 Select Edit>Copy from the menu bar.

The control is copied to a private PocketBuilder clipboard.

3 Copy (paste) the control to the same DataWindow object or to another one:

- To copy the control within the same DataWindow object, select Edit>Paste from the menu bar
- To copy the control to another DataWindow object, open the desired DataWindow object and paste the control

PocketBuilder pastes the control at the same location as in the source DataWindow object. If you are pasting into the same DataWindow object, you should move the pasted control so that it does not cover the original control. PocketBuilder displays a message box if the control you are pasting is not valid for the destination DataWindow object.

Resizing controls in a DataWindow object

You can resize a control using the mouse or the keyboard. You can also resize multiple controls to the same size using the Layout drop-down toolbar on PainterBar2.

Using the mouse

To resize a control using the mouse, select it, then grab an edge and drag it with the mouse.

Using the keyboard

To resize a control using the keyboard, select it and use the keyboard keys as described in Table 19-4.

Table 19-4: Resizing the controls with the arrow keys

To make the control	Press
Wider	Shift+Right Arrow
Narrower	Shift+Left Arrow
Taller	Shift+Down Arrow
Shorter	Shift+Up Arrow

In grid DataWindow objects

You can resize columns in grid DataWindow objects.

❖ **To resize a column in a grid DataWindow object:**

1 Position the mouse pointer at a column boundary.

The pointer changes to a two-headed arrow.

2 Press and hold the left mouse button and drag the mouse to move the boundary.

3 Release the mouse button when the column is the correct width.

Aligning controls in a DataWindow object

You might need to align several controls or make them all the same size. You can use the grid to align the controls, or you can have PocketBuilder align them for you.

❖ **To align controls in a DataWindow object:**

1 Select the control with which you want to align the others.

PocketBuilder displays handles around the selected control.

2 Extend the selection by pressing and holding the Ctrl key and clicking the controls you want to align with the first one.

   All the controls have handles on them.

3 Select Format>Align from the menu bar.

4 From the cascading menu, select the dimension along which you want to align the controls.

   For example, to align the controls along the left side, select the first choice on the cascading menu. You can also use the Layout drop-down toolbar on PainterBar2.

   PocketBuilder moves all the selected controls to align with the first one.

Equalizing the space between controls in a DataWindow object

If you have a series of controls and the spacing is fine between two of them but is wrong for the rest, you can easily equalize the spacing around all the controls.

❖ **To equalize the space between controls in a DataWindow object:**

1 Select the two controls whose spacing is correct.

   To do so, click one control, then press Ctrl and click the second control.

2 Select the other controls whose spacing should match that of the first two controls. To do so, press Ctrl and click each control.

3 Select Format>Space from the menu bar.

4 From the cascading menu, select the dimension whose spacing you want to equalize.

   You can also use the Layout drop-down toolbar on PainterBar2.

Equalizing the size of controls in a DataWindow object

If you have several controls in a DataWindow object and want their sizes to be the same, you can change their sizes on the Position page of the Properties view or from the Format menu.

❖ **To equalize the size of controls in a DataWindow object:**

1 Select the control whose size is correct.

2 Select the other controls whose size should match that of the first control by pressing Ctrl and clicking.

3 Select Format>Size from the menu bar.

4 From the cascading menu, select the dimension whose size you want to equalize.

You can also use the Layout drop-down toolbar on PainterBar2.

Sliding controls to remove blank space in a DataWindow object

You can specify that you want to eliminate blank lines or spaces in a DataWindow object by sliding columns and other controls to the left or up if there is blank space. You can use this feature to remove extra spaces between fields such as first and last name.

Table 19-5: Slide options for controls in a DataWindow object

Option	Description
Slide Left	When selected, slides the column or control to the left if there is nothing to the left. Be sure the control does not overlap the control to the left. Sliding left will not work if the controls overlap.
Slide Up	You can select from the following values: • **All Above** Slide the column or control up if there is nothing in the row above. The row above must be completely empty for the column or control to slide up. • **Directly Above** Slide the column or control up if there is nothing *directly above it* in the row above.

If you are sliding columns up
Even blank columns have height; if there are blank columns above the column you want to slide up, you need to set the Autosize Height property for all of them.

❖ **To use sliding columns or controls in a DataWindow object:**

1 Select Properties from the control's pop-up menu and then select the Position tab in the Properties view.

2 Select the Slide options you want.

Slide options are also available on PainterBar2.

Positioning controls in a DataWindow object

Table 19-6 shows the properties for each control in a DataWindow object that determine how it is positioned within the DataWindow object. All except the HideSnaked property are on the Position page of the object's Properties view.

Sliding properties also affect control positioning. Sliding properties are described on "Sliding controls to remove blank space in a DataWindow object" on page 510.

Table 19-6: Position properties for controls in a DataWindow object

Property	Description
X	Position (in units you selected for the DataWindow object) from the left edge of the DataWindow band where the control is placed.
Y	Position (in units you selected for the DataWindow object) from the top edge of the DataWindow band where the control is placed.
Width	Width of the control in units you selected for the DataWindow object.
Height	Height of control in units you selected for the DataWindow object.
Layer	You can select from the following values: • **Background** Control is behind other controls. It is not restricted to one band. This is useful for adding a watermark (such as the word CONFIDENTIAL) to the background of a report. • **Band** Control is placed within one band. It cannot extend beyond the band's border. • **Foreground** Control is in front of other controls. It is not restricted to one band.
Moveable	When selected, control can be moved during execution and in preview. This is useful for designing layout.
Resizable	When selected, control can be resized during execution and in preview. This is useful for designing layout.
HideSnaked	When selected, control appears only in the first column on the page; in subsequent columns the control does not appear. This is only for newspaper columns, where the entire DataWindow object snakes from column to column (set on the General page of the Properties view).

Default positioning

PocketBuilder uses the defaults shown in Table 19-7 when you place a new control in a DataWindow object.

Table 19-7: Default position for controls in a DataWindow object

Control	Default position
Graph	Foreground, movable, resizable
All other controls	Band, not movable, not resizable

Rotating controls in a DataWindow object

Controls that display text, such as text controls, columns, and computed fields, can be rotated from the original baseline of the text. The Escapement option on the Font properties page for the control lets you specify the amount of rotation, also known as escapement.

Several other properties of a rotated control affect its final placement when the DataWindow object runs. The location of the control in Design view, the amount of rotation specified for it, and the location of the text within the control (for example centered text or left-aligned text) all contribute to what you see in the DataWindow object Preview view.

The following procedure includes design practices that help ensure that you get the final results you want. As you become more experienced you can drop or alter some of the steps. The procedure recommends setting a visible border on the control so you can see where the control is located in the Preview view and making the control movable in the Preview view, which is often helpful.

❖ **To rotate a control in a DataWindow object:**

1 Select the control in the Design view.

2 On the General properties page, change the control's border to Box. On the Position properties page make the control movable.

3 In Design view, enlarge the area in which the control is placed.

For example, in a grid DataWindow object, make the band deeper and move the control down into the center of the band.

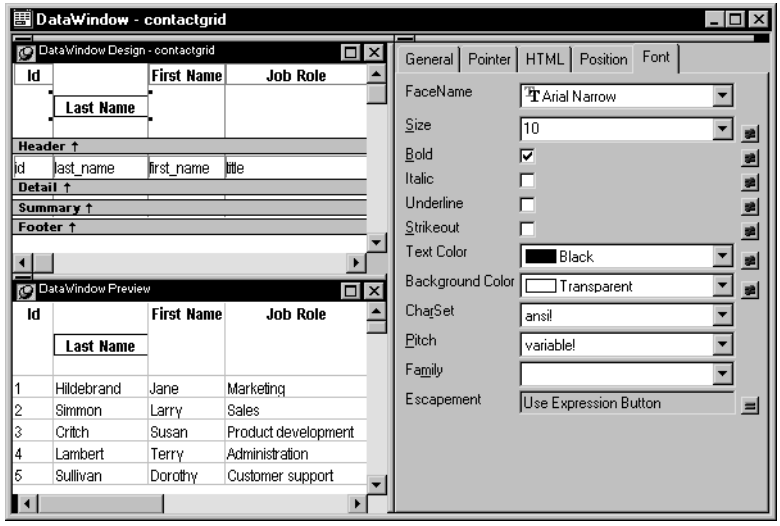
4 From the Font properties page, display the Modify Expression dialog box.

You click the button next to the Escapement property to display the dialog box.

5 Specify the amount of rotation you want as an integer in tenths of a degree. (For example, 450 means 45 degrees of rotation; 0 means horizontal or no rotation.)

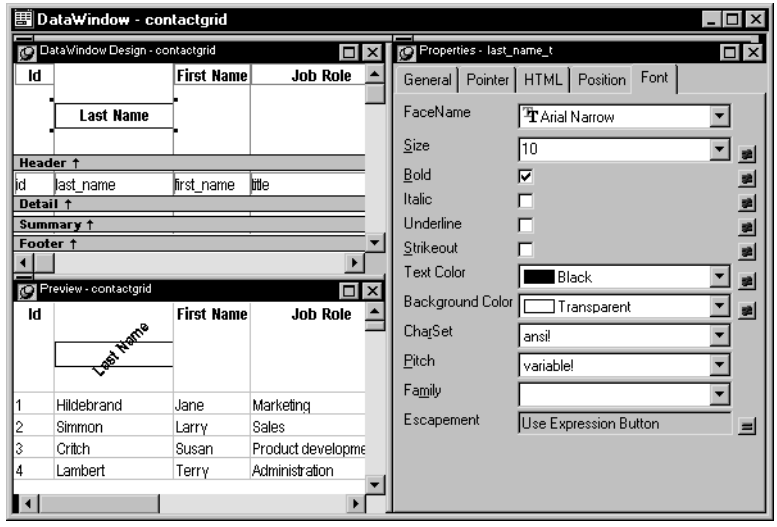
The origin of rotation is the center of the top border of the box containing the text. It is often helpful to use left-aligned text (General properties page>Alignment>Left) because it makes it easier to position the control correctly. The example in Figure 19-2 shows text centered within the control.

Figure 19-2: Positioning a text control to make space for text rotation



6 To display the current rotation in Preview, close the Preview view and reopen it (View>Preview on the menu bar).

Figure 19-3: Displaying rotated text in the Preview view



7 Drag and drop the control in the Preview view or Design view until it is where you want it.

8 In Design view, select the control that is being rotated, remove the temporary border, and deselect the Moveable check box.

If you are using a conditional expression for rotation

If you are specifying different rotations depending on particular conditions, you might need to add conditions to the control's x and y properties to conditionally move the control to match the various amounts of rotation. An alternative to moving the control around is to have multiple controls positioned exactly as you want them, taking into account the different amounts of rotation. Then you can add a condition to the visible property of each control to ensure that the correctly rotated control displays.

CHAPTER 20

Controlling Updates in DataWindow Objects

About this chapter

When PocketBuilder generates a basic DataWindow object, it defines whether the data is updatable. This chapter describes the default update settings and how you can modify them.

Contents

About controlling updates

When PocketBuilder generates a basic DataWindow object, it defines whether the data is updatable using the following default settings:

- If the DataWindow object contains columns from a single table and includes that table's key columns, PocketBuilder defines all columns as updatable and specifies a nonzero tab order for each column, allowing users to tab to the columns.
- If the DataWindow object contains columns from two or more tables or from a view, PocketBuilder defines all columns as not updatable and sets all tab orders to zero, preventing users from tabbing to the columns.

You can accept the default settings or modify the update characteristics for a DataWindow object.

If using a Stored Procedure or External data source
If the data source is Stored Procedure or External, you can use the GetNextModified method to write your own update script. For more information, see the *DataWindow Reference* in the online Help.

Changing update settings

You can change DataWindow update settings to:

- Allow updates in a DataWindow object associated with multiple tables or a view, and define one of the tables as being updatable
- Prevent updates in a DataWindow object associated with one table
- Prevent updates to specific columns in a DataWindow object that is associated with an updatable table
- Specify which columns uniquely identify a row to be updated
- Specify which columns will be included in the WHERE clause of the UPDATE or DELETE statement PocketBuilder generates to update the database
- Specify whether PocketBuilder generates an UPDATE statement, or a DELETE then an INSERT statement, to update the database when users modify the values in a key column

Updatability of views
Some views are logically updatable; some are not. For the rules your DBMS follows for updating views, see your DBMS documentation.

Changing tab values
PocketBuilder does not change the tab values associated with columns after you change the update characteristics of the DataWindow object. If you have allowed updates to a table in a multitable DataWindow object, you should change the tab values for the updatable columns so users can tab to them.

For more information, see “Defining the tab order in a DataWindow object” on page 481.

❖ **To specify update characteristics for a DataWindow object:**

1 Select Rows>Update Properties from the menu bar.

The Specify Update Properties dialog box displays.

2 To prevent updates to the data, make sure the Allow Updates box is not selected.
To allow updates, select the Allow Updates box and make changes as required to the update settings by:

Specifying the table to update
Specifying the unique key columns
Specifying an identity column
Specifying updatable columns
Specifying the WHERE clause for update/delete
Specifying update when key is modified

3 Click OK.

Specifying the table to update

Each DataWindow object can update one table, which you select from the Table to Update box in the Specify Update Properties dialog box.

Figure 20-1: Selecting a table to update

Specifying the unique key columns

The Unique Key Columns box in the Specify Update Properties dialog box specifies which columns PocketBuilder uses to identify a row being updated. PocketBuilder uses the column or columns you specify here as the key columns when generating the WHERE clause to update the database.

The key columns you select must uniquely identify a row in the table. They can be the table's primary key, though they do not have to be.

Figure 20-2: Selecting columns that identify the rows to be updated

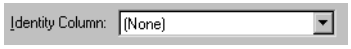
Using the primary key
Clicking the Primary Key button cancels any changes in the Unique Key Columns box and highlights the primary key for the updatable table.

Specifying an identity column

SQL Anywhere allows you to specify that the value for a column in a new row is to be automatically assigned. This kind of column is called an identity column.

For example, SQL Anywhere allow you to define autoincrement columns so that the column for a new row is automatically assigned a value one greater than that of the previous highest value. You could use this feature to specify that the order number should be automatically incremented when someone adds a new order:

Figure 20-3: Selecting identity columns for autoincrementation



By specifying an identity column in the Specify Update Properties dialog box, you tell PocketBuilder to bring back the value of a new row's identity column after an insert in the DataWindow object so that users can see it.

Specifying updatable columns

You can make all or some of the columns in a table updatable.

Updatable columns are displayed with highlighting. Click a nonupdatable column to make it updatable. Click an updatable column to make it nonupdatable.

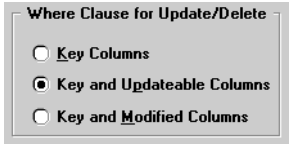
Changing tab values

If you have changed the updatability of a column, you should change its tab value. If you have allowed a column to be updated, you should change its tab value to a nonzero number so that users can tab to it.

Specifying the WHERE clause for update/delete

Sometimes multiple users access the same tables at the same time. In this type of situation, you need to decide when to allow your application to update the database. If you allow your application to always update the database, it could overwrite changes made by other users.

Figure 20-4: Controlling when updates succeed



You can control when updates succeed by specifying which columns PocketBuilder includes in the WHERE clause in the UPDATE or DELETE statement used to update the database:

```
UPDATE table...
SET column = newvalue
WHERE col1 = value1
AND col2 = value2 ...

DELETE
FROM table
WHERE col1 = value1
AND col2 = value2 ...
```

Using timestamps

SQL Anywhere lets you create special timestamp columns so that you can ensure that users are working with the most current data. If the SELECT statement for the DataWindow object contains a timestamp column, PocketBuilder includes the key column and the timestamp column in the WHERE clause for an UPDATE or DELETE statement regardless of which columns you specify in the Where Clause for Update/Delete box.

If the value in the timestamp column changes (possibly because another user modifies the row), the update fails.

Meanings of WHERE clause options

Choose one of the options in Table 20-1 in the Where Clause for Update/Delete box. The results are illustrated by an example following the table.

Table 20-1: Specifying the WHERE clause for UPDATE and DELETE

Option	Result
Key Columns	The WHERE clause includes the key columns only. These are the columns you specified in the Unique Key Columns box. The values in the originally retrieved key columns for the row are compared against the key columns in the database. No other comparisons are done. If the key values match, the update succeeds. **Caution** Be very careful when using this option. If you tell PocketBuilder to include only the key columns in the WHERE clause and someone else modifies the same row after you retrieve it, those modifications will be overwritten when you update the database. Use this option only with a single-user database or if you are using database locking. In other situations, choose one of the other two options described in this table.
Key and Updatable Columns	The WHERE clause includes all key and updatable columns. The values in the originally retrieved key columns and the originally retrieved updatable columns are compared against the values in the database. If any of the columns have changed in the database since the row was retrieved, the update fails.
Key and Modified Columns	The WHERE clause includes all key and modified columns. The values in the originally retrieved key columns and the modified columns are compared against the values in the database. If any of the columns have changed in the database since the row was retrieved, the update fails.

Example

Consider this situation: a DataWindow object is updating the Employee table, whose key is Emp_ID; all columns in the table are updatable. If the user changes the salary of employee 1001 from $50,000 to $65,000, this is what happens with the different settings for the WHERE clause columns:

- If you selected the Key Columns option for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
```

This statement will succeed whether or not other users have modified the row since your application retrieved it. For example, if another user has modified the salary to $70,000, that change will be overwritten when your application updates the database.

- If you selected Key and Modified Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
   AND Salary = 50000
```

Here the UPDATE statement also checks the original value of the modified column in the WHERE clause. The statement will fail if another user has changed the salary of employee 1001 since your application retrieved the row.

- If you selected Key and Updatable Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
   AND Salary = 50000
   AND Emp_Fname = original_value
   AND Emp_Lname = original_value
   AND Status = original_value
   ...
```

Here the UPDATE statement checks all updatable columns in the WHERE clause. This statement will fail if any of the updatable columns for employee 1001 have been changed since your application retrieved the row.

Specifying update when key is modified

The Key Modification property determines the SQL statements PocketBuilder generates whenever a key column—a column you specified in the Unique Key Columns box—is changed. The options are:

- Use DELETE then INSERT (default)
- Use UPDATE

How to choose a setting

Consider the following when choosing the Key Modification setting:

- If multiple rows are changed, DELETE and INSERT always work. In some DBMSs, UPDATE fails if the user modifies two keys and sets the value in one row to the original value of the other row.
- If only one row can be modified by the user before the database is updated, use UPDATE because it is faster.

Using stored procedures to update the database

Updates to the database can be performed using stored procedures.

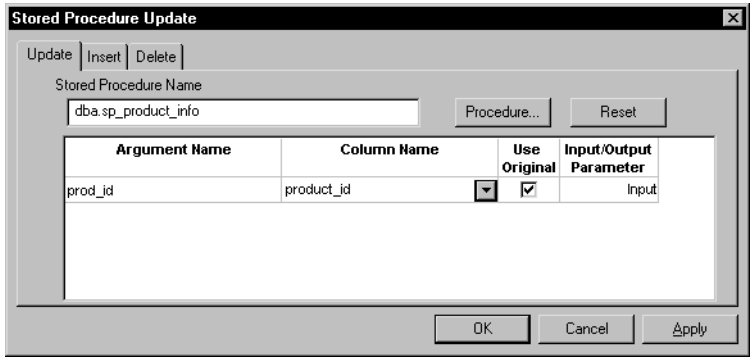
Why use stored procedures?

The DataWindow control submits updates to the database by dynamically generating INSERT, DELETE, and UPDATE SQL statements after determining the status of each row in the DataWindow object. You can also define procedural SQL statements in a stored procedure for use by all applications accessing a database. Using stored procedures to perform database updates allows you to enhance database security, integrity, and performance. Since stored procedures provide for conditional execution, you can also use them to enforce additional business rules.

Specifying stored procedures

The Stored Procedure Update dialog box allows you to associate only an existing stored procedure with your DataWindow object. The stored procedure must have been previously defined in the database.

Figure 20-5: Stored Procedure Update dialog box



❖ **To use stored procedures to update the database**

1 In the DataWindow painter, select Rows>Stored Procedure Update to display the Stored Procedure Update dialog box.

2 Select the tab for the SQL update method (Delete, Insert, or Update) with which you want to associate a stored procedure.

3 Click the Procedure button, select the stored procedure you want executed when the SQL update method is generated, and click OK.

The parameters used in the stored procedure are displayed in the Argument Name list in the order in which they are defined in the procedure. Column Name lists the columns used in your DataWindow object.

4 Select a column in the DataWindow object to associate with a procedure parameter.

If a stored procedure uses parameters that are not matched to column names, you can substitute the value from a DataWindow object computed field or expression.

Matching a column to a procedure parameter
You must be careful to match a column in the DataWindow object correctly to a procedure parameter, since PocketBuilder is able to verify only that datatypes match.

5 If the parameter is to receive a column value, indicate whether the parameter will receive the updated column value entered through the DataWindow object or retain the original column value from the database.

Typically, you select Use Original when the parameter is used in a WHERE clause in an UPDATE or DELETE SQL statement. If you do not select Use Original, the parameter uses the new value entered for that column. Typically, you would use the new value when the parameter is used in an INSERT or UPDATE SQL statement.

What happens when the stored procedure is executed

The stored procedure you associate with a SQL update method in the Stored Procedure Update dialog box is executed when the DataWindow control calls the Update function. The DataWindow control examines the table in the DataWindow object, determines the appropriate SQL statement for each row, and submits the appropriate stored procedure (as defined in the Stored Procedure Update dialog box) with the appropriate column values or expressions substituted for the procedure arguments.

If a stored procedure for a particular SQL update method is not defined, the DataWindow control submits the appropriate SQL syntax in the same manner it has always used.

Return values from stored procedures cannot be handled by the DataWindow control. The Update function returns 1 if it succeeds and -1 if an error occurs. Additional information is returned to SQLCA.

Using Describe and Modify

You can use the DataWindow Describe and Modify functions to access DataWindow property values, including the stored procedures associated with a DataWindow object. For information, see Table.property for the DataWindow object in the *DataWindow Reference* in the online Help.

Restrictions on the use of Modify

Since a database driver can only report stored procedure names and parameter names and position, it cannot verify that changes made to stored procedures are valid. Consequently, if you use Modify to change a stored procedure, be careful that you do not inadvertently introduce changes into the database.

In addition, you must specify the type qualifier first when you use Modify to enable a DataWindow object that is not already using stored procedures to use them to update the database. Calling the type qualifier ensures that internal structures are built before calls are made to Modify. If a new method or method arguments are specified without a preceding definition of type, Modify fails.

CHAPTER 21

Displaying and Validating Data

About this chapter

This chapter describes how to customize your DataWindow object by modifying the display values in columns and specifying validation rules.

Contents

About displaying and validating data

When PocketBuilder generates a basic DataWindow object, it uses the extended attributes defined for the data and stored in the extended attribute system tables.

For more information about the extended attribute system tables, see Appendix A, “Extended Attribute System Tables.”

In the Database painter, you can create the extended attribute definitions that specify a column's display format, edit style, and validation rules.

In the DataWindow painter, you can override these extended attribute definitions for a column in a DataWindow object. These overrides do not change the information stored with the column definition in the extended attribute system tables.

Presenting the data

When you generate a new DataWindow object, PocketBuilder presents the data according to the properties already defined for a column, such as a column's display format and edit style.

Display formats

Display formats embellish data values while still displaying them as letters, numbers, and special characters. Using display formats, for example, you can:

- Change the color of numbers when displaying a negative value
- Add parentheses and dashes to format a telephone number
- Add a dollar sign and period to indicate a currency format

For information, see "About display formats" on page 527.

Edit styles

Edit styles usually take precedence over display formats and specify how column data is presented. For example, using edit styles, you can:

- Display valid values in a drop-down list
- Indicate that a single value is selected by using a check box
- Indicate which one of a group of values is selected by using radio buttons

Edit styles not only affect the way data displays, they also affect how the user interacts with the data at runtime.

For more information, see "About edit styles" on page 539.

Validating data

When data is entered in the Database painter or in a DataWindow object, PocketBuilder evaluates the data against validation rules defined for that column. If the data is valid, PocketBuilder accepts the entry; otherwise, PocketBuilder displays an error message and does not accept the entry.

For more information, see "About validation rules" on page 555.

About display formats

You can use display formats to customize the display of column data in a DataWindow object. Display formats are masks in which certain characters have special significance. For example, you can display currency values preceded by a dollar sign, show dates with month names spelled out, and use a special color for negative numbers. You can use the many predefined display formats provided with PocketBuilder or define your own.

The DataWindow object in Figure 21-1 uses no display formats; all values display as they are stored in the database.

Figure 21-1: DataWindow object without display formats

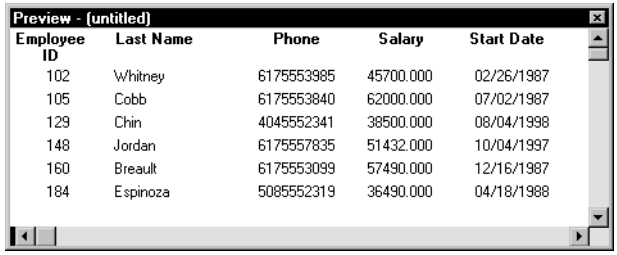
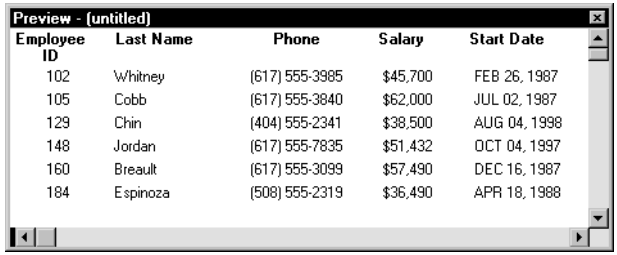


Figure 21-2 shows the same DataWindow object with display formats for the Phone, Salary, and Start Date columns, that make the data easier to interpret.

Figure 21-2: DataWindow object with display formats



Display formats not used for data entry
When users tab to a column containing a display format, PocketBuilder removes the display format and displays the raw value for users to edit.

If you want to provide formatting that is displayed during data entry, you need to specify edit masks, as described in “The EditMask edit style” on page 547.

About display format masks and EditMasks

The differences between display format masks and EditMask edit styles can be hard to understand. A display format mask determines the appearance of the column when the focus is *off* the column, or when the DataWindow object is in print preview mode. When you apply an EditMask edit style, the mask you use determines the appearance of the column when focus is *on* the column.

If you want data to display differently depending on whether the focus is on or off the column, specify an edit mask (on the Edit properties page for the column) as well a display format (on the Format properties page for the column), then check the Use Format check box on the Format properties page. The Use Format check box displays only when an edit mask has been specified.

Working with display formats

You work with display formats in the Database painter and the DataWindow painter.

Display formats and the extended attribute system tables

When you place a column in a DataWindow object and have given it a display format (either the default format from the assignment made in the Database painter for the column or a format assigned in the DataWindow painter), no link is maintained to the named format in the extended attribute system tables.

If the definition of the display format later changes in the extended attribute system tables, the format for the column in a DataWindow object does not change. If you want to use the modified format, you can reapply it to the column in the DataWindow painter.

Working with display formats in the Database painter

Typically, you define display formats and associate them with columns in the Database painter, because display formats are properties of the data itself. After you have associated a display format with a column in the Database painter, it is used by default each time the column is placed in a DataWindow object.

Edit style takes precedence
If a column has an associated edit style, the edit style takes precedence over a display format unless you use an EditMask edit style and check the Use Format box on the Format properties page.

For more information, see "About edit styles" on page 539.

What you do in the Database painter

In the Database painter, you can:

- Create, modify, and delete named display formats

  The named display formats are stored in the extended attribute system tables. When you have defined a display format, it can be used by any column of the appropriate datatype in the database.

- Assign display formats to columns and remove display formats associated with columns

  Formats that you associate with columns are used by default when you place the columns in a DataWindow object in the DataWindow painter.

❖ **To create a new display format:**

1 In the Database painter, select Object>Insert>Display Format from the menu bar.

  The Display Format view displays.

2 Name the display format and specify a datatype.

3 Define the display format using masks.

  You can use this display format with any column of the appropriate datatype in the database.

  For information, see "Defining display formats" on page 532.

❖ **To modify an existing display format:**

1 In the Database painter, open the Extended Attributes view.

2 In the Extended Attributes view, open the list of display formats.

3 Position the pointer on the display format you want to modify, display the pop-up menu, and select Properties.

4 In the Display Format view, modify the display format as desired.

  For information, see "Defining display formats" on page 532.

❖ **To associate a display format with a column in the Database painter:**

1 In the Database painter Objects view, position the pointer on the column, select Properties from the pop-up menu, and select the Display tab in the Properties view.

2 Select a format from the list in the Display Format box.

The column now has the selected format associated with it in the extended attribute system tables.

❖ **To remove a display format from a column in the Database painter:**

1 In the Database painter Objects view, position the pointer on the column, select Properties from the pop-up menu, and select the Display tab in the Properties view.

2 Select (None) from the list in the Display Format box.

The display format is no longer associated with the column.

Working with display formats in the DataWindow painter

Changing the display format assigned to a column

Display formats you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the default format in the DataWindow painter by choosing another format from the extended attribute system tables or defining an ad hoc format for one specific column.

About computed fields
You can assign display formats to computed fields using the same techniques as for columns in a table.

What you do in the DataWindow painter

In the DataWindow painter, you can:

- Accept the default display format assigned to a column in the Database painter
- Override the default display format with another named format stored in the extended attribute system tables
- Create an ad hoc, unnamed format to use with one specific column

❖ **To change a display format for a column in the DataWindow painter:**

1 In the DataWindow painter, move the pointer to the column, select Properties from the column's pop-up menu, and then select the Format tab.

Information appropriate to the datatype of the selected column displays. The currently used format displays in the Format box. All formats for the datatype defined in the extended attribute system tables are listed in the pop-up list (displayed by clicking the button next to the Format box).

2 Do one of the following:

- Remove the display format from the column by clearing the Format box.
- Select a format in the extended attribute system tables from the pop-up list.
- Create a format for the column by typing it in the Format box. For more information, see "Defining display formats" next.

Format not saved in the extended attribute system tables
If you create a format here, it is used only for the current column and is not saved in the extended attribute system tables.

Using toolbar shortcuts

To assign the Currency or Percent display format to a numeric column in a report, select the column, then click the Currency or Percent button in the PainterBar, or select Format>Currency or Format>Percent from the menu bar.

You can add buttons to the PainterBar that assign a specified display format to selected columns in reports.

For more information, see the section on customizing toolbars in Chapter 2, "Customizing PocketBuilder."

Defining display formats

Display formats are represented through masks, where certain characters have special significance. PocketBuilder supports four kinds of display formats, each using different mask characters:

Numbers
Strings
Dates
Times

For example, in a string format mask, each @ represents a character in the string and all other characters represent themselves. You can use the following mask to display phone numbers:

```
(@@@) @@@-@@@@
```

Combining formats

You can include different types of display format masks in a single format. Use a space to separate the masks. For example, the following format section includes a date and time format:

```
mmmm/dd/yyyy h:mm
```

Using sections

Each type of display format can have multiple sections, with each section corresponding to a form of the number, string, date, or time. Only one section is required; additional sections are optional and should be separated with semicolons (;).

The following format specifies different displays for positive and negative numbers; negative numbers are displayed in parentheses:

```
$#,##0;($#,##0)
```

Using keywords

Enclose display format keywords in square brackets. For example, you can use the keyword [General] when you want PocketBuilder to determine the appropriate format for a number.

Using colors

You can define a color for each display format section by specifying a color keyword before the format. The color keyword is the name of the color, or a number that represents the color, enclosed in square brackets: [RED] or [255]. The number is usually used only when a color is required that is not provided by name. The named color keywords are:

[BLACK]
[BLUE]
[CYAN]
[GREEN]
[MAGENTA]
[RED]

[WHITE]
[YELLOW]

The formula for combining primary color values into a number is:

256*256**blue** + 256**green** + **red**=**number**

where the amount of each primary color is specified as a value from 0 to 255. For example, to specify cyan, substitute 255 for blue, 255 for green, and 0 for red. The result is 16776960.

Table 21-1 lists the blue, green, and red values you can use in the formula to create other colors.

Table 21-1: Numeric values used to create colors

Blue	Green	Red	Number	Color
0	0	255	255	Red
0	255	0	65280	Green
0	128	0	32768	Dark green
255	0	0	16711680	Blue
0	255	255	65535	Yellow
0	128	128	32896	Brown
255	255	0	16776960	Cyan
192	192	192	12632256	Light gray

Using special characters

To include a character in a mask that has special meaning in a display format, such as a square bracket ([ ), precede the character with a backslash (\). For example, to display a single quotation mark, enter \'.

Setting display formats during execution

In scripts, you can use GetFormat to get the current format for a column and SetFormat to change the format for a column during execution.

Number display formats

A number display format can have up to four sections. Only the first is required. The three other sections determine how the data displays if its value is negative, zero, or NULL. The sections are separated by semi-colons:

Positive-format;negative-format;zero-format;null-format

Special characters

Table 21-2 lists characters that have special meaning in number display formats.

Table 21-2: Characters with special meaning in display formats

Character	Meaning
#	A number
0	A required number; a number will display for every 0 in the mask

Percent signs, decimal points, parentheses, and spaces display as entered in the mask.

Use at least one 0

In general, a number display format should include at least one 0. If users enter 0 in a field with the mask ###, the field will appear to be blank if you do not provide a zero-format section. If the mask is ###.##, only the period displays. If you want two decimal places to display even if both are 0, use the mask ##0.00.

Number keywords

You can use the following keywords as number display formats when you want PocketBuilder to determine an appropriate format to use:

- [General]
- [Currency]

Note that [Currency(7)] and [Currency(n)] can be used as edit masks, but they are not permitted as display formats.

Number and currency settings

To ensure that an application behaves the same in every country in which it is deployed, DataWindow expressions and the masks used in display formats and edit masks require U.S. notation for numbers. That is, when you specify a number in a DataWindow expression or in a number mask, a comma always represents the thousands delimiter and a period always represents the decimal place. You should also always use the $ sign to represent the symbol for currency.

At runtime, the locally correct symbols are displayed for numbers and currency. The comma and period are replaced by the delimiters defined in the user's Number settings in the Regional or International Settings property sheet in the Windows Control Panel. The $ sign in the mask is replaced by the local currency symbol as defined in the user's Currency setting in the Windows Control Panel. For example, in countries where a comma represents the decimal place and a period represents thousands, users see numbers in those formats.

Percentages

When you enter a number in a column with a percent edit mask and tab off the column, PocketBuilder divides the number by 100 and stores the result in the buffer. For example, if you enter 23, PocketBuilder passes .23 to the buffer. When you retrieve from the database, PocketBuilder multiplies the number by 100 and, if the mask is ##0%, displays 23%.

Use caution when defining an edit mask for a percentage. The datatype for the column must be numeric or decimal to handle the result of a division by 100. If the column has an integer datatype, a percentage entered as 333 is retrieved from the database as 300, and 33 is retrieved as 0.

If you use an edit mask with decimals, such as ##0.00%, the datatype must have enough decimal places to handle the division. For example, if you enter 33.33, the datatype for the column must have at least four decimal places because the result of the division is .3333. If the datatype has only three decimal places, the percentage is retrieved as 33.30.

Examples

Table 21-3 shows how the values 5, –5, and .5 display when different format masks are applied.

Table 21-3: Number display format examples

Format	5	-5	.5
[General]	5	-5	0.5
0	5	-5	1
0.00	5.00	-5.00	0.50
#,##0	5	-5	1
#,##0.00	5.00	-5.00	0.50
$#,##0;($#,##0)	$5	($5)	$1
$#,##0;-$#,##0	$5	-$5	$1
$#,##0;[RED]($#,##0)	$5	($5)	$1
[Currency]	$5.00	($5.00)	$0.50
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50
$#,##0.00;[RED]($#,##0.00)	$5.00	($5.00)	$0.50
##0%	500%	-500%	50%
##0.00%	500.00%	-500.00%	50.00%
0.00E+00	5.00E+00	-5.00E+00	5.00E-01

String display formats

String display formats can have two sections. The first is required and contains the format for strings; the second is optional and specifies how to represent NULLs:

string-format;null-format

In a string format mask, each at-sign (@) represents a character in the string and all other characters represent themselves.

Example

This format mask:

```
[red](@@@) @@@-@@@@
```

displays the string `800YESCELT` in red as:

```
(800) YES-CELT
```

Date display formats

Date display formats can have two sections. The first is required and contains the format for dates; the second is optional and specifies how to represent NULLs:

date-format;null-format

Special characters

Table 21-4 shows characters that have special meaning in date display formats.

Table 21-4: Characters with special meaning in data display formats

Character	Meaning	Example
d	Day number with no leading zero	9
dd	Day number with leading zero if appropriate	09
ddd	Day name abbreviation	Mon
dddd	Day name	Monday
m	Month number with no leading zero	6
mm	Month number with leading zero if appropriate	06
mmm	Month name abbreviation	Jun
mmmm	Month name	June
yy	Two-digit year	97
yyyy	Four-digit year	1997

Colons, slashes, and spaces display as entered in the mask.

About 2-digit years
If users specify a 2-digit year in a DataWindow object, PocketBuilder assumes the date is the 20th century if the year is greater than or equal to 50. If the year is less than 50, PocketBuilder assumes the 21st century. For example:

- 1/1/85 is interpreted as January 1, 1985.
- 1/1/40 is interpreted as January 1, 2040.

Date keywords

You can use the following keywords as date display formats when you want PocketBuilder to determine an appropriate format to use:

- [ShortDate]
- [LongDate]

The format used is determined by the regional settings for date in the registry. Note that [Date] is not a valid display format.

Examples

Table 21-5 shows how the date Friday, January 30, 2008, displays when different format masks are applied.

Table 21-5: Date display format examples

Format	Displays
[red]m/d/yy	1/30/08 in red
d-mmm-yy	30-Jan-08
dd-mmmm	30-January
mmm-yy	Jan-08
dddd, mmm d, yyyy	Friday, Jan 30, 2008

Time display formats

Time display formats can have two sections. The first is required and contains the format for times; the second is optional and specifies how to represent NULLs:

```
time-format;null-format
```

Special characters

Table 21-6 shows characters that have special meaning in time display formats.

Table 21-6: Characters with special meaning in time display formats

Character	Meaning
h	Hour with no leading zero (for example, 1)
hh	Hour with leading zero if appropriate (for example, 01)
m	Minute with no leading zero (must follow h or hh)
mm	Minute with leading zero if appropriate (must follow h or hh)
s	Second with no leading zero (must follow m or mm)
ss	Second with leading zero (must follow m or mm)
ffffff	Microseconds with no leading zeros. You can enter one to six f's; each f represents a fraction of a second (must follow s or ss)
AM/PM	Two-character, uppercase abbreviation (AM or PM as appropriate)
am/pm	Two-character, lowercase abbreviation (am or pm as appropriate)
A/P	One-character, uppercase abbreviation (A or P as appropriate)
a/p	One-character, lowercase abbreviation (a or p as appropriate)

Colons, slashes, and spaces display as entered in the mask.

24-hour format is the default
Times display in 24-hour format unless you specify AM/PM, am/pm, A/P, or a/p.

Time keyword

You can use the following keyword as a time display format to specify the format specified in the Windows control panel:

- [Time]

Examples

Table 21-7 shows how the time 9:45:33:234567 PM displays when different format masks are applied.

Table 21-7: Time display format examples

Format	Displays
h:mm AM/PM	9:45 PM
hh:mm A/P	09:45 P
h:mm:ss am/pm	9:45:33 pm
h:mm	21:45
h:mm:ss	21:45:33
h:mm:ss:f	21:45:33:2
h:mm:ss:fff	21:45:33:234
h:mm:ss:ffffff	21:45:33:234567
m/d/yy h:mm	1/30/98 21:45

About edit styles

You can define edit styles for columns. Edit styles specify how column data is presented in DataWindow objects. Unlike display formats, edit styles do not only affect the display of data; they also affect how users interact with the data at runtime. Once you define an edit style, it can be used by any column of the appropriate datatype in the database.

When edit styles are used

If both a display format and an edit style have been assigned to a column, the edit style is always used, with one exception. When you assign an EditMask edit style to a column, you can check the Use Format check box on the Format properties page for the column to use the edit mask format when focus is on the column, and the display format mask when focus is off the column.

Edit styles

Table 21-8 shows the available edit styles.

Table 21-8: Edit styles

Edit style	What the edit style does	Example
Edit box (default)	Displays a value in the box. For data entry, type a value.	Boston
DropDownListBox	Displays a value from the drop-down list. For data entry, select or enter a value.	Small Small Medium Large One size fits all
CheckBox	Displays a check box selected or cleared. For data entry, select or clear the check box.	Day Care
RadioButtons	Displays radio buttons, one of which is selected. For data entry, select one of the radio buttons.	Male Female
EditMask	Displays formatted data. For data entry, type a value.	$62,000.00
DropDownDataWindow	Displays a value from a drop-down DataWindow. For data entry, select a value.	100 R & D Sales Finance

For example, suppose you have a column Status that takes one of three values: the letters A, T, and L, each representing a status (Active, Terminated, or On Leave). If you assign it the RadioButton edit style, users can simply click a button instead of having to type A, T, or L. You do not have to create a validation rule to validate typed input.

Working with edit styles

You work with edit styles in the Database painter and DataWindow painter.

Edit styles and the extended attribute system tables

When you place a column in a DataWindow object and give it an edit style (either the default style from the assignment made in the Database painter for the column or a style assigned in the DataWindow painter), PocketBuilder records the name and definition of the edit style in the DataWindow object.

However, if the definition of the edit style later changes in the extended attribute system tables, the edit style for the column in a DataWindow object does not change automatically. You can update the column by reassigning the edit style to it in the DataWindow object.

Working with edit styles in the Database painter

Typically, you define edit styles in the Database painter, because edit styles are properties of the data itself. Once defined in the Database painter, the styles are used by default each time the column is placed in a DataWindow object.

What you do in the Database painter

In the Database painter, you can:

- Create, modify, and delete named edit styles

  The edit styles are stored in the extended attribute system tables. Once you define an edit style, it can be used by any column of the appropriate datatype in the database.

- Assign edit styles to columns

  These styles are used by default when you place the column in a DataWindow object in the DataWindow painter.

❖ **To create a new edit style:**

1. In the Database painter, select Object>Insert>Edit Style from the menu bar.
2. In the Edit Style dialog box, select the edit style type from the Style drop-down list.
3. Specify the properties of the edit style and click OK.

   For information, see “Defining edit styles” on page 542.

   You can use the new edit style with any column of the appropriate datatype in the database.

❖ **To modify an existing edit style:**

1. In the Database painter, open the Extended Attributes view.
2. In the Extended Attributes view, open the list of edit styles.
3. Position the pointer on the Edit style you want to modify, display the pop-up menu, then select Properties.
4. In the Edit Style dialog box, modify the edit style as desired and click OK.

   For information, see "Defining edit styles" on page 542.

   You can use the modified edit style with any column of the appropriate datatype in the database.

❖ **To associate an edit style with a column in the Database painter:**

1. In the Database painter (Objects view), position the pointer on the column, select Properties from the pop-up menu, then select the Edit Style tab in the Properties view.
2. Select a style for the appropriate datatype from the list in the Style Name box.

   PocketBuilder associates the selected edit style with the column in the extended attribute system tables.

❖ **To remove an edit style from a column in the Database painter:**

1. In the Database painter (Objects view), position the pointer on the column, select Properties from the pop-up menu, then select the Edit Style tab in the Properties view.
2. Select (None) from the list in the Style Name box.

   The edit style is no longer associated with the column.

Working with edit styles in the DataWindow painter

An edit style you assign to a column in the Database painter is used by default when you place the column in a DataWindow object. You can override the edit style in the DataWindow painter by choosing another edit style from the extended attribute system tables or defining an ad hoc style for one specific column.

What you do in the DataWindow painter

In the DataWindow painter, you can:

- Accept the default edit style assigned to a column in the Database painter
- Override the default edit style with another named style stored in the extended attribute system tables
- Create an ad hoc, unnamed edit style to use with one specific column

❖ **To specify an edit style for a column:**

1 In the DataWindow painter, move the pointer to the column, select Properties from the column's pop-up menu, and then select the Edit tab.

2 Select the type of edit style you want from the Style Type drop-down list.

The information on the Edit properties page changes to apply to the type of edit style you selected.

3 Do one of the following:

- Select an edit style from the Style Name box
- Create an ad hoc edit style for the column, as described in "Defining edit styles" next

Defining edit styles

This section describes how to specify each type of edit style:

- The Edit edit style
- The DropDownListBox edit style
- The CheckBox edit style
- The RadioButtons edit style
- The EditMask edit style
- The DropDownDataWindow edit style

The Edit edit style

By default, columns use the Edit edit style, which displays data in an edit control. You can customize the appearance and behavior of the edit control by modifying a column's Edit edit style:

- To restrict the number of characters users can enter, enter a value in the Limit box
- To convert the case of characters upon display, enter an appropriate value in the Case box
- To have entered values display as asterisks for sensitive data, check the Password box
- To allow users to tab to the column but not change the value, check the Display Only box
- To define a code table to determine which values are displayed to users and which values are stored in the database, check the Use Code Table box and enter display and data values for the code table

  See "Defining a code table" on page 552.

❖ **To use the Edit edit style:**

1 Select Edit from the Style Type box, if it is not already selected.

2 Select the properties you want.

Date columns and regional settings

Using the Edit edit style, or no edit style, with a date column can cause serious data entry and validation problems if a user's computer is set up to use a nonstandard date style, such as yyyy/dd/mm. For example, if you enter 2001/03/05 in the Retrieval Arguments dialog box for a date column when the mask is yyyy/dd/mm, the date is interpreted as March 5 instead of May 3. To ensure that the order of the day and month is interpreted correctly, use an EditMask edit style.

The DropDownListBox edit style

You can use the DropDownListBox edit style to have columns display as drop-down lists during execution.

Typically, this edit style is used with code tables, where you can specify display values that users see and shorter data values that are stored in the database.

In the DropDownListBox edit style, the display values of the code table display in the ListBox portion of the DropDownListBox. The data values (not the display values) are the values that are put in the DataWindow buffer when the user selects an item in the ListBox portion of the drop-down list. These values are sent to the database when an Update is issued.

For example, when a user selects the value Business Services in Figure 21-3, the corresponding data value could be a department number such as 200.

Figure 21-3: Example of a drop-down list edit style

❖ **To use the DropDownListBox edit style:**

1 Select DropDownListBox from the Style Type box for a DataWindow column.

   The Style Type box is on the Edit page of the Properties view for the column.

2 Select other properties on the Edit page of the Properties view.

3 Enter values you want to have appear in the Display Value box and corresponding data values in the Data Value box.

During execution

You can define and modify a code table for a column in a script by using the SetValue function during execution. To obtain the value of a column during execution, use the GetValue function. To clear the code table of values, use the ClearValues function.

For more about code tables, see “Defining a code table” on page 552.

The CheckBox edit style

If a column can take only one of two (or perhaps three) values, you might want to display the column as a check box; users can select or clear the check box to specify a value.

❖ **To use the CheckBox edit style:**

1 Select CheckBox from the Style Type box for a DataWindow column.

2 If you want a label as part of the CheckBox style for the column, enter the label for the check box in the Text box.

By default, the label displays to the right of the check box when you make the column wide enough to accomodate both the check box and the label. If you want the label to display to the left of the check box, select the Left Text check box on the Edit page of the Properties view for the column.

Using accelerator keys
You can designate an accelerator key in a CheckBox label by including an ampersand (&) before the letter that you want to use for the accelerator. On the desktop, the Alt key in combination with the accelerator key changes focus to the CheckBox control and changes the value of the control.

On a Windows CE device, the SIP keyboard allows users to enter an accelerator key that changes the value of the check box. However, because it does not have an Alt key, the SIP keyboard cannot be used to change focus to the check box control.

3 Enter the values you want to have placed in the DataWindow buffer:

- In the Data Value For On box, enter the value for when the CheckBox control is checked
- In the Data Value For Off box, enter the value for when the CheckBox control is not checked

4 (Optional) Select the 3 States check box, then enter a value for a third CheckBox state in the Other State box.

The Other State box does not display unless you select the 3 States check box.

What happens

The value you enter in the Text box becomes the display value, and values entered for On, Off, and Other become the data values.

When users check or clear the check box at runtime, PocketBuilder enters the appropriate data value in its buffer. When the Update function is issued, PocketBuilder sends the corresponding data values to the database.

Centering check boxes without text

You might find it useful to center check boxes used for columns of information. First make the text control used for the column header and the column control the same size and left aligned. Then you can center the check boxes and the column header.

❖ **To center check boxes without text:**

1 In the Edit properties page for the column, make sure the Left Text check box is not selected and that the Text box where you specify associated text is empty.

2 Specify centering (Alignment>Center) in the General properties page, or specify centering using the StyleBar.

The RadioButtons edit style

If a column can take one of a small number of values, you might want to display the column as radio buttons.

❖ **To use the RadioButtons edit style:**

1 Select RadioButtons from the Style Type box for a DataWindow column.

2 Specify how many radio buttons will display in the Columns Across box.

3 Enter a set of display and data values for each button you want to display.

The display values you enter become the text of the buttons; the data values are put in the DataWindow buffer when the button is clicked.

Using accelerator keys
To use an accelerator key on a radio button, enter an ampersand (&) in the Display Value before the letter that will be the accelerator key. On Windows CE devices, you can use the SIP keyboard to enter the accelerator key and change the radio button selection. However, there is no Alt key on the SIP keyboard that would allow users to change focus to the radio button control.

What happens

Users select values by clicking a radio button. When the Update function is issued, the data values are sent to the database.

The EditMask edit style

Sometimes users need to enter data that has a fixed format. For example, in North America phone numbers have a 3-digit area code, followed by three digits, followed by four digits. You can define an edit mask that specifies the format to make it easier for users to enter values.

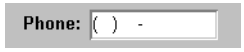
Edit masks consist of special characters that determine what can be entered in the column. They can also contain punctuation characters to aid users.

For example, to make it easier for users to enter North American phone numbers in the proper format, specify this mask:

```
(###) ###-####
```

During execution, the punctuation characters display in the box and the cursor jumps over them as the user types.

Figure 21-4: Example of EditMask edit style display before data entry



Special characters and keywords

Edit masks use the same special characters as display formats, and there are special considerations for using numeric, string, date, and time masks.

For information, see “Defining display formats” on page 532.

Keyboard behavior

Note that certain keystrokes from the SIP in edit masks behave as follows:

- The Backspace key deletes the preceding character
- Delete (Shift + Backspace) deletes the character after the current location
- Both Backspace and Delete delete everything that is selected
- Non-numeric edit masks treat any characters that do not match the mask pattern as delimiters

Also, note this behavior in Date edit masks:

- The strings 00/00/00 or 00/00/0000 are interpreted as the NULL value for the column.

Using the Mask pop-up menu

Click the button to the right of the Mask box on the Mask properties page to display a list that contains complete masks, as well as special characters that you can use to construct your own mask. For example, the menu for a Date edit mask contains complete masks such as mm/dd/yy and dd/mmm/yyyy. It also has components such as dd and jjj (for a Julian day). You might use these to construct a mask like dd-mm-yy, typing in the hyphens as separators.

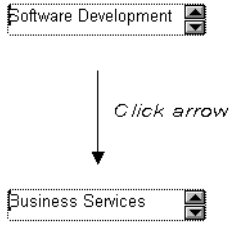
You cannot use a partial mask, such as dd or mmm, in a date edit mask. Any mask that does not include any characters representing the year will be replaced by a mask that does.

Using spin controls

You can define an edit mask as a spin control, which is a box with up and down arrows that users can click to cycle through fixed values. For example, you can set up a code table that provides the valid entries in a column; users simply click an arrow to select an entry. Used this way, a spin control works like a drop-down list that displays one value at a time.

For more about code tables, see "Defining a code table" on page 552.

Figure 21-5: Effect of clicking arrow on a spin control edit mask



❖ **To use an EditMask edit style:**

1 Select EditMask in the Style Type box if it is not already selected.

2 Define the mask in the Mask box.

You can select a named style in the Style Name box, which automatically places an edit mask in the mask box, or you can click the button to the right of the Mask box, then click the special characters that you want from the pop-up menu to use them in the mask. You can also type a mask in the Mask box.

3 Specify other properties for the edit mask.

When you use your EditMask, check its appearance and behavior. If characters do not appear as you expect, you might want to change the font size or the size of the EditMask.

The DropDownDataWindow edit style

Sometimes another data source determines which data is valid for a column.

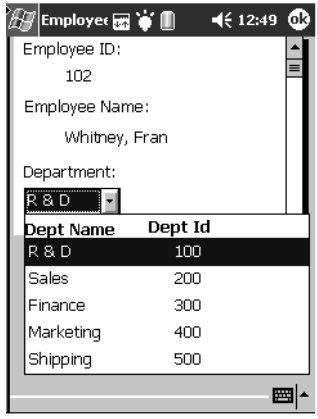
Advantage of a dynamic edit style

Consider this situation: the Department table includes two columns, Dept_id and Dept_name, listing your company's departments. The Employee table lists your employees. The Department column in the Employee table can have any of the values from the Dept_id column in the Department table.

As new departments are added to your company, you want the DataWindow object containing the Employee table to automatically provide the new departments as choices when users enter values in the Department column.

In situations such as this, you can specify the DropDownDataWindow edit style for a column: it is populated from another DataWindow object. When users go to the column, the contents of the DropDownDataWindow display, showing the latest data.

Figure 21-6: Example of a DropDownDataWindow edit style



In Figure 21-6, a DataWindow object contains a DropDownDataWindow edit style that displays a different DataWindow object containing only the Dept_ID and Dept_Name columns from the Department table. Typically only a single column is displayed in a drop-down DataWindow. The example in the figure displays both columns from the DataWindow because the drop-down width property for the column using the DropDownDataWindow style has been set to more than 100%.

Example used in procedure

In the following procedure, you create a DataWindow similar to the DataWindow object used in Figure 21-6. You name the DataWindow d_dddw_dept and save it for use in a DropDownDataWindow style. You then create another DataWindow from the Employee table and use the Properties view in the DataWindow painter to associate the Dept_ID column in the new DataWindow with an ad hoc DropDownDataWindow style. The ad hoc style will display the Dept_ID column from the Department table that is used by the d_dddw_dept DataWindow object.

You can also create a named DropDownDataWindow edit style in the Database painter that uses the d_dddw_dept DataWindow object. If you create a named style, and then select it in the DataWindow painter for a column in another DataWindow, properties that you set for the style automatically populate the fields on the Edit page of the Properties view for that column.

For information on creating a named edit style, see "Working with edit styles in the Database painter" on page 540.

❖ **To use the DropDownDataWindow edit style:**

1. Create a DataWindow object with columns that you want to use in a drop-down list, and save the DataWindow.

   You will often choose at least two columns for the DataWindow: one column that contains values that the user sees and another column containing values to be stored in the database. For example, to create the d_dddw_dept DataWindow object, create a tabular DataWindow that contains only the Dept_ID and Dept_Name columns from the Department table in the ASA Sample database.

2. Select the column for which you want to obtain data from a DataWindow object, click the Edit tab in the Properties view of the DataWindow painter, then select the DropDownDW edit style in the Style Type box.

   For example, create a DataWindow from the Employee table, then select the Dept_ID column and specify the DropDownDW edit style for this column.

3. In the Lines in DropDown box, type the number of rows you want to see in the drop-down DataWindow.

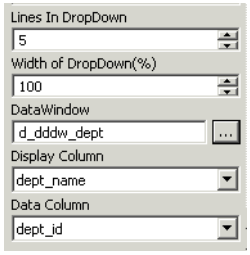
   There are five rows in the Dept_ID column, so you might consider typing 5 for the number of lines you want to see. You might also consider selecting the V ScrollBar check box on the Edit tab, so that when you add a sixth or seventh row, users can scroll to see the entire list of departments.

4 In the Width of DropDown (%) box, type `100` to display the complete width of a single column you select as the display column.

5 Click the ellipsis button next to the DataWindow box and select the DataWindow object that contains the data you want to use to populate the drop-down list.

Continuing with the same example, you would select d_dddw_dept. After you select a DataWindow object, two more fields are added to the Edit tab: one for a display column, and the other for a data column.

Figure 21-7: Setting properties for a DropDownDataWindow edit style



6 In the Display Column box, select the column containing the values that you want to display in the DataWindow object.

In this example, you could choose to display the Dept_Name column, instead of the Dept_ID column, from the drop-down DataWindow d_dddw_dept.

7 In the Data Column box, select the column containing the values that will be stored in the database.

Here you would probably want the Dept_ID as the data column.

8 Specify other properties for the edit style and click OK when done.

What happens

During execution, when data is retrieved into the DataWindow object, the column whose edit style is DropDownDataWindow will itself be populated. This will occur as data is retrieved into the DataWindow object that is serving as the drop-down DataWindow object.

When the user clicks the down arrow next to the column at runtime, the contents of the drop-down DataWindow object display. When the user selects a display value, the corresponding data value is stored in the DataWindow buffer and is stored in the database when an Update command is issued.

Limit on size of data value
The data value for a column that uses the DropDownDataWindow edit style is limited to 511 characters.

Defining a code table

To reduce storage needs, you might frequently want to store short, encoded values in the database, but these encoded values might not be meaningful to users. To make DataWindow objects easy to use, you can define code tables.

Each row in a code table is a pair of corresponding values: a display value and a data value. The display values are those users see during execution. The data values are those saved in the database.

Limit on size of data value
The data value you specify for the Checkbox, DropDownListBox, Edit, EditMask, and RadioButtons edit styles is limited to 255 characters.

How code tables are implemented

You can define a code table as a property of the following column edit styles:

Edit
DropDownListBox
RadioButtons
DropDownDataWindow
EditMask, using spin control

The steps for specifying the code table property for each edit style are similar: you begin by defining a new edit style in the Database painter. Once you select an edit style, use the specific procedure that follows to define the code table property.

For information on how to create an edit style, see “Working with edit styles in the Database painter” on page 540.

Allowing NULL values
An internal PocketBuilder code, NULL!, indicates NULL values are allowed. To use this code, specify NULL! as the data value, then specify a display format for NULLs for the column.

❖ **To define a code table as a property of the Edit edit style:**

1 Select the Use Code Table check box.

2 Enter the display and data values for the code table.

3 If you want to restrict input in the column to values in the code table, select the Validate check box.

For more information, see "Validating user input" on page 555.

❖ **To define a code table as a property of the DropDownListBox edit style:**

1 Enter the display and data values for the code table.

2 If you want to restrict input in the column to values in the code table, clear the Allow Editing check box.

For more information, see "Validating user input" on page 555.

❖ **To define a code table as a property of the RadioButtons edit style:**

1 Enter the display and data values for the code table.

2 If you want the radio buttons to display on a single line, enter the number of rows in the Columns Across box that you entered in the code table.

❖ **To define a code table as a property of the DropDownDataWindow edit style:**

1 Specify the column that provides the display values in the Display Column box.

2 Specify the column that provides the data values in the Data Column box.

3 If you want to restrict input to values in the code table, clear the Allow Editing check box.

❖ **To define a code table as a property of the EditMask edit style:**

1 Select the Spin Control check box.

2 Select the Code Table check box.

3 Enter the display and data values for the code table.

How code tables are processed

When data is retrieved into a DataWindow object column with a code table, processing begins at the top of the data value column. If the data matches a data value, the corresponding display value displays. If there is no match, the actual value displays.

Consider the example in Table 21-9.

Table 21-9: Data values and display values

Display values	Data values
Massachusetts	MA
Massachusetts	ma
ma	MA
Mass	MA
Rhode Island	RI
RI	RI

If the data is MA or ma, the corresponding display value (Massachusetts) displays. If the data is Ma, there is no match, so Ma displays.

Case sensitivity
Code table processing is case sensitive.

If the code table is in a DropDownListBox edit style, and if the column has a code table that contains duplicate display values, each value displays only once. So if this code table (Table 21-9) is defined for a column in a DataWindow object that has a DropDownListBox edit style, Massachusetts and Rhode Island display in the ListBox portion of the DropDownListBox.

For information about how a user selection is matched to a data value when a display value corresponds to multiple data values, see “Validating user input” next.

Validating user input

When users enter data into a column in a DataWindow object, processing begins at the top of the display value column of the associated code table.

If the data matches a display value, the corresponding data value is put in the internal buffer. For each display value, the first data value is used. Using the sample code table, if the user enters Massachusetts, ma, or Mass, MA is the data value.

You can specify that only the values in the code table are acceptable:

- For a column using the Edit edit style, select the Validate check box

  If you have selected the Validate check box for the Edit edit style, an ItemError event is triggered whenever a user enters a value not in the code table. Otherwise, the entered value is validated using the column's validation rule, if any, and put in the DataWindow buffer.

- For the DropDownListBox and DropDownDataWindow edit styles, clear the Allow Editing check box to prevent users from typing a value

  Although users cannot type a value when you clear the Allow Editing check box, they can search for a row in the drop-down list or DataWindow by typing in the initial character for the row display value. The search is case sensitive. For the DropDownDataWindow edit style, the initial character for a search cannot be an asterisk or a question mark. This restriction does not apply to the DropDownListBox edit style.

When the code table processing is complete, the ItemChanged or ItemError event is triggered.

Code table data
The data values in the code table must pass validation for the column and must have the same datatype as the column.

About validation rules

When users enter data in a DataWindow object, you want to be sure the data is valid before using it to update the database. One way to do this is through validation rules.

You usually define validation rules in the Database painter. To use a validation rule, you associate it with a column in the Database painter or DataWindow painter.

Another technique
You can also perform data validation through code tables, which are implemented through a column's edit style.

For more information, see “About edit styles” on page 539.

Understanding validation rules

Validation rules are criteria that a DataWindow object uses to validate data entered into a column by users. They are specific to PocketBuilder and therefore not enforced by the DBMS.

Validation rules assigned in the Database painter are used by default when you place columns in a DataWindow object. You can override the default rules in the DataWindow painter.

A validation rule is an expression that evaluates to either TRUE or FALSE. If the expression evaluates to TRUE for an entry into a column, PocketBuilder accepts the entry. If the expression evaluates to FALSE, the entry is not accepted and the ItemError event is triggered. By default, PocketBuilder displays a message box to the user.

Figure 21-8: Message box triggered by validation rule violation

You can customize the message displayed when a value is rejected. You can also code an ItemError script to cause different processing to happen.

For more information, see the chapter on using DataWindow objects in the *Resource Guide*.

At runtime
In scripts, you can use the GetValidate function to obtain the validation rule for a column and the SetValidate function to change the validation rule for a column.

For information about the GetValidate and SetValidate functions, see the *DataWindow Reference* in the online Help.

Working with validation rules

You work with validation rules in the Database painter and DataWindow painter.

Defining validation rules

Typically, you define validation rules in the Database painter, because validation rules are properties of the data itself. Once defined in the Database painter, the rules are used by default each time the column is placed in a DataWindow object. You can also define a validation rule in the DataWindow painter that overrides the rule defined in the Database painter.

Validation rules and the extended attribute system tables

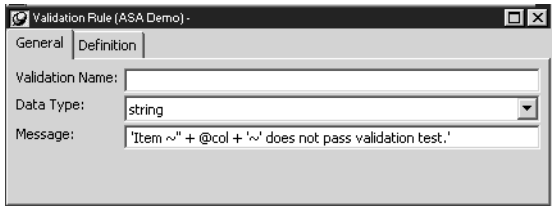
Once you place a column that has a validation rule from the extended attribute system tables in a DataWindow object, no link is maintained to the named rule in the extended attribute system tables.

If the definition of the validation rule changes in the extended attribute system tables, the rule for the column in a DataWindow object does not change.

Working with validation rules in the Database painter

This section describes the ways you can manipulate validation rules in the Database painter. You can create and modify validation rules from the Validation Rule view. This view replaces the Object Details view in the Database painter when you create a new validation rule or select Properties from the pop-up menu for an existing validation rule.

Figure 21-9: Defining a validation rule in the Database painter



What you do in the Database painter

In the Database painter, you can:

- Create, modify, and delete named validation rules

  The validation rules are stored in the extended attribute system tables. Once you define a validation rule, it can be used by any column of the appropriate datatype in the database.

- Assign validation rules to columns and remove them from columns

  These rules are used by default when you place the column in a DataWindow object in the DataWindow painter.

❖ **To create a new validation rule**

1. In the Database painter, select Object>Insert>Validation Rule from the menu bar.

   The Validation Rule view displays.

2. On the General tab, assign a name to the rule, select the datatype of the columns to which it applies, and customize the error message (if desired).

   For information, see “Customizing the error message” on page 561.

3. Click the Definition tab and define the expression for the rule.

   For information, see “Defining the expression” on page 559.

You can use this rule with any column of the appropriate datatype in the database.

❖ **To modify a validation rule:**

1. In the Database painter, open the Extended Attributes view.
2. In the Extended Attributes view, open the list of validation rules.
3. Right-click the validation rule you want to modify and select Properties from the pop-up menu.

4 In the Validation Rule view, modify the validation rule as desired.

For information, see "Defining the expression" on page 559 and "Customizing the error message" on page 561.

❖ **To associate a validation rule with a column in the Database painter:**

1 In the Objects view of the Database painter, right-click the column with which you want to associate a validation rule, select Properties from the pop-up menu, and select the Validation tab.

2 Select a validation rule from the Validation Rule drop-down list.

The column now has the selected validation rule associated with it in the extended attribute system tables. Whenever you use this column in a DataWindow object, it will use this validation rule unless you override it in the DataWindow painter.

❖ **To remove a validation rule from a column in the Database painter:**

1 In the Objects view of the Database painter, right-click the column from which you want to remove a validation rule, select Properties from its pop-up menu, and select the Validation tab in the Properties view.

2 Select `(None)` from the list in the Validation Rule drop-down list.

The validation rule is no longer associated with the column.

Defining the expression

A validation rule is a boolean expression. PocketBuilder applies the boolean expression to an entered value. If the expression returns TRUE, the value is accepted. Otherwise, the value is not accepted and an ItemError event is triggered.

What expressions can contain

You can use any valid DataWindow expression in validation rules.

Validation rules can include most DataWindow expression functions. A DataWindow object that will be used in PocketBuilder can also include user defined functions. DataWindow expression functions are displayed in the Functions list and can be pasted into the definition.

For information about these functions, see the *DataWindow Reference* in the online Help.

Use the notation *@placeholder* (where *placeholder* is any group of characters) to indicate the current column in the rule. When you define a validation rule in the Database painter, PocketBuilder stores it in the extended attribute system tables with the placeholder name. During execution, PocketBuilder substitutes the value of the column for *placeholder*.

Pasting the placeholder

The *@col* can be easily used as the placeholder. A button in the Paste area is labeled with *@col*. You can click the button to paste the *@col* into the validation rule.

An example

For example, to make sure that both Age and Salary are greater than zero using a single validation rule, define the validation rule as follows:

```
@col > 0
```

Then associate the validation rule with both the Age and Salary columns. At runtime, PocketBuilder substitutes the appropriate values for the column data when the rule is applied.

Using match values for character columns

If you are defining the validation rule for a character column, you can use the Match button on the Definition page of the Validation Rule view. This button lets you define a match pattern for matching the contents of a column to a specified text pattern (for example, ^[0-9]+$ for all numbers and ^[A-Za-z]+$ for all letters).

❖ **To specify a match pattern for character columns:**

1 Click the Match button on the Definition page of the Validation Rule view.

The Match Pattern dialog box displays.

2 Enter the text pattern you want to match the column to, or select a displayed pattern.

3 (Optional) Enter a test value and click the Test button to test the pattern.

4 Click OK when you are satisfied that the pattern is correct.

For more on the Match function and text patterns, see the *DataWindow Reference* in the online Help.

Customizing the error message

When you define a validation rule, PocketBuilder automatically creates the error message that displays by default when users enter an invalid value:

'Item ~" + *@Col* + '~' does not pass validation test.'

You can edit the string expression to create a custom error message.

Different syntax in the DataWindow painter

If you are working in the DataWindow painter, you can enter a string expression for the message, but you do not use the @ sign for placeholders. For example, this is the default message:

'Item ~" + *ColumnName* + '~' does not pass validation test.'

A validation rule for the Salary column in the Employee table might have the following custom error message associated with it:

```
'Please enter a salary greater than $10,000.'
```

If users enter a salary less than or equal to $10,000, the custom error message displays:

Figure 21-10: Custom message triggered by validation rule violation

Specifying initial values

As part of defining a validation rule, you can supply an initial value for a column.

❖ **To specify an initial value for a column in the Database painter:**

1 Select Properties from the column's pop-up menu and select the Validation tab.

2 Specify a value in the Initial Value box.

Working with validation rules in the DataWindow painter

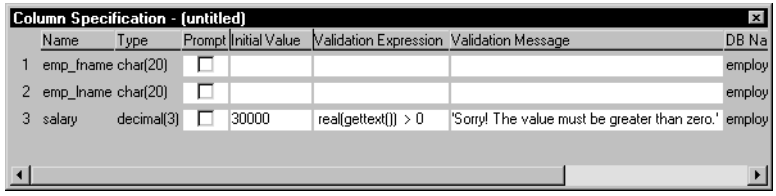
Validation rules you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the validation rule in the DataWindow painter by defining an ad hoc rule for one specific column. You define ad hoc rules in the Column Specification view.

What you do in the DataWindow painter

In the DataWindow painter, you can:

- Accept the default validation rule assigned to a column in the Database painter
- Create an ad hoc, unnamed rule to use with one specific column

Figure 21-11: Specifying a validation rule in the DataWindow painter



❖ **To specify a validation rule for a column in the DataWindow painter:**

1 In the DataWindow painter, select View>Column Specifications from the menu bar.

The Column Specification view displays.

2 Create or modify the validation expression.

To display the Modify Expression dialog box, right-click the cell under the Validation Expression column that corresponds to the column for which you are creating or modifying a validation rule, and select Expression from the pop-up menu. Follow the directions in “Specifying the expression” next.

3 (Optional) Enter a string or string expression to customize the validation error message.

To display the Modify Expression dialog box, right-click the cell under the Validation Expression column that corresponds to the column for which you are creating or modifying a validation rule, and select Expression from the pop-up menu. For more information, see “Customizing the error message” on page 561.

4 (Optional) Enter an initial value.

Used for current column only
If you create a validation rule in the DataWindow painter, it is used only for the current column and is not saved in the extended attribute system tables.

Specifying the expression

Since a user might just have entered a value in the column, validation rules refer to the current data value, which you can obtain through the GetText function.

Using GetText ensures that the most recent data entered in the current column is evaluated.

PocketBuilder does the conversion for you
If you have associated a validation rule for a column in the Database painter, PocketBuilder automatically converts the syntax to use GetText when you place the column in a DataWindow object.

GetText returns a string. Be sure to use a data conversion function (such as Integer or Real) if you want to compare the entered value with a datatype other than string.

For more on the GetText function and text patterns, see the *DataWindow Reference* in the online Help.

Referring to other columns

You can refer to the values in other columns by specifying their names in the validation rule. You can paste the column names in the rule using the Columns box.

Examples

Here are some examples of validation rules.

Example 1 To check that the data entered in the current column is a positive integer, use this validation rule:

```
Integer(GetText( )) > 0
```

Example 2 If the current column contains the discounted price and the column named Full_Price contains the full price, you could use the following validation rule to evaluate the contents of the column using the Full_Price column:

```
Match(GetText( ),"^[0-9]+$") AND
Real(GetText( )) < Full_Price
```

To pass the validation rule, the data must be all digits (must match the text pattern `^[0-9]+$`) and must be less than the amount in the Full_Price column.

Notice that to compare the numeric value in the column with the numeric value in the Full_Price column, the Real function was used to convert the text to a number.

Example 3 Suppose that in your company, a product price and a sales commission are related in the following way:

- If the price is greater than or equal to $1000, the commission is between 10 percent and 20 percent
- If the price is less than $1000, the commission is between 4 percent and 9 percent

The Sales table has two columns, Price and Commission. The validation rule for the Commission column is:

```
(Number(GetText( )) >= If(price >= 1000, .10, .04))
AND
(Number(GetText( )) <= If(price >= 1000, .20, .09))
```

A customized error message for the Commission column is:

```
"Price is " + if(price >= 1000,
"greater than or equal to","less than") +
" 1000. Commission must be between " +
If(price >= 1000,".10", ".04") + " and " +
If(price >= 1000, ".20.", ".09.")
```

How to maintain extended attributes

PocketBuilder provides facilities you can use to create, modify, and delete display formats, edit styles, and validation rules independently of their association with columns. The following procedure summarizes how you do this.

❖ **To maintain display formats, edit styles, and validation rules:**

1 Open the Database painter.

2 Select View>Extended Attributes.

The Extended Attributes view displays, listing all the entities in the extended attribute system tables.

3 Do one of the following:

- To create a new entity, display the pop-up menu for the type you want to add, then select New
- To modify an entity, display its pop-up menu, then select Properties
- To delete an entity, display its pop-up menu, then select Delete

Caution
If you delete a display format, edit style, or validation rule, it is removed from the extended attribute system tables. Columns in the database are no longer associated with the entity.

CHAPTER 22

Filtering, Sorting, and Grouping Rows

About this chapter

This chapter describes how you can customize your DataWindow object by defining filters, sorting rows, and displaying rows in groups.

Contents

Filtering rows

To limit the data that is retrieved from a database, you can use WHERE and HAVING clauses and retrieval arguments in SQL SELECT statements for a DataWindow object. This reduces retrieval time and space requirements at runtime.

However, you might want to further limit the data that displays in a DataWindow object. For example, you might want to:

- Retrieve many rows but initially display only a subset—perhaps allowing the user to specify different subsets of rows to display
- Limit the data that is displayed using DataWindow expression functions (such as If) that are not valid in the SELECT statement

Using filters

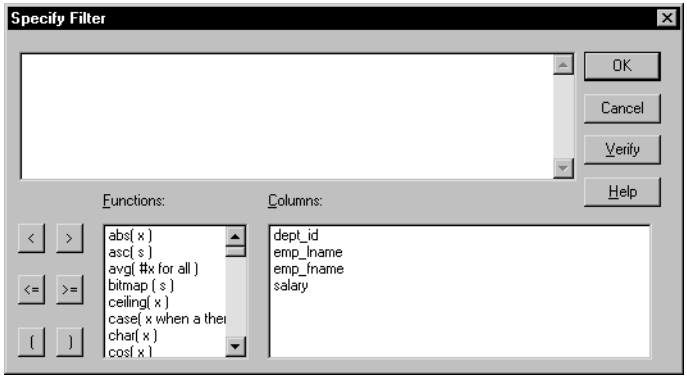
In the DataWindow painter, you can define filters, which will limit the rows that display at runtime. Filters can use most DataWindow expression functions or user-defined functions.

Filters do not affect which rows are retrieved. A filter operates against the retrieved data. It does not re-execute the SELECT statement.

Defining a filter

You define a filter in PocketBuilder in the Specify Filter dialog box.

Figure 22-1: Specifying filters for retrieved data



❖ **To define a filter:**

1. In the DataWindow painter, select Rows>Filter from the menu bar.

   The Specify Filter dialog box displays.

2. In the Specify Filter dialog box, enter a boolean expression that PocketBuilder will test against each retrieved row.

   If the expression evaluates to TRUE, the row is displayed. You can specify any valid expression in a filter. Filters can use any non-object-level PowerScript function, including user-defined functions. You can paste commonly used functions, names of columns, computed fields, retrieval arguments, and operators into the filter.

   International considerations
   For applications to run the same in any country, filter expressions require U.S. notation for numbers. That is, a comma must always represent the thousands delimiter and a period must always represent the decimal place when you specify expressions in the development environment.

   For information about expressions for filters, see the *DataWindow Reference* in the online Help.

3. (Optional) Click Verify to make sure the expression is valid.

4. Click OK.

   Only rows meeting the filter criteria are displayed in the Preview view.

Filtered rows and updates
Modifications of filtered rows are applied to the database when you issue an update request.

Removing a filter

You can remove filters in the Specify Filter dialog box.

❖ **To remove a filter:**

1 Select Rows>Filter from the menu bar.

2 Delete the filter expression from the Specify Filter dialog box, then click OK.

Examples of filters

Assume that a DataWindow object retrieves employee rows and three of the columns are Salary, Status, and Emp_Lname. Table 22-1shows some examples of filters you might use.

Table 22-1: Sample filters

To display these rows	Use this filter
Employees with salaries over $50,000	`Salary > 50000`
Active employees	`Status = 'A'`
Active employees with salaries over $50,000	`Salary > 50000 AND Status = 'A'`
Employees whose last names begin with H	`left(Emp_Lname, 1) = 'H'`

Setting filters in a script

You can use the SetFilter and Filter functions in a script to dynamically modify a filter that was set in the DataWindow painter. For information about SetFilter and Filter, see the *DataWindow Reference* in the online Help.

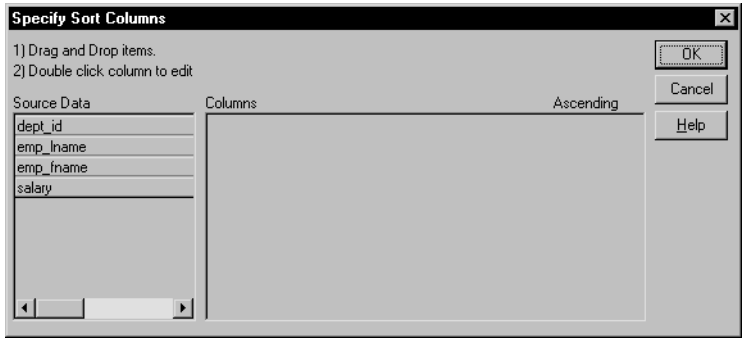
Sorting rows

You can use an ORDER BY clause in the SQL SELECT statement for the DataWindow object to sort the data that is retrieved from the database. If you do this, the DBMS itself does the sorting, and the rows are brought into PocketBuilder already sorted.

However, you might want to sort the rows after they are retrieved. For example, you might want to:

- Offload the processing from the DBMS
- Sort on an expression, which is not allowed in the SELECT statement but is allowed in PocketBuilder

Figure 22-2: Selecting rows for sorting retrieved data



❖ **To sort the rows:**

1 Select Rows>Sort from the menu bar.

2 Drag the columns that you want to sort the rows on to the Columns box, and specify whether you want to sort in ascending or descending order.

The order of the columns determines the precedence of the sort. To reorder the columns, drag them up or down in the list. To delete a column from the sort columns list, drag the column outside the dialog box.

3 You can also specify expressions to sort on: for example, if you have two columns, Revenues and Expenses, you can sort on the expression *Revenues – Expenses*.

To specify an expression to sort on, double-click a column name in the Columns box, modify the expression in the Modify Expression dialog box, and click OK. You return to the Specify Sort Columns dialog box with the expression displayed.

You can remove a column or expression from the sorting specification by simply dragging it and releasing it outside the Columns box.

4 Click OK when you have specified all the sort columns and expressions.

Suppressing repeating values

When you sort on a column, there might be several rows with the same value in one column. You can choose to suppress the repeating values in that column. When you suppress a repeating value, the value displays at the start of each new page and, if you are using groups, each time a value changes in a higher group.

You can select columns for which you want to suppress repeating values in the Specify Repeating Value Suppression List dialog box.

Figure 22-3: Selecting columns for suppression of repeated values

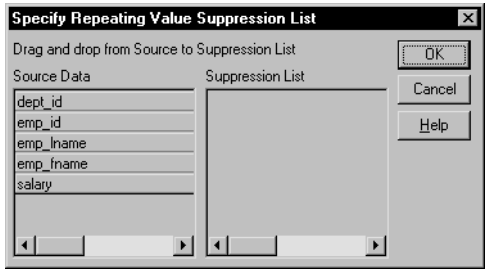
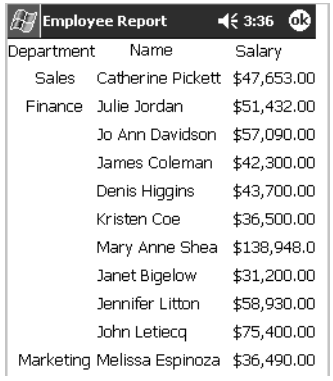


Figure 22-4 shows a DataWindow with a list of employees sorted by department ID. In this figure, all but the first occurrence of each department ID has been filtered out.

Figure 22-4: Filtering out repeat values for the dept_name column



Department	Name	Salary
Sales	Catherine Pickett	$47,653.00
Finance	Julie Jordan	$51,432.00
	Jo Ann Davidson	$57,090.00
	James Coleman	$42,300.00
	Denis Higgins	$43,700.00
	Kristen Coe	$36,500.00
	Mary Anne Shea	$138,948.0
	Janet Bigelow	$31,200.00
	Jennifer Litton	$58,930.00
	John Letiecq	$75,400.00
Marketing	Melissa Espinoza	$36,490.00

❖ **To suppress repeating values:**

1 Select Rows>Suppress Repeating Values from the menu bar.

The Specify Repeating Value Suppression List dialog box displays.

2 Drag the columns whose repeated values you want to suppress from the Source Data box to the Suppression List box and click OK.

You can remove a column from the suppression list by simply dragging it and releasing it outside the Suppression List box.

Grouping rows

You can group related rows together and, optionally, calculate statistics for each group separately. For example, you might want to group employee information by department and get total salaries for each department.

How groups are defined

Each group is defined by one or more DataWindow object columns. Each time the value in a grouping column changes, a break occurs and a new section begins.

For each group you can:

- Display the rows in each section
- Specify the information you want displayed at the beginning and end of each section
- Specify page breaks after each break in the data
- Reset the page number after each break

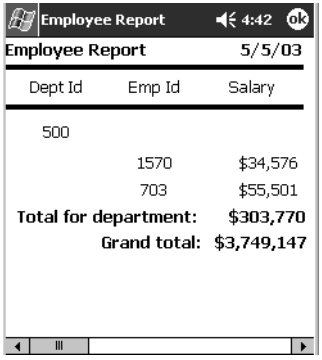
Grouping example

The following DataWindow object retrieves employee information. It has one group defined, Dept_ID, so it groups rows into sections according to the value in the Dept_ID column. In addition, it displays:

- Department ID before the first row for that department
- Totals and averages for salary and salary plus benefits (a computed column) for each department
- Grand totals for the company at the end

Figure 22-5 shows the DataWindow object. Grouping the data has the effect of suppressing repeated values for the column or columns on which the grouping is based.

Figure 22-5: Last page of report for employees grouped by department



How to do it

You can create a grouped DataWindow object in two ways:

- Use the Group presentation style to create a grouped DataWindow object from scratch.
- Take an existing tabular DataWindow object and define grouping (“Defining groups in an existing DataWindow object” on page 577).

Making the DataWindow control large enough

If a DataWindow object has grouped rows, each page contains all group headers (including zero-height headers) at the top of the page.

The last row of a group displays on the same page as that row's group trailer and each applicable higher level group trailer. If the DataWindow object has a summary band, it displays on the same page as the last row of the report. If the control is not large enough, you might see anomalies when scrolling through the DataWindow object. This is particularly likely in the last row of the report, which needs room to display the report's header band, all group headers, all group trailers, the summary band, and the footer band.

If you cannot increase the height of the DataWindow control so that it has room for all the headers and trailers, you can change the design of the DataWindow object so that they require less space.

Scrolling through a grouped DataWindow

When you scroll through a grouped DataWindow object, you might see the group header repeated where you do not expect it. This is because the data is paginated in a fixed layout based on the size of the DataWindow control. You can scroll to a point that shows the bottom half of one page and the top of the next.

When you use the arrow keys to page through the data, you see the data one page at a time, but you (or your application users) might have to press the arrow keys several times before the next page is displayed. The SIP keyboard or a peripheral keyboard can be used to scroll through DataWindow pages deployed to a PDA device.

Using the Group presentation style

One of the DataWindow object presentation styles, Group, is a shortcut to creating a grouped DataWindow object. It generates a tabular DataWindow object that has one group level and some other grouping properties defined. You can then further customize the DataWindow object.

You specify columns on which to group a DataWindow in the Set Report Definition page of the Group DataWindow wizard, or in the Specify Group Columns dialog box that you open from the DataWindow painter.

Figure 22-6 shows selections for grouping by department, as specified by the dept_id column.

Figure 22-6: Selecting a column on which to group a report

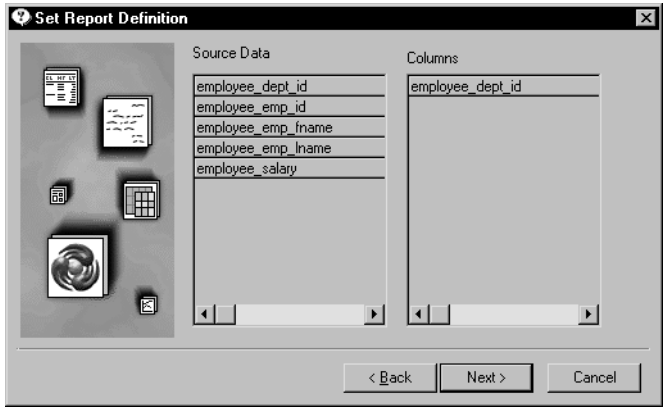
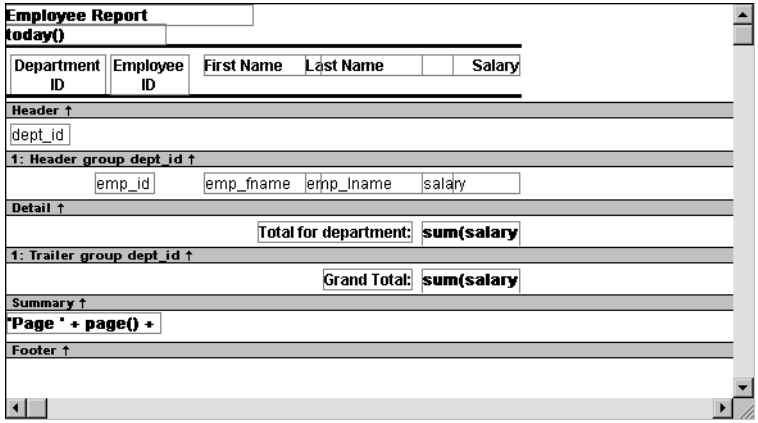


Figure 22-7 is an example of a Group style DataWindow object in the Design view of the DataWindow painter.

Figure 22-7: Design view of a DataWindow with a Group presentation style



❖ **To create a basic grouped DataWindow object using the Group presentation style:**

1 Select File>New from the menu bar.

The New dialog box displays.

2 Choose the DataWindow tab page and the Group presentation style, and click OK.

3 Choose a data source and define the data.

You are prompted to define the grouping column(s).

4 Drag the column(s) you want to group from the Source Data box to the Columns box.

Multiple columns and multiple group levels
You can specify more than one column, but all columns apply to group level one. You can define one group level at this point, and define additional group levels later.

If you want to use an expression
If you want to use an expression, you can enter it as the Group Definition on the General page in the Properties view after you have finished using the Group Wizard. You can specify more than one grouping item expression for a group. A break occurs whenever the value concatenated from each column/expression changes.

5 Click Next.

PocketBuilder suggests a header based on your data source. For example, if your data comes from the Employee table, PocketBuilder uses the name Employee in the suggested header.

6 Specify the Page Header text, specify page break and page numbering properties for data groups, and click Next.

If you want a page break each time a grouping value changes, select the New Page On Group Break box.

If you want page numbering to restart at 1 each time a grouping value changes, select the Reset Page Number On Group Break box *and* the New Page On Group Break box.

7 Select Color and Border settings and click Next.

8 Review your specification and click Finish.

The DataWindow object displays with the basic grouping properties set.

What PocketBuilder does

As a result of your specifications, PocketBuilder generates a tabular DataWindow object and:

- Creates group header and trailer bands
- Places the column you chose as the grouping column in the group header band
- Sorts the rows by the grouping column
- Places the page header and the date (as a computed field) in the header band
- Places the page number and page count (as computed fields) in the footer band
- Creates sum computed fields for all numeric columns (the fields are placed in the group trailer and summary bands)

What you can do

You can use any of the techniques available in a tabular DataWindow object to modify and enhance the grouped DataWindow object, such as moving controls, specifying display formats, and so on.

For information about the bands in a grouped DataWindow object and how to add features especially suited for grouped DataWindow objects (such as adding a second group level, defining additional summary statistics, and so on), see "Defining groups in an existing DataWindow object" next.

DataWindow object is not updatable by default
When you generate a DataWindow object using the Group presentation style, PocketBuilder makes it not updatable by default. If you want to be able to update the database through the grouped DataWindow object, you must modify its update characteristics. For more information, see Chapter 20, "Controlling Updates in DataWindow Objects."

Defining groups in an existing DataWindow object

Instead of using the Group presentation style to create a grouped DataWindow object from scratch, you can take an existing tabular DataWindow object and define groups in it.

The following procedure is an overview of how you group data in an existing tabular DataWindow object. Steps 2 through 6 are described in more detail in separate procedures.

❖ **To add grouping to an existing DataWindow object:**

1 Start with a tabular DataWindow object that retrieves all the columns you need.

2 Specify the grouping columns.

3 Sort the rows.

4 (Optional) Rearrange the DataWindow object.

5 (Optional) Add summary statistics.

6 (Optional) Sort the groups.

Specify the grouping columns

❖ **To specify the grouping columns:**

1 In the DataWindow painter, Select Rows>Create Group from the menu bar.

The Specify Group Columns dialog box displays.

2 Specify the group columns, as described in “Using the Group presentation style” on page 574.

3 Set the Reset Page Count and New Page on Group Break properties on the General page in the Properties view.

Creating subgroups

After defining your first group, you can define subgroups, which are groups within the group you just defined.

❖ **To define subgroups:**

1 Select Rows>Create Group from the menu bar and specify the column/expression for the subgroup.

2 Repeat step 1 to define additional subgroups if you want.

You can specify as many levels of grouping as you need.

How groups are identified

PocketBuilder assigns each group a number (or level) when you create the group. The first group you specify becomes group 1, the primary group. The second group becomes group 2, a subgroup within group 1, and so on.

For example, say you defined two groups. The first group uses the dept_id column and the second group uses the status column. The rows will be grouped first by department (group 1). Within department, rows will be grouped by status (group 2). If you specify page breaks for the groups, a page break will occur when any of these values changes.

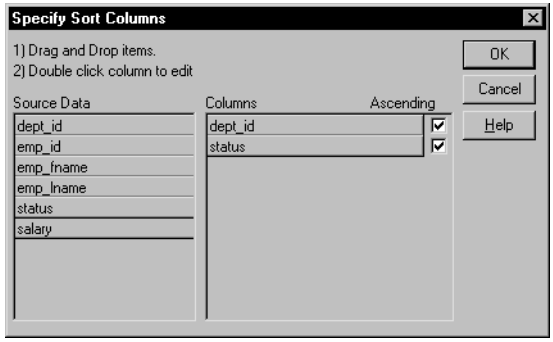
You use the group's number to identify it when defining summary statistics for the group. This is described in “Add summary statistics” on page 580.

Sort the rows

PocketBuilder does not sort the data when it creates a group. Therefore, if the data source is not sorted, you must sort the data by the same columns (or expressions) specified for the groups.

For example, if you are grouping by dept_id then status, select Rows>Sort from the menu bar and specify dept_id and then status as sorting columns.

Figure 22-8: Sorting data by columns used to group data



You can also sort on additional rows. For example, if you want to sort by employee ID within each group, specify emp_id as the third sorting column.

For more information about sorting, see "Sorting rows" on page 569.

Rearrange the DataWindow object

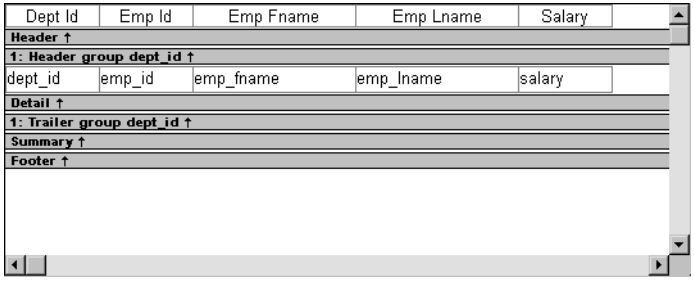
When you create a group, PocketBuilder creates two new bands for each group:

- A group header band
- A group trailer band

The bar identifying the band contains:

- The number of the group
- The name of the band
- The name of each column that defines the group
- An arrow pointing to the band

Figure 22-9: Group header and trailer bands in the Design view



You can include any control in the DataWindow object (such as columns, text, and computed fields) in the header and trailer bands of a group.

Using the group header band

The contents of the group header band display at the top of each page and after each break in the data. Typically, you use this band to identify each group. You might move the grouping column from the detail band to the group header band, since it now serves to identify one group rather than each row.

For example, if you group the rows by department and include the department in the group header, the department will display before the first line of data each time the department changes. For an example of how this might look at runtime, see Figure 22-5 on page 573.

Using the group trailer band

The contents of the group trailer display after the last row for each value that causes a break.

In the group trailer band, you specify the information you want displayed after the last line of identical data for each value in the group. Typically, you include summary statistics here, as described next.

Add summary statistics

One of the advantages of creating a grouped DataWindow object is that you can have PocketBuilder calculate statistics for each group. To do that, you place computed fields that reference the group. Typically, you place these computed fields in the group's trailer band.

❖ **To add a summary statistic:**

1 Select Insert>Control>Computed Field from the menu bar.

2 Click in the Design view where you want the statistic.

The Modify Expression dialog box displays.

3 Specify the expression that defines the computed field (see “Specifying the expression” next).

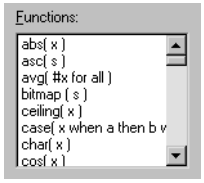
4 Click OK.

A shortcut to sum values
If you want to sum a numeric column, select the column in Design view and click the Sum button in the Controls drop-down toolbar. PocketBuilder automatically places a computed field in the appropriate band.

Specifying the expression

Typically, you use aggregate and other functions in your summary statistic. PocketBuilder lists functions you can use in the Functions box in the Modify Expression dialog box. When you are defining a computed field in a group header or trailer band, PocketBuilder automatically lists forms of the functions that reference the group.

Figure 22-10: Listing of functions in the Modify Expression dialog box



You can paste a function into the expression, then replace the placeholder (such as *#x*) that is pasted in as a function argument with the appropriate column or expression.

For example, to count the employees in each department (group 1), specify this expression in the group trailer band:

```
Count( Emp_Id for group 1 )
```

To get the average salary of employees in a department, specify:

```
Avg( Salary for group 1 )
```

To get the total salary of employees in a department, specify:

```
Sum( Salary for group 1 )
```

Figure 22-11: Design view with group functions in trailer band

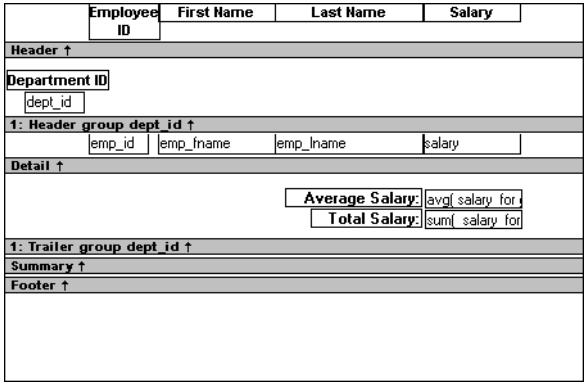
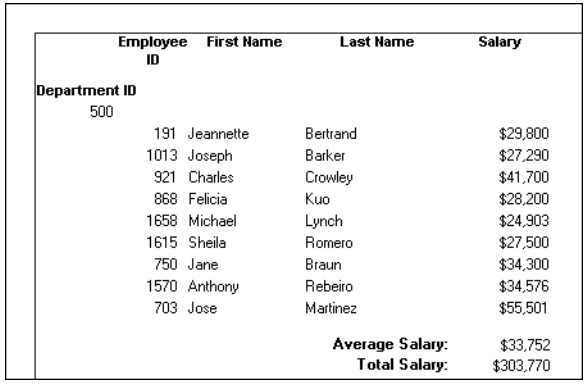


Figure 22-12 shows the same DataWindow as it appears in the Print Preview view.

Figure 22-12: Print Preview view with group functions in trailer band



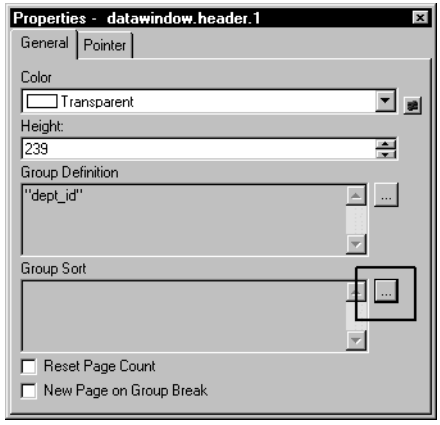
Employee ID	First Name	Last Name	Salary
Department ID			
500			
191	Jeannette	Bertrand	$29,800
1013	Joseph	Barker	$27,290
921	Charles	Crowley	$41,700
868	Felicia	Kuo	$28,200
1658	Michael	Lynch	$24,903
1615	Sheila	Romero	$27,500
750	Jane	Braun	$34,300
1570	Anthony	Rebeiro	$34,576
703	Jose	Martinez	$55,501
		Average Salary:	$33,752
		Total Salary:	$303,770

Sort the groups

You can sort the groups in a DataWindow object. For example, in a DataWindow object showing employee information grouped by department, you might want to sort the departments (the groups) by total salary.

Typically, this involves aggregate functions, as described in “Add summary statistics” on page 580. In the department salary example, you could sort the groups using the aggregate function Sum to calculate total salary in each department.

Figure 22-13: Group Sort button in the Properties view for a group header



❖ **To sort the groups:**

1 Click the group header bar in the Design view of the DataWindow painter.

The mouse pointer displays as a double-headed arrow when it is positioned in the group header bar.

After you click the group header bar, the General properties page for the group displays in the Properties view.

2 Click the ellipsis button next to the Group Sort box.

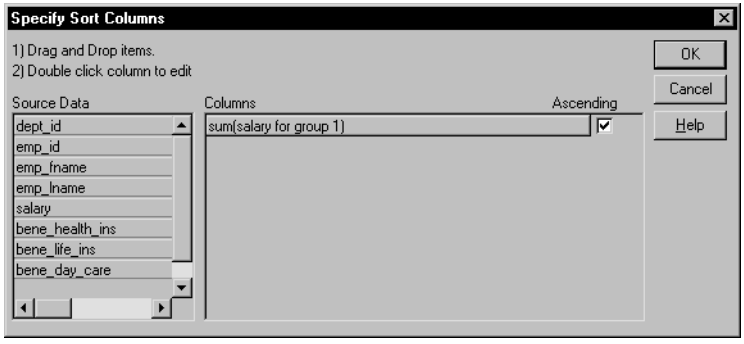
The Specify Sort Columns dialog box displays.

3 Drag the column by which you want to sort the groups from the Source Data box into the Columns box.

If you chose a numeric column, PocketBuilder uses the Sum function in the expression; if you chose a non-numeric column, PocketBuilder uses the Count function.

For example, if you chose the Salary column, PocketBuilder specifies that the groups will be sorted by the expression `sum(salary for group 1)`:

Figure 22-14: Sorting a group based on the sum of group salaries



4 Select ascending or descending sort as appropriate.

5 If you want to modify the expression to sort on, double-click the column in the Columns box.

The Modify Expression dialog box displays.

6 Specify the expression to sort on.

For example, to sort the department group (the first group level) on average salary, specify `avg(salary for group 1)`.

7 Click OK.

You return to the Specify Sort Columns dialog box with the expression displayed.

8 Click OK again.

At runtime, the groups will be sorted on the expression you specified.

CHAPTER 23

Highlighting Information in DataWindow Objects

About this chapter

This chapter describes how you modify the way information displays in DataWindow objects and reports by specifying various conditions. The conditions are usually related to data values, which are not available until runtime.

Contents

Highlighting information

Every control in a DataWindow object has a set of properties that determines what the control looks like and where it is located. For example, the values in a column of data display in a particular font and color, in a particular location, with or without a border, and so on.

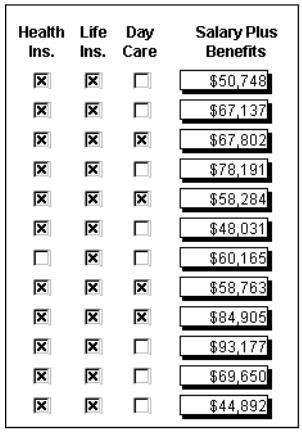
Modifying properties when designing

You define the appearance and behavior of controls in DataWindow objects in the DataWindow painter. By doing that, you specify the controls' properties. For example, by placing a border around a column, you are setting that column's Border property.

In most cases, the appearance and behavior of controls is fixed; you do not want them to change at runtime. When you make headings bold, you generally want them to be bold at all times.

In the following DataWindow object, the Salary Plus Benefits column has a Shadow box border around every data value in the column. To display the border, you set the border property for the column.

Figure 23-1: Shadow box border around a computed column



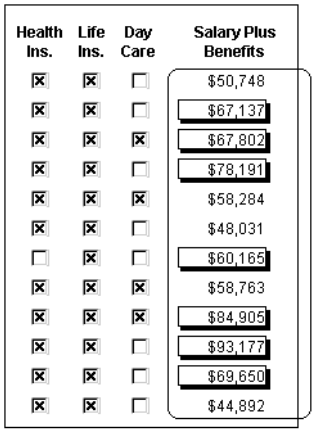
Modifying properties at runtime

In some applications, you might want to make certain properties of controls in DataWindow objects dependent on the data, which is not known to you when you define the DataWindow object in the painter. For these situations you can define property conditional expressions, which are expressions that are evaluated at runtime.

You can use these expressions to modify the appearance and behavior of DataWindow objects conditionally and dynamically during execution. When the conditions of the expressions have been met, the results of the expressions change the values of properties of controls in the DataWindow objects.

For example, in the following DataWindow object, the Salary Plus Benefits column has a Shadow box border highlighting each data value that is greater than $60,000.

Figure 23-2: Conditional implementation of shadow box highlighting



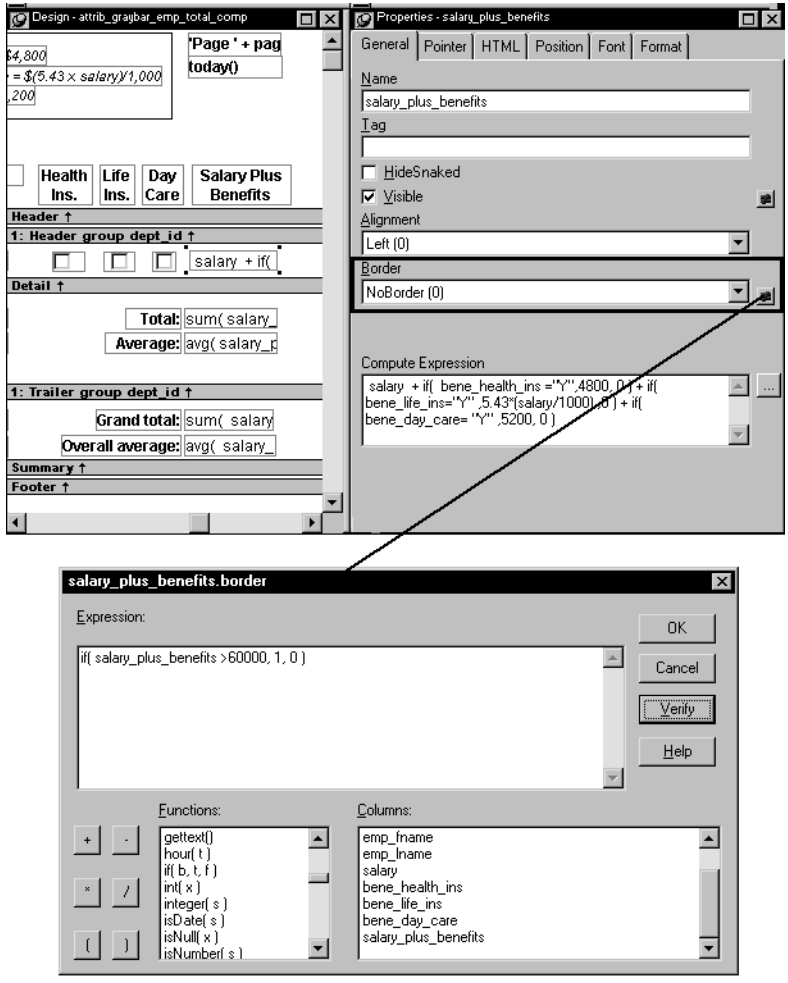
To control the display of the border, you define a conditional expression for the column's Border property. When users run the DataWindow object, PocketBuilder changes the border of individual data values based on the condition (value greater than $60,000).

Defining an expression

Figure 23-3 shows the Salary_Plus_Benefits column selected in the Design view. To the right of the Design view, the Properties view shows properties for the column, including the Border property. Next to the Border property is a button for accessing the dialog box where you enter the expression. The button displays a red equals sign with a slash through it when no expression has been entered, and a green equals sign with no slash when it has.

You click the button to open the Border expression dialog box. A line in the figure connects the button to the dialog box. In this example, although the Border property is set to NoBorder in the Properties view, the expression defined for the property overrides the NoBorder setting at runtime.

Figure 23-3: Setting a conditional expression for the Border property



A closer look at the expression

The expression you enter almost always begins with `If`. Then you specify three things: the condition, what happens if it is true, and what happens if it is false. Parentheses surround the three things and commas separate them:

If(*expression, true, false*)

The following expression is used in Figure 23-3. Because the expression is for the Border property, the values for true and false indicate particular borders. The value 1 means Shadow box border and the value 0 means no border:

```
If(salary_plus_benefits > 60000, 1, 0)
```

When users run the DataWindow object, PocketBuilder checks the value in the computed column called salary_plus_benefits to see if it is greater than 60,000. If it is (true), PocketBuilder displays the value with the Shadow box border. If not (false), PocketBuilder displays the value with no border.

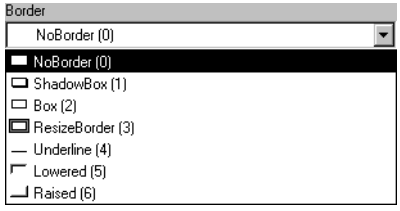
About specifying properties

Usually you specify a number to indicate what to assign to a particular property. For example, the following list shows all of the borders you can specify and the numbers you use. If you want the border property to be Shadow box for either true or false, you specify `1` in the appropriate position in the If statement.

`0`—None
`1`—Shadow box
`2`—Box
`3`—Resize (not supported on the Windows CE platform)
`4`—Underline
`5`—3D Lowered
`6`—3D Raised

In the Properties view, the list of choices for setting a property includes the values that correspond to choices in parentheses. This makes it easier to define an expression for a property; you do not need to look up the values. For example, if you want to specify the 3D raised border in your expression, you use the number 6, as shown in the drop-down list.

Figure 23-4: Drop-down list for Border property values



For details on the values of properties you can set using expressions, see “Supplying property values” on page 595.

For information about properties associated with a DataWindow object, see the discussion of DataWindow object properties in the online Help.

Modifying properties programmatically

You can programmatically modify the properties of controls in a DataWindow object during execution. For more information, see the *DataWindow Reference* in the online Help.

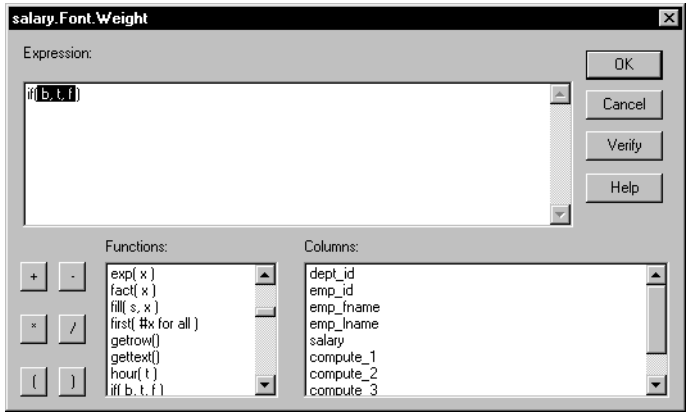
Modifying properties conditionally at runtime

"Modifying properties at runtime" on page 586 describes how you can use conditional expressions that are evaluated at runtime to highlight information in a DataWindow object. This section presents a procedure for modifying properties at runtime and includes some examples.

❖ **To modify properties conditionally at runtime:**

1. Position the pointer on the control, band, or DataWindow object background whose properties you want to modify during execution.
2. Select Properties from the pop-up menu, then select the page that contains the property you want to modify at runtime.
3. Click the button next to the property you want to change.
4. Scroll the list of functions in the Functions box until you see the IF function, and then select it.

Figure 23-5: Adding a conditional expression for font weight



5 Replace the *b* (boolean) argument of the IF function with your condition.

You can select columns and functions and use the buttons to add the symbols shown on them. As an example expression, you can use `salary>40000`.

6 Replace the *t* (true) argument with the value to use for the property when the condition is true.

Values for properties are usually numbers. They are different for each property. For more information about property values you can set on the Expressions page, see "Supplying property values" on page 595.

For information about valid values for properties of controls in the DataWindow object, see the discussion of DataWindow object properties in the online Help.

7 Replace the *f* (false) argument with the value to use for the property when the condition is false.

8 Click OK.

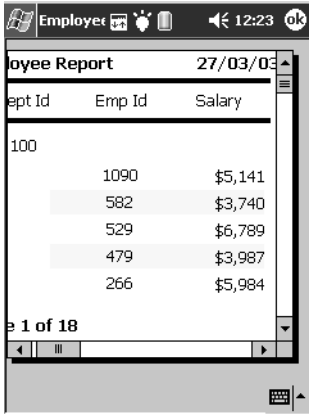
The following sections show examples of data and label highlighting and the use of conditional properties:

Example 1: creating a gray bar effect
Example 2: rotating controls
Example 3: highlighting rows of data

Example 1: creating a gray bar effect

The following DataWindow object shows alternate rows with a light gray bar. The gray bars make it easier to track data values across the row.

Figure 23-6: Displaying alternate rows in gray bars for data highlighting



To create the gray bar effect:

1 Add a rectangle control to the detail band and size it so that it surrounds the controls you want highlighted.

   To make sure that you have selected the detail band, select the Position tab in the Properties view and select Band from the Layer drop-down list.

2 To make it easier to see what you are doing in the Design view, select the General tab and set the Brush Color to White and the Pen Color to Black. A narrow black line bounds the rectangle.

3 Select Send to Back from the rectangle's pop-up menu.

4 To hide the border of the rectangle, set the Pen Style to No Visible Line.

5 Click the button next to the Brush Color property on the General page.

6 In the Modify Expression dialog box, enter the following expression for the Brush.Color property:

```
If(mod(getrow(),2 )=1, rgb(255, 255, 255 ), rgb(240,
240, 240 ))
```

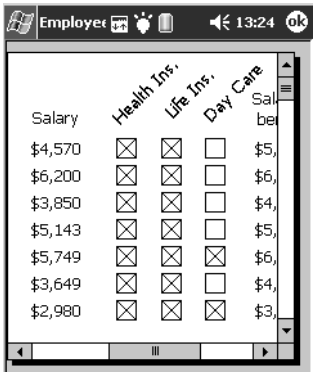
   The mod function takes the row number (`getrow()`), divides it by `2`, then returns the remainder. The remainder can be either 0 or 1. If the row number is odd, mod returns 1; if the row number is even, mod returns 0. The boolean expression `mod(getrow(),2)=1` distinguishes odd rows from even rows.

The rgb function specifies maximum amounts of red, green, and blue: `rgb (255, 255, 255)`. Specifying 255 for red, green, and blue results in the color white. If the row number is odd (the condition evaluates as true), the rectangle displays as white. If the row number is even (the condition evaluates as false), the rectangle displays as light gray (`rgb (240, 240, 240)`).

Example 2: rotating controls

The following DataWindow object shows the column headers for Health Insurance, Life Insurance, and Day Care rotated 45 degrees.

Figure 23-7: DataWindow with rotated text highlighting



To rotate each of these three text controls:

1 Select one of the controls, then use Ctrl + click to select the other two controls.

   The Properties view changes to show the properties that are common to all selected controls.

2 On the Font page in the Properties view, click the button next to the Escapement property.

3 Enter the number 450 in the Modify Expression dialog box and click OK.

   The value entered for font escapement is in tenths of degrees, so the number 450 means 45 degrees. Specifying a condition is optional. Typically, you do not specify a condition for control rotation.

   The rotation of the controls does not change in the Design view.

4 To see the change, close and reopen the Preview view.

Example 3: highlighting rows of data

The following DataWindow object is an employee phone list for a company in the U.S. state of Massachusetts. Out-of-state (not in Massachusetts) employees are shown in bold and preceded by two asterisks (**):

Figure 23-8: Data highlighting with asterisks

*(** means out of state)*

Employee Phone List

08/20/02

Employee Name	Phone	Employee Name	Phone
Ahmed, Alex	(617) 555-8748	Pickett, Catherine	(617) 555-3478
Barker, Joseph	(617) 555-8021	Poitras, Kathleen	(617) 555-3920
Barletta, Irene	(617) 555-8345	Powell, Thomas	(617) 555-1956
Bertrand, Jeannette	(508) 555-8138	Preston, Mark	(617) 555-5862
Bigelow, Janet	(617) 555-1493	Rabkin, Andrew	(617) 555-4444
Blaikie, Barbara	(617) 555-9345	Rebeiro, Anthony	(617) 555-5737
Braun, Jane	(617) 555-7857	Romero, Sheila	(617) 555-8138
Breault, Robert	(617) 555-3099	Samuels, Peter	(617) 555-8342
Bucceri, Matthew	(617) 555-5336	** **Savarino, Pamela**	**(310) 555-1857**
Butterfield, Joyce	(617) 555-2232	Scott, David	(617) 555-3246
Chao, Shih Lin	(617) 555-5921	Shea, Mary Anne	(617) 555-4616
Charlton, Doug	(508) 555-9246	** **Sheffield, John**	**(713) 555-3877**
** **Chin, Philip**	**(404) 555-2341**	Shishov, Natasha	(617) 555-2755

In the Design view, the detail band includes four controls: the employee last name, a comma, the employee first name, and the phone number. (A computed field that concatenates the last name and first name columns, and separates them with a comma, could be used instead of the first three controls.)

Logic that relies on the state column

In this example, you use logic that relies on the state column, so you need to include it in the data source. You can add the column after creating the DataWindow object by modifying the data source. Notice that the state column does not actually appear anywhere in the DataWindow object in Figure 23-8. Values must be available but do not need to be displayed.

To make these controls display in bold with two asterisks if the employee is not from Massachusetts:

1 Select one of the controls, then use Ctrl + Click to select the other controls.

   The Properties view changes to show the properties that are common to all selected controls.

2 On the Font page in the Properties view, click the button next to the Bold property.

3 Enter the following expression in the Modify Expression dialog box and click OK:

```
If(state = 'MA', 400, 700)
```

The expression states that if the value of the state column is MA, use 400 as the font weight. This means employees from Massachusetts display in the normal font. For any state except MA, use 700 as the font weight. This means all other employees display in bold font.

4 To insert two asterisks (**) in front of the employee name if the employee is not from Massachusetts, add a text control to the left of the employee name with the two asterisks in bold.

5 With the text control selected, click the button next to its Visible property on the General page in the Properties view.

6 In the Modify Expression dialog box that displays, enter the following expression and click OK:

```
If(state = 'MA', 0, 1)
```

This expression says that if the state of the employee is MA (the true condition), the Visible property of the ** control is off (indicated by 0). If the state of the employee is not MA (the false condition), the Visible property of the ** control is on (indicated by 1). The asterisks are visible next to that employee's name.

Other highlighting tips
You can use underlines, italics, strikethrough, borders, and colors to highlight information.

Supplying property values

Each property has its own set of property values that you can use to specify the true and false conditions in the If expression. Usually you specify a number to indicate what you want. For example, if you are working with the Border property, you use the number 0, 1, 2, 4, 5, or 6 to specify a border. (The number 3 specifies the Resize border that does not display correctly on the Windows CE platform.)

Valid values display in the Properties view wherever possible.
For example, the drop-down list showing border selections includes the correct number for specifying each border in code. In this case, the number is shown parenthetically after the name of the border type, such as ShadowBox (1), Box (2), and so on.

Table 23-1 summarizes the properties available for controls in a DataWindow. A detailed description of each property follows the table. For more information about control properties, see the *DataWindow Reference* in the online Help.

Table 23-1: Properties for controls in the DataWindow painter

Property	Painter option in Properties view	Description
Background.Color	Background Color on General page or Font page	Background color of a control
Border	Border on General page	Border of a control
Brush.Color	Brush Color on General page	Color of a graphic control
Brush.Hatch	Brush Hatch on General page	Pattern used to fill a graphic control
Color	Text Color on Font page; Color on General page; Line Color on General page	Color of text for text controls, columns, and computed fields; background color for the DataWindow object; line color for graphs
Font.Escapement (for rotating controls)	Escapement on Font page	Rotation of a control
Font.Height	Size on Font page	Height of text
Font.Italic	Italic on Font page	Use of italic font for text
Font.Strikethrough	Strikeout on Font page	Use of strikethrough for text
Font.Underline	Underline on Font page	Use of underlining for text
Font.Weight	Bold on Font page	Weight (for example, bold) of text font
Format	Format on Format page	Display format for columns and computed fields
Height	Height on Position page	Height of a control
Pen.Color	Pen Color on General page	Color of a line or the line surrounding a graphic control
Pen.Style	Pen Style on General page	Style of a line or the line surrounding a graphic control
Pen.Width	Pen Width on General page	Width of a line or the line surrounding a graphic control
Protect	Protect on General page	Whether a column can be edited
Timer_Interval	Timer Interval on General page	How often time fields are to be updated
Visible	Visible on General page	Whether a control is visible
Width	Width on Position page	Width of a control
X	X on Position page	X position of a control
X1, X2	X1, X2 on Position page	X coordinates of both ends of a line

Property	Painter option in Properties view	Description
Y	Y on Position page	Y position of a control relative to the band in which it is located
Y1, Y2	Y1, Y2 on Position page	Y coordinates of both ends of a line

Background.Color

Description

Setting for the background color of a control.

In the painter

Background Color on the General page or Font page in the Properties view.

Value

A number that specifies the control's background color. For more information, see “Specifying colors” on page 610.

The background color of a line is the color that displays between the segments of the line when the pen style is not solid.

If Background.Mode is transparent (1), Background.Color is ignored.

Example

The following statement specifies that if the person represented by the current row uses the day care benefit, the background color of the control is set to light gray (15790320). If not, the background color is set to white (16777215):

```
If(bene_day_care = 'Y', 15790320, 16777215)
```

In this example, the condition is applied to the Background.Color property for three controls: the emp_id column, the emp_lname column, and the salary column.

Figure 23-9 shows the Design view for a tabular DataWindow where the employee ID, last name, and salary have a gray background if the employee uses the day care benefit.

Figure 23-9: DataWindow columns with a conditional background color

Border

Description

The type of border for the control.

In the painter

Border on the General page in the Properties view.

Value

A number that specifies the type of border. Values are:

0—None
1—Shadow box
2—Box
3—Resize (not supported on the Windows CE platform)
4—Underline
5—3D Lowered
6—3D Raised

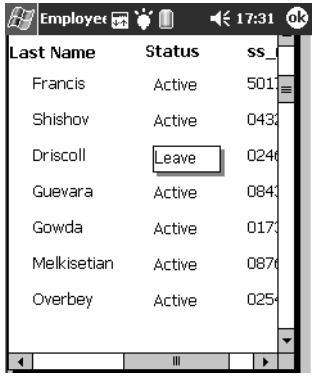
Example

The following statement specifies that if the person represented by the current row has a status of L (on leave), the status column displays with a Shadow box border:

```
If(status = 'L', 1, 0)
```

In this example, the condition is applied to the Border property of the status column. Figure 23-10 is a portion of the resulting DataWindow object. Notice that the Leave status displays with a Shadow box border.

Figure 23-10: DataWindow with a conditional Shadow box border



About the value L and the value On Leave
The status column uses an edit style. The internal value for on leave is L and the display value is Leave. The conditional expression references the internal value L, which is the actual value stored in the database. The DataWindow object shows the value Leave, which is the display value assigned to the value L in the code table for the Status edit style.

Brush.Color

Description — Setting for the fill color of a graphic control.

In the painter — Brush Color on the General page in the Properties view.

Value — A number that specifies the color that fills the control.

For information on specifying colors, see "Specifying colors" on page 610.

Example — See the example for "Brush.Hatch" next.

Brush.Hatch

Description — Setting for the fill pattern of a graphic control.

In the painter — Brush Hatch on the General page in the Properties view.

Value — A number that specifies the pattern that fills the control. The only values that are supported on the Windows CE platform are:

- 6—Solid
- 7—Transparent

Color

Description — The color of text for text controls, columns, and computed fields; background color for the DataWindow object; line color for graphs.

In the painter — In the Properties view, Text Color on the Font property page; Color on the General property page; Line Color on the General property page.

Value — A number that specifies the color used for text.

For information on specifying colors, see "Specifying colors" on page 610.

Example — The following statement is for the Color property of the emp_id, emp_fname, emp_lname, and emp_birth_date columns:

```
If(month(birth_date) = month (today()), 255, 0)
```

If the employee has a birthday in the current month, the information for the employee displays in red (255). Otherwise, the information displays in black (0).

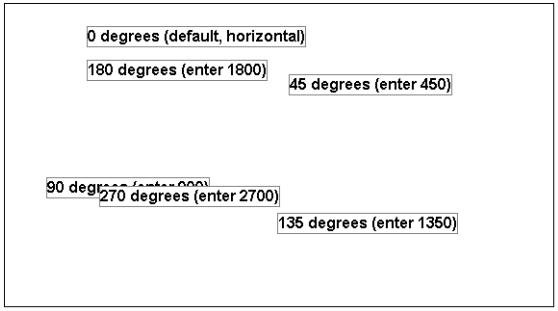
The Font.Underline property has the same conditional expression defined for it so that the example shows clearly on paper when printed in black and white.

Font.Escapement (for rotating controls)

Description	The angle of rotation from the baseline of the text.
In the painter	Escapement on the Font page in the Properties view.
Value	An integer in tenths of degrees. For example, 450 means 45 degrees. 0 is horizontal.
Example	To enter rotation for a control, select the control in the Design view and click the button next to the Escapement property in the Properties view. In the dialog box that displays, enter the number of tenths of degrees.

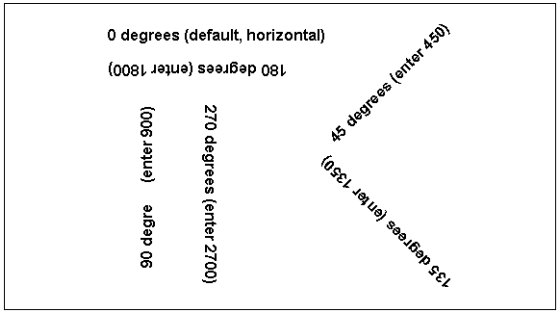
The following picture shows the Design view with a number of text controls. Each text control shows the Font.Escapement value entered and the number of degrees of rotation. In the Design view, you do not see rotation; it looks as if the controls are all mixed up. One control seems to overlay another.

Figure 23-11: Design view with rotated text controls



The next picture shows the same controls in the Preview view after it has been closed and reopened. Each control is rotated appropriately.

Figure 23-12: Preview view with rotated text controls



Viewing rotated controls in the Preview view
If the Preview view is open in the DataWindow painter when you set the Font.Escapement property, you must close and reopen it in order to see the rotated text.

Font.Height

Description
The height of the text.

In the painter
Size on the Font page in the Properties view.

Value
An integer in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels. To specify size in points, specify a negative number.

Example
The following statement is specified for the Font.Height property of a text control. Note that the DataWindow object is defined as using thousandths of an inch as its unit of measure. The statement says that if the control is in the first row, show the text 1/2-inch high (500 1/1000ths of an inch), and if it is not the first, show the text 1/5-inch high (200 1/1000ths of an inch):

```
If(GetRow() = 1, 500, 200)
```

The boundaries of the control might need to be extended to allow for the increased size of the text. At runtime, the first occurrence of the text control is big (1/2 inch); subsequent ones are small (1/5 inch).

Font.Italic

Description
A number that specifies whether the text should be italic.

In the painter
Italic on the Font page in the Properties view.

Value
Values are:

- `0`—Not italic
- `1`—Italic

Example

The following statements are specified for the Font.Italic, Font.Underline, and Font.Weight properties, respectively. If the employee has health insurance, the employee's information displays in italics. If not, the employee's information displays in bold and underlined:

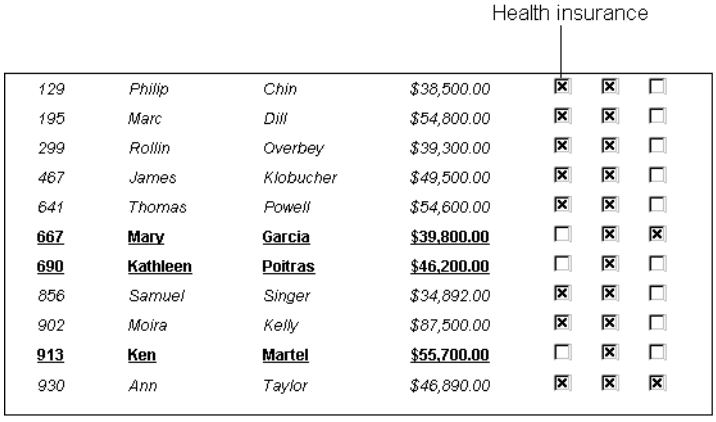
Condition for Font.Italic `If(bene_health_ins = 'Y', 1, 0)`

Condition for Font.Underline `If(bene_health_ins = 'N', 1, 0)`

Condition for Font.Weight `If(bene_health_ins = 'N', 700, 400)`

Statements are specified in this way for four controls: the emp_id column, the emp_fname column, the emp_lname column, and the emp_salary column. In the resulting DataWindow object, those with health insurance display in italics. Those without health insurance are emphasized with bold and underlining.

Figure 23-13: DataWindow with font highlighting of row data



Font.Strikethrough

Description

A number that specifies whether the text should be crossed out.

In the painter

Strikeout on the Font page in the Properties view.

Value

Values are:

- `0`—Not crossed out
- `1`—Crossed out

Example

The following statement is for the Font.Strikethrough property of the emp_id, emp_fname, emp_lname, and emp_salary columns. The status column must be included in the data source even though it does not appear in the DataWindow object itself. The statement says that if the employee's status is L, which means On Leave, cross out the text in the control:

```
If(status = 'L', 1, 0)
```

An extra text control is included to the right of the detail line. It only becomes visible if the status of the row is L (see “Visible” on page 607).

Figure 23-14 is a portion of the resulting DataWindow object. It shows two employees who are on leave. The four columns of information for each are crossed out.

Figure 23-14: DataWindow with font strikethrough highlighting

102	Fran	Whitney	$45,700.00	☒	☒	☐	
105	Matthew	Cobb	$62,000.00	☒	☒	☐	
160	Robert	Breault	$57,490.00	☒	☒	☒	
243	Natasha	Shishov	$72,995.00	☒	☒	☐	
~~247~~	~~Kurt~~	~~Driscoll~~	~~$48,023.69~~	☒	☒	☒	*On leave*
249	Rodrigo	Guevara	$42,998.00	☒	☒	☐	
266	Ram	Gowda	$59,840.00	☐	☒	☐	
278	Terry	Melkisetian	$48,500.00	☒	☒	☒	
316	Lynn	Pastor	$74,500.00	☒	☒	☒	
445	Kim	Lull	$87,900.00	☒	☒	☐	
453	Andrew	Rabkin	$64,500.00	☒	☒	☐	
~~479~~	~~Linda~~	~~Siperstein~~	~~$39,875.50~~	☒	☒	☐	*On leave*

Font.Underline

Description

A number that specifies whether the text should be underlined.

In the painter

Underline on the Font page in the Properties view.

Value

Values are:

- `0`—Not underlined
- `1`—Underlined

Example

The following statement, when applied to the Font.Underline property of columns of employee information, causes the information to be underlined if the employee does not have health insurance:

```
If(bene_health_ins = 'N', 1, 0)
```

See Figure 23-13 for a picture of this example.

Font.Weight

Description

The weight of the text.

In the painter

Bold on the Font page in the Properties view.

Value

Values are:

- `100`—Thin
- `200`—Extra light
- `300`—Light
- `400`—Normal
- `500`—Medium
- `600`—Semibold
- `700`—Bold
- `800`—Extrabold
- `900`—Heavy

Most commonly used values
The most commonly used values are 400 (Normal) and 700 (Bold). Your printer driver may not support all of the settings.

Example

The following statement, when applied to the Font.Weight property of columns of employee information, causes the information to be displayed in bold if the employee does not have health insurance:

```
If(bene_health_ins = 'N', 700, 400)
```

For a picture of this example, see “Font.Italic” on page 601.

Format

Description

The display format for a column.

In the painter

Format on the Format page in the Properties view.

Values

A string specifying the display format.

Example

The following statement, when applied to the Format property of the Salary column, causes the column to display the word `Overpaid` for any salary greater than $60,000 and `Underpaid` for any salary under $60,000:

```
If(salary>60000, 'Overpaid', 'Underpaid')
```

Edit style consideration
Typically you would have an Edit Mask edit style assigned to a salary column. Because edit styles take precedence over display formats, the strings `Overpaid` and `Underpaid` will not display in the salary column in the preceding example unless you also change the assigned edit style from Edit Mask to Edit, or select Use Format on the Format page of the Properties view.

Height

Description
The height of the column or other control.

In the painter
Height on the Position page in the Properties view.

Value
An integer in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), and pixels.

Example
The following statement causes the height of a rectangle to be 160 PBUs if the state column for the row has the value NY. Otherwise, the rectangle is 120 PBUs high:

```
if (state = 'NY', 160, 120)
```

Pen.Color

Description
The color of the line or the outline of a graphic control.

In the painter
Pen Color on the General page in the Properties view.

Value
A number that specifies the color of the line or outline.

For information on specifying colors, see “Specifying colors” on page 610.

Example
See the example for the Pen.Style property, next.

Pen.Style

Description

The style of the line or the outline of a graphic control.

In the painter

Pen Style on the General page in the Properties view.

Value

The only values that are supported on the Windows CE platform are:

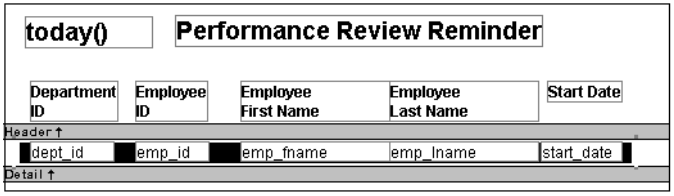
0—Solid
1—Dash
5—Null (no visible line)

Example

In this example, statements check the employee's start date to see if the month is the current month or the next month. Properties of a rectangle control placed behind the row of data are changed to highlight employees with months of hire that match the current month or the next month.

The Design view includes columns of data and a rectangle behind the data. The rectangle has been changed to black in the following picture to make it stand out.

Figure 23-15: Preview view of DataWindow with conditional pen style properties



The following statement is for the Pen.Color property of the line around the edge of the rectangle. If the month of the start date matches the current month or the next one, Pen.Color is set to light gray (12632256). If not, it is set to white (16777215), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
12632256, 16777215)
```

The following statement is for the Pen.Style property of the rectangle. If the month of the start date matches the current month or the next one, Pen.Style is set to Solid (0). If not, it is set to NULL (5), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
0, 5)
```

Pen.Width

Description — The width of the line or the outline of a graphic control.

In the painter — Pen Width on the General page in the Properties view.

Value — An integer in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example — The following statement causes the width of a line to be 10 PBUs if the state column for the row has the value NY. Otherwise, the line is 4 PBUs wide:

```
If(state = 'NY', 10, 4)
```

Protect

Description — The protection setting of a column.

In the painter — Protect on the General page in the Properties view.

Value — Values are:

- 0—False, the column is not protected
- 1—True, the column is protected

Timer_Interval

Description — The number of milliseconds between the internal timer events.

In the painter — Timer Interval on the General page in the Properties view.

Value — The default is 0 (which is defined to mean 60,000 milliseconds or one minute).

Visible

Description — Whether the control is visible in the DataWindow object.

In the painter — Visible on the General page in the Properties view.

Value — Values are:

- 0—Not visible
- 1—Visible

Example

The following statement is for the Visible property of a text control with the words On Leave located to the right of columns of employee information. The statement says that if the current employee's status is L, which means On Leave, the text control is visible. Otherwise, it is invisible:

```
If(status = 'L', 1, 0)
```

The status column must be retrieved
The status column must be included in the data source even though it does not appear in the DataWindow object itself.

The Design view includes the text control at the right-hand end of the detail line. The text control is visible at runtime only if the value of the status column for the row is L.

In the resulting DataWindow object, the text control is visible only for the two employees on leave. For a picture, see "Font.Strikethrough" on page 602.

Width

Description

The width of the control.

In the painter

Width on the Position page in the Properties view.

Value

An integer in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example

The following statement causes the width of a rectangle to be 500 PBUs if the state column for the row has the value NY. Otherwise, the rectangle is 1000 PBUs wide:

```
if (state = 'NY', 500, 1000)
```

X

Description

The distance of the control from the left edge of the DataWindow object. At runtime, the distance from the left edge of the DataWindow object is calculated by adding the margin to the x value.

In the painter

X on the Position page in the Properties view.

Value

An integer in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example

The following statement causes a rectangle to be located 6.250 inches from the left if the state column for the row has the value NY. Otherwise, the rectangle is 4.000 inches from the left:

```
If(state = 'NY', 6250, 4000)
```

X1, X2

Description

The distance of each end of the line from the left edge of the DataWindow object as measured in the Design view. At runtime, the distance from the left edge of the DataWindow object is calculated by adding the margin to the x1 and x2 values.

In the painter

X1, X2 on the Position page in the Properties view.

Value

Integers in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), and pixels.

Example

The following statements for the X1 and X2 properties of a line cause the line to extend from 6.250 to 7.150 inches from the left if the state column for the row has the value NY. Otherwise, the line extends from 4.000 to 6.000 inches from the left:

```
If(state = 'NY', 6250, 4000)
If(state = 'NY', 7150, 6000)
```

Y

Description

The distance of the control from the top of the band in which the control is located.

In the painter

Y on the Position page in the Properties view.

Value

An integer in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Y1, Y2

Description

The distance of each end of the specified line from the top of the band in which the line is located.

In the painter

Y1, Y2 on the Position page in the Properties view.

Value

Integers in the unit of measure specified for the DataWindow object. Units of measure include PBUs, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), and pixels.

Example

The following statements for the Y1 and Y2 properties of a line cause the line to be located .400 inches (Y1 and Y2 equal .400 inches) from the top of the detail band, if the state column for the row has the value NY. Otherwise, the line is located .250 inches (Y1 and Y2 equal .250 inches) from the top of the detail band:

```
If(state = 'NY', 400, 250)
If(state = 'NY', 400, 250)
```

Specifying colors

You specify a color by specifying a number that represents the color. You can specify the number explicitly or by using an expression that includes the RGB (*r*, *g*, *b*) function.

For the numbers and expressions that specify common colors, see Table 23-2 on page 611.

How the number is calculated

The formula for combining color values into a number is:

red + 256**green* + 256*256**blue*

where the amount of each primary color (red, green, and blue) is specified as a value from 0 to 255.

The RGB function calculates the number from the amounts of red, green, and blue specified.

Sample numeric calculation

To create cyan, you use blue and green, but no red. If you wanted to create the most saturated (bright) cyan, you would use maximum amounts of blue and green in the formula, which is indicated by the number 255 for each. The following statements show the calculation:

red + 256**green* + 256*256**blue*

0 + 256**255* + 256*256**255*

0 + 65280 + 16711680

16776960

Sample expression using the RGB function

The following expression specifies the brightest cyan:

RGB (*0*,*255*,*255*)

Notice that the expression specifies the maximum for green and blue (255) and 0 for red. The expression returns the value 16776960. To specify cyan, entering the expression `RGB(0, 255, 255)` is the same as entering the number `16776960`.

Numbers and expressions to enter for the common colors

Table 23-2 shows the numbers and expressions to enter for some common colors. The number and the expression return the same result; you can use either.

Table 23-2: Numbers and expressions for common colors

Color	Expression to enter	Number to enter	How the number is calculated
Black	RGB (0, 0, 0)	0	0 (no color)
Blue	RGB (0, 0, 255)	16711680	256*256*255 (blue only)
Cyan	RGB (0, 255, 255)	16776960	256*255 + 256*256*255 (green and blue)
Dark Green	RGB (0, 128, 0)	32768	256*128 (green only)
Green	RGB (0, 255, 0)	65280	256*255 (green only)
Light Gray	RGB (192, 192, 192)	12632256	192 + 256*192 + 256*256*192 (some red, green, and blue in equal amounts)
Lighter Gray	RGB (224, 224, 224)	14737632	224 + 256*224 + 256*256*224 (some red, green, and blue in equal amounts)
Lightest Gray	RGB (240, 240, 240)	15790320	240 + 256*240 + 256*256*240 (some red, green, and blue in equal amounts)
Magenta	RGB (255, 0, 255)	16711935	255 + 256*256*255 (red and blue)
Red	RGB (255, 0, 0)	255	255 (red only)

Color	Expression to enter	Number to enter	How the number is calculated
White	RGB (255, 255, 255)	16777215	255 + 256*255 + 256*256*255 (red, green, and blue in equal amounts at the maximum of 255)
Yellow	RGB (255, 255, 0)	65535	255 + 256*255 (red and green)

CHAPTER 24

Working with Graphs

About this chapter

This chapter describes how to build and use graphs in PocketBuilder.

Contents

About graphs

Often the best way to display information is graphically. Instead of showing users a series of rows and columns of data, you can present information as a graph in a DataWindow object or window. For example, in a sales application, you might want to present summary information in a column graph.

PocketBuilder provides many types of graphs and allows you to customize your graphs in many ways. Probably most of your use of graphs will be in a DataWindow object; the source of the data for your graphs will be the database.

You can also use graphs as standalone controls in windows (and user objects) and populate the graphs with data through scripts.

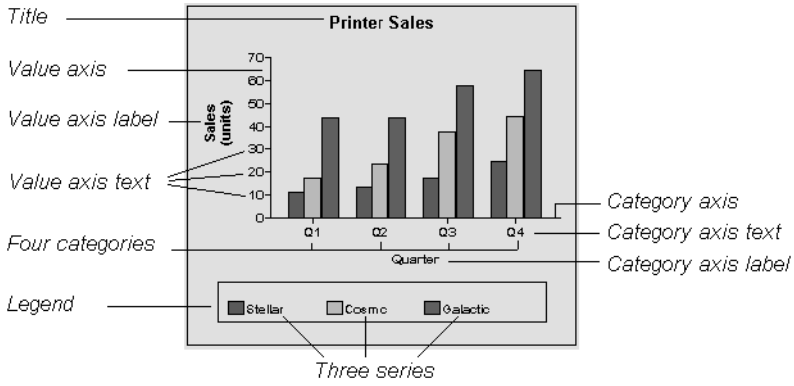
The way you define graphs is the same whether you are using them in a DataWindow object or directly in a window. However, the way you manipulate graphs in a DataWindow object is different from the way you manipulate them in a window.

Before using graphs in an application, you need to understand the parts of a graph and the kinds of graphs that PocketBuilder provides.

Parts of a graph

Figure 24-1 is a column graph created in PocketBuilder that contains most major parts of a graph. It shows quarterly sales of three products: Stellar, Cosmic, and Galactic printers:

Figure 24-1: Parts of a graph



How data is represented

Graphs display data points. To define graphs, you need to know how the data is represented. PocketBuilder organizes data into three components.

Table 24-1: Components of a graph

Component	Meaning
Series	A set of data points. Each set of related data points makes up one series. In Figure 24-1, there is a series for Stellar sales, another series for Cosmic sales, and another series for Galactic sales. Each series in a graph is distinguished by color, pattern, or symbol.
Categories	The major divisions of the data. Series data are divided into categories, which are often non-numeric. In Figure 24-1, the series are divided into four categories: Q1, Q2, Q3, and Q4. Categories represent values of the independent variable(s).
Values	The values for the data points (dependent variables).

Organization of a graph

Table 24-2 lists the parts of a typical graph.

Table 24-2: Organization of a graph

Part of graph	What it is
Title	An optional title for the graph. The title appears at the top of the graph.
Value axis	The axis of the graph along which the values of the dependent variable(s) are plotted. In a column graph such as Figure 24-1, the Value axis corresponds to the y axis in an XY presentation. In other types of graphs, such as a bar graph, the Value axis can be along the x dimension.
Category axis	The axis along which are plotted the major divisions of the data, representing the independent variable(s). In Figure 24-1, the Category axis corresponds to the x axis. It plots four categories: Q1, Q2, Q3, and Q4. These form the major divisions of data in the graph.
Series axis	The axis along which the series (a set of data points) are plotted in three-dimensional (3D) graphs.
Legend	An optional listing of the series. The graph in Figure 24-1 contains a legend that shows how each series is represented in the graph.

Types of graphs

PocketBuilder provides many types of graphs for you to choose from. You choose the type on the Define Graph Style page in the DataWindow wizard or in the General page in the Properties view for the graph.

Figure 24-2: Types of graph

Area, bar, column, and line graphs

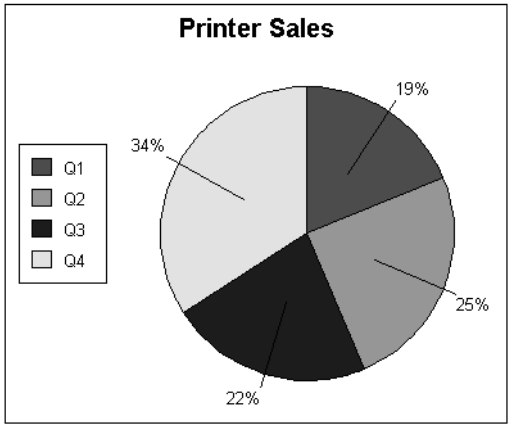
Area, bar, column, and line graphs are conceptually very similar. They differ only in how they physically represent the data values—whether they use areas, bars, columns, or lines to represent the values. All other properties are the same. Typically you use area and line graphs to display continuous data and bar and column graphs to display noncontinuous data.

The only difference between a bar graph and a column graph is the orientation. In column graphs, values are plotted along the y axis and categories are plotted along the x axis. In bar graphs, values are plotted along the x axis and categories are plotted along the y axis.

Pie graphs

Pie graphs typically show one series of data points with each data point displayed as a percentage of a whole. The following pie graph shows the sales for Stellar printers for each quarter. You can easily see the relative values in each quarter. (PocketBuilder automatically calculates the percentages of each slice of the pie.)

Figure 24-3: Pie chart example



You can have pie graphs with more than one series if you want; the series are shown in concentric circles. Multiseries pie graphs can be useful in comparing series of data.

Scatter graphs

Scatter graphs show data points with their x and y coordinates. Typically you use scatter graphs to show the relationship between two sets of numeric values. Non-numeric values, such as string and DateTime datatypes, do not display correctly.

Scatter graphs do not use categories. Instead, numeric values are plotted along both axes—as opposed to other graphs, which have values along one axis and categories along the other axis.

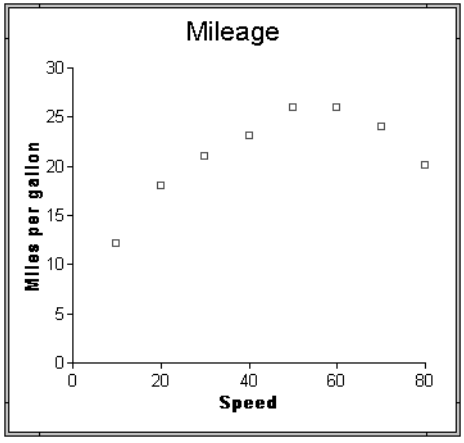
For example, the data in Table 24-3 shows the effect of speed on the mileage of a sedan.

Table 24-3: Data showing the effect of car speed on gas mileage

Speed	Mileage
10	12
20	18
30	21
40	23
50	26
60	26
70	24
80	20

The same data is displayed in a scatter graph in Figure 24-4.

Figure 24-4: Scatter graph example

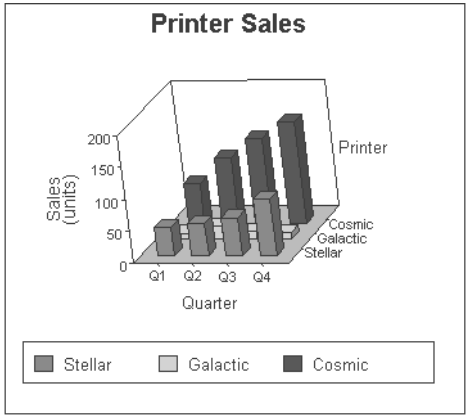


You can have multiple series of data in a scatter graph. You might want to plot mileage versus speed for several makes of cars in the same graph.

Three-dimensional graphs

You can also create 3-dimensional (3D) graphs of area, bar, column, line, and pie graphs. In 3D graphs (except for 3D pie graphs), series are plotted along a third axis (the Series axis) instead of along the Category axis. You can specify what perspective to use to show the third dimension.

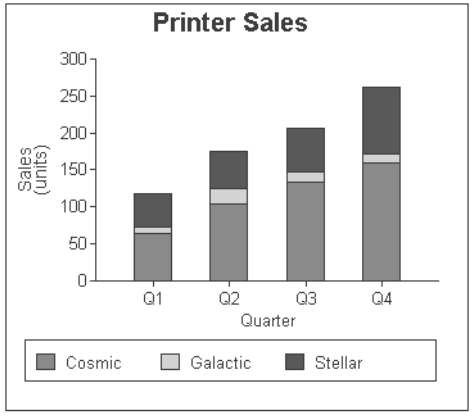
Figure 24-5: 3D graph example



Stacked graphs

In bar and column graphs, you can choose to stack the bars and columns. In stacked graphs, each category is represented as one bar or column instead of as separate bars or columns for each series.

Figure 24-6: Stacked graph example



Using graphs in applications

You can use graphs in DataWindow objects and in windows. You specify the properties of a graph (such as its type and title) the same way in a DataWindow object as in a window.

Using graphs in user objects
You can also use graphs in user objects. Everything in this chapter about using graphs in windows also applies to using graphs in user objects.

The major differences between using a graph in a DataWindow object and using a graph in a window (or user object) are in the following areas:

- Specifying the data for the graph

  In DataWindow objects, you associate columns in the database with the axes of a graph. In windows, you write scripts containing PowerScript functions to populate a graph.

- Specifying the location of the graph

  In DataWindow objects, you can place a graph in the foreground or in a DataWindow band. In windows, graphs are placed the same way as all other window controls.

Using graphs in DataWindow objects

Graphs are used most often in DataWindow objects; the data for the graph comes from tables in the database.

Graphs in DataWindow objects are dynamic

Graphs in DataWindow objects are tied directly to the data that is in the DataWindow object. As the data changes, the graph is automatically updated to reflect the new values.

Two techniques

You can use graphs in DataWindow objects in two ways:

- By including a graph as a control in a DataWindow object

  The graph enhances the display of information in a DataWindow object, such as a tabular or freeform DataWindow object. This technique is described in “Placing a graph in a DataWindow object” next.

- By using the Graph presentation style

The entire DataWindow object is a graph. The underlying data is not visible. This technique is described in "Using the Graph presentation style" on page 636.

Placing a graph in a DataWindow object

❖ **To place a graph in a DataWindow object:**

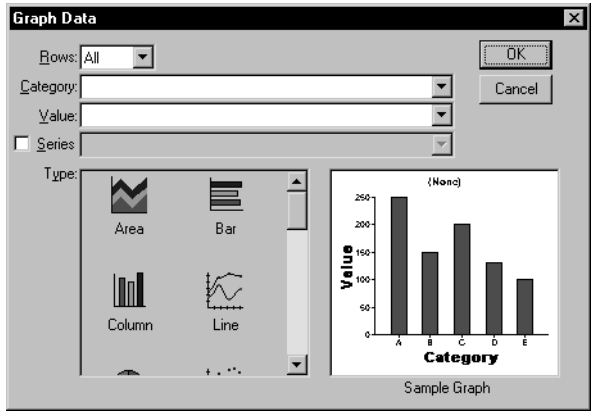
1 Open or create the DataWindow object that will contain the graph.

2 Select Insert>Control>Graph from the menu bar.

3 Click where you want the graph.

PocketBuilder displays the Graph Data dialog box.

Figure 24-7: Inserting a graph control in a DataWindow

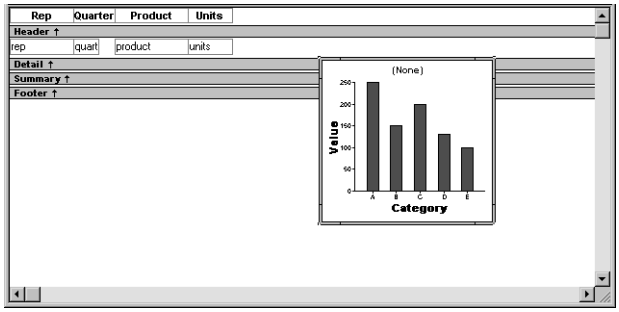


4 Specify which columns contain the data and the type of graph you want, and click OK.

For more information, see “Associating data with a graph” on page 624.

The Design view now contains a representation of the graph.

Figure 24-8: Design view with representation of a graph



5 Specify the graph's properties in the Properties view.

See “Using the graph's Properties view” next.

Using the graph's Properties view

A graph has a Properties view in which you can specify the data as well as other properties of the graph.

❖ **To display the graph’s Properties view:**

- Select Properties from the graph's pop-up menu.

The Properties view for a graph has property pages in which you specify information about the graph. Table 24-4 lists the property pages and describes what each property page specifies.

Table 24-4: Property pages for graphs

Property page	What it specifies
Axis	Labels, scale, information about major and minor divisions for the category axes.
Data	Where to get the graph's data.
General	Various general graph properties, including border, graph colors, whether to size the graph to the full screen display, whether to suppress the graph in newspaper columns, as well as graph type, title, legend location. Some general properties are restricted to certain types of graphs. Most 3D graphs have properties for perspective, rotation, and elevation, as well as spacing and depth. Other types of graph, such as bar and column graphs, have overlap percent and spacing properties.
Position	The x,y location of the upper left corner of the graph, its width and height, sliding options, and the layer in which the graph is to be positioned. Not applicable to PocketBuilder: Check boxes that determine whether the graph can be moved or resized during execution.
Text	Text properties for text controls that display on the graph, including title, axis text, axis label, and legend. Text properties include font, font style, font size, alignment, rotation, color, display expression, and display format.

Changing a graph's position and size

When you first place a graph in a DataWindow object, it is in the foreground—it sits above the bands in the DataWindow object. Unless you change this setting, the graph displays in front of any retrieved data.

❖ **To specify a graph's position and size:**

1 Select Properties from the graph's pop-up menu.

2 Select the settings you want for the options on the Position page of the Properties view.

Table 24-5: Settings on the Position page for graphs

Setting	Meaning
Layer	*Background* — The graph displays behind other elements in the DataWindow object. *Band* — The graph displays in one particular band. If you choose this setting, you should resize the band to fit the graph. Often you will want to place a graph in the Footer band. As users scroll through rows in the DataWindow object, the graph remains at the bottom of the screen as part of the footer. *Foreground* — (Default) The graph displays above all other elements in the DataWindow object.
Moveable	The graph can be moved in the Preview view and during execution. Not applicable to PocketBuilder applications at runtime.
Resizable	The graph can be resized in the Preview view and during execution. Not applicable to PocketBuilder applications at runtime.
Slide Left, Slide Up	The graph slides to the left or up to remove extra white space. For more information, see “Sliding controls to remove blank space in a DataWindow object” on page 510.
X, Y	The location of the upper-left corner of the graph.
Width, Height	The width and height of the graph.

3 Select or clear the Size To Display check box on the General page of the Properties view.

When you select the Size To Display check box, the graph fills the DataWindow object and resizes when users resize the DataWindow object. This setting is used automatically with the Graph presentation style.

Associating data with a graph

When using a graph in a DataWindow object, you associate axes of the graph with columns in the DataWindow object.

The only way to get data into a graph in a DataWindow object is through columns in the DataWindow object. You cannot add, modify, or delete data in the graph except by adding, modifying, or deleting data in the DataWindow object.

You can graph data from any columns retrieved into the DataWindow object. The columns do not have to be displayed.

About the examples

The process of specifying data for a graph is illustrated below using the Printer table in the EAS Demo DB. The EAS Demo DB is installed with PowerBuilder. The demo database that ships with SQL Anywhere contains many of the same tables as EAS Demo DB, but it does not have the Printer table.

You can add the Printer table to a SQL Anywhere database from the ISQL Session view of the PocketBuilder Database painter by:

1. Pasting the contents of the *Printer.SQL* file that the PocketBuilder setup program installs in the *Code Examples\SQL* directory in the PocketBuilder path.
2. Selecting the Execute icon in the Database painter toolbar (or selecting Design>Execute ISQL from the main menu) while connected to a SQL Anywhere database on the desktop.
3. Synchronizing the desktop database with a remote database on the device or emulator (for runtime display on a Windows CE platform).

❖ **To specify data for a graph:**

1. If you are creating a new graph, the Graph Data dialog box displays. Otherwise, select Properties from the graph's pop-up menu and select the Data tab in the Properties view.
2. Fill in the boxes as described in the sections that follow, and click OK.

Specifying which rows to include in a graph

The Rows drop-down list allows you to specify which rows of data are graphed at any one time.

Table 24-6: Specifying which rows to include in a graph

Setting	Meaning
All	Graphs the data from all the rows that have been retrieved but not filtered or deleted (that is, the rows in the primary buffer of the DataWindow object)
Page	Graphs only the data from the rows that are currently displayed on the page
Group n	Graphs only the data in the specified group (in a grouped DataWindow object)

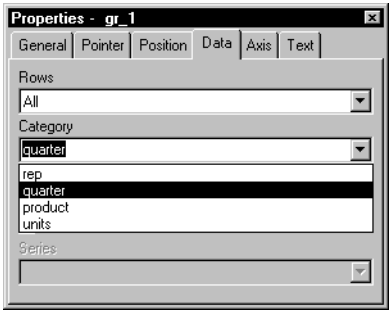
If you select Group

If you are graphing data in the current group in a grouped DataWindow object and have several groups displayed at the same time, you should localize the graph in a group-related band in the Design view to make it clear which group the graph represents. Usually, the group header band is the most appropriate band.

Specifying the categories

Specify the column or expression whose values determine the categories. In the Graph Data page in the Graph dialog box and on the Data page in the Properties view, you can select a column name from a drop-down list. In the Data tab, you can also type an expression.

Figure 24-9: Selecting a column as a category in the Properties view



There is an entry along the Category axis for each different value of the column or expression you specify.

Using display values of data

If you are graphing columns that use code tables, which allow you to store data a data value but display it to users with more meaningful display values, the graph uses the column's data values by default. To have the graph use a column's display values, use the LookupDisplay DataWindow expression function when specifying Category or Series. LookupDisplay returns a string that matches the display value for a column:

LookupDisplay (*column*)

For more about code tables, see "Defining a code table" on page 552. For more about LookupDisplay, see the *DataWindow Reference* in the online Help.

Specifying the values

PocketBuilder populates the Value drop-down list. The list includes the names of all the retrieved columns as well as the following aggregate functions:

- Count for all non-numeric columns
- Sum for all numeric columns

Select an item from the drop-down list or type an expression (in the Properties view). For example, if you want to graph the sum of units sold, you can specify:

```
sum(units for graph)
```

To graph 110 percent of the sum of units sold, you can specify:

```
sum(units*1.1 for graph)
```

Specifying the series

Graphs can have one or more series.

Single-series graphs

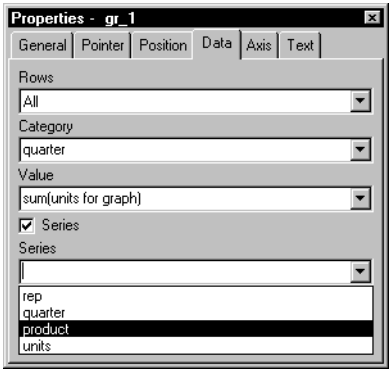
If you want only one series—that is, if you want to graph all retrieved rows as one series of values—leave the Series box empty.

Multiple-series graphs

If you want to graph more than one series, select the Series check box and specify the column that will provide the series values.

You can select column names from the drop-down list.

Figure 24-10: Selecting a column to provide graph series values



There is a set of data points for each different value of the column you specify here. For example, if you specify in the Series box a column that has 10 values, then your graph will have 10 series: one set of data points for each different value of the column.

Using expressions

You can also specify expressions for Series (on the Data page of the Properties view). For example, you could specify the following for Series:

```
Units / 1000
```

In this case, if a table had unit values of 10,000, 20,000, and 30,000, the graph would show series values of 10, 20, and 30.

Specifying multiple entries

You can specify more than one of the retrieved columns to serve as series. Separate multiple entries by commas.

You must specify the same number of entries in the Value box as you do in the Series box. The first value in the Value box corresponds to the first series identified in the Series box, the second value corresponds to the second series, and so on. The example about graphing actual and projected sales in "Examples" next illustrates this technique.

Examples

This section shows how to specify the data for several different graphs of the data in the Printer table in the EAS Demo DB. The table records quarterly unit sales of three printers by three sales representatives.

Table 24-7: Q1 and Q4 data from the Printer table in EAS Demo DB

Rep	Quarter	Product	Units
Jones	Q1	Cosmic	5
Perez	Q1	Cosmic	26
Simpson	Q1	Cosmic	33
Jones	Q1	Galactic	2
Perez	Q1	Galactic	1
Simpson	Q1	Galactic	6
Jones	Q1	Stellar	18
Perez	Q1	Stellar	15
Simpson	Q1	Stellar	12
…	…	…	…
Jones	Q4	Cosmic	52
Perez	Q4	Cosmic	48
Simpson	Q4	Cosmic	60
Jones	Q4	Galactic	3
Perez	Q4	Galactic	6
Simpson	Q4	Galactic	3
Jones	Q4	Stellar	24
Perez	Q4	Stellar	36
Simpson	Q4	Stellar	30

Graphing total sales

To graph total sales of printers in each quarter, retrieve all the columns into a DataWindow object and create a graph with the following settings on the Data page in the Properties view:

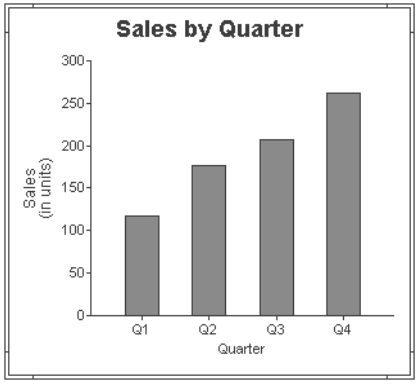
- Set Rows to `All`
- Set Category to `quarter`
- Set Value to `sum(units for graph)`

Leave the Series check box and text box empty.

The Quarter column serves as the category. Because the Quarter column has four values (Q1, Q2, Q3, and Q4), there will be four categories along the Category axis. Suppose that you want only one series (total sales in each quarter). You can leave the Series box empty or type a string literal to identify the series in a legend. Setting Value to `sum(units for graph)` graphs total sales in each quarter.

Figure 24-11 shows the resulting column graph. PocketBuilder automatically generates the category text based on the data in the table.

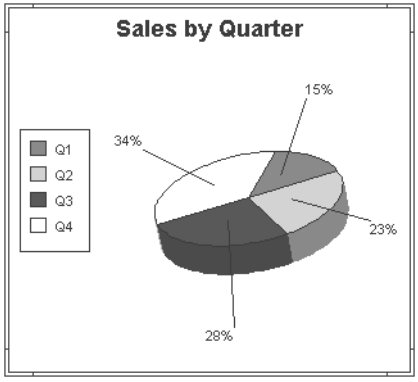
Figure 24-11: Column graph showing printer sales by quarter



In the preceding graph, there is one set of data points (one series) across four quarters (the category values).

Figure 24-12 is a pie graph, which has exactly the same properties as the column graph above except for the type, which is 3D Pie.

Figure 24-12: 3D Pie graph showing printer sales by quarter



In pie graphs, categories are shown in the legend.

Graphing unit sales of each printer

To graph total quarterly sales of each printer type, retrieve all the columns into a DataWindow object and create a graph with the following settings on the Data page in the Properties view:

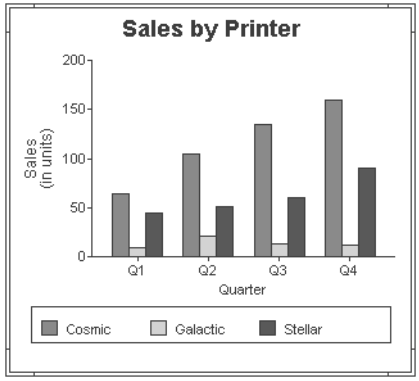
- Set Rows to `All`
- Set Category to `quarter`
- Set Value to `sum(units for graph)`

- Select the Series check box
- Set Series to `product`

Suppose that you want a different series for each printer, so the column Product can serve as the series. Because the Product column has three values (Cosmic, Galactic, and Stellar), there will be three series in the graph. As in the first example, you want a value for each quarter, so the Quarter column serves as the category, and you want to graph total sales in each quarter, so the Value box is specified as `sum(units for graph)`.

Figure 24-13 shows the resulting graph. PocketBuilder automatically generates the category and series labels based on the data in the table. The series labels display in the graph's legend.

Figure 24-13: Column graph with series data by printer type



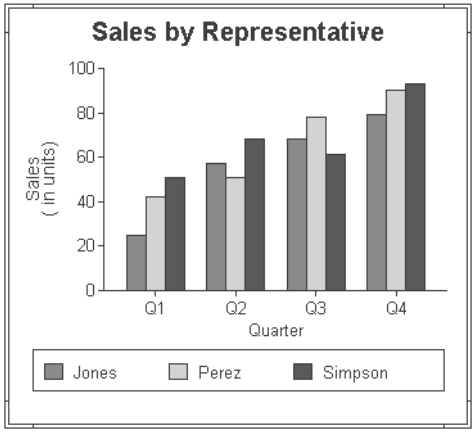
Graphing unit sales by representative

To graph quarterly sales made by each representative, create a graph with the following settings on the Data page in the Properties view:

- Set Rows to `All`
- Set Category to `quarter`
- Set Value to `sum(units for graph)`
- Select the Series check box
- Set Series to `rep`

Figure 24-14 shows the resulting graph.

Figure 24-14: Column graph with series data by sales representative

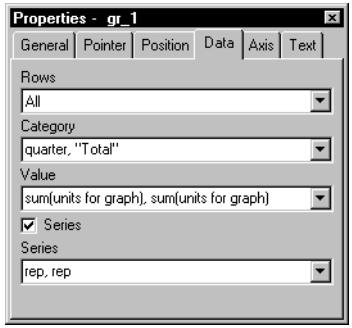


Graphing unit sales by representative and total sales

To graph quarterly sales made by each representative, plus total sales for each printer, create a graph with the following settings on the Data page in the Properties view:

- Set Rows to `All`
- Set Category to `quarter, "Total"`
- Set Value to `sum(units for graph), sum(units for graph)`
- Select the Series check box
- Set Series to `rep, rep`

Figure 24-15: Adding two series to a graph

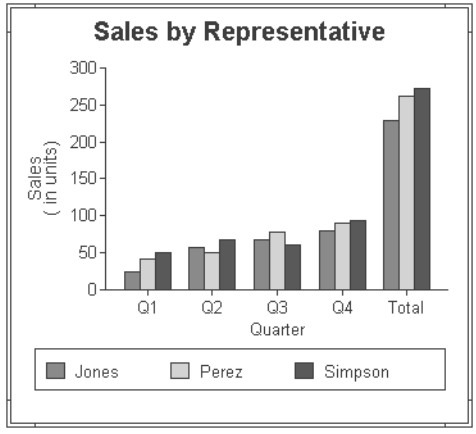


Here you have two types of categories: the first is Quarter, which shows quarterly sales, as in the previous graph. You also want a category for total sales. There is no corresponding column in the DataWindow object, so you can simply type the literal "Total" to identify the category. You separate multiple entries with a comma.

For each of these category types, you want to graph the sum of units sold for each representative, so the Value and Series values are repeated.

Figure 24-16 shows the resulting graph.

Figure 24-16: Graph with dual series data for two categories



Notice that PocketBuilder uses the literal "Total" supplied in the Category box in the Graph Data window as a value in the Category axis.

Graphing actual and projected sales

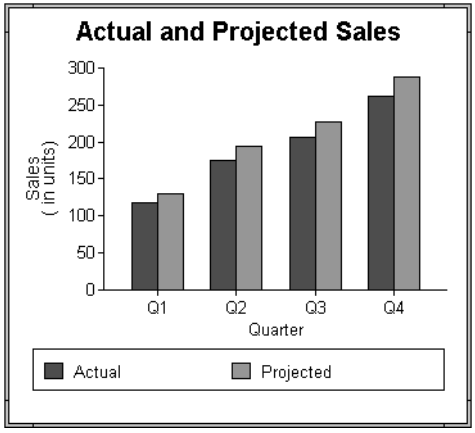
To graph total quarterly sales of all printers and projected sales for next year, where you assume that sales will increase by 10% next year, create a graph with the following settings on the Data page in the Properties view:

- Set Rows to `All`
- Set Category to `quarter`
- Set Value to `sum(units for graph), sum(units*1.1 for graph)`
- Select the Series check box
- Set Series to `'Actual','Projected'`

You are using labels to identify two series, Actual and Projected. Note the single quotation marks around the literals. For Values, you enter the expressions that correspond to Actual and Projected sales. For Actual, you use the same expression as in the examples above, `sum(units for graph)`. For Projected sales, you multiply each unit sale by 1.1 to get the 10 percent increase. Therefore, the second expression is `sum(units*1.1 for graph)`.

Figure 24-17 shows the resulting graph. PocketBuilder uses the literals you typed for the series as the series labels in the legend.

Figure 24-17: Graph with dual series data for a single category



Using overlays

Overlays are useful when you want to call special attention to one of the series in a graph, particularly in a bar or column graph. You call attention to the series by defining it as an overlay. An overlay series is graphed as a line on top of the other series in the graph. To define a series as an overlay, do as follows:

- To specify a column name to identify the series, specify this for the series:

  "@overlay~t" + *ColumnName*

- To use a label to identify the series, specify this for the series:

  "@overlay~t*SeriesLabel* "

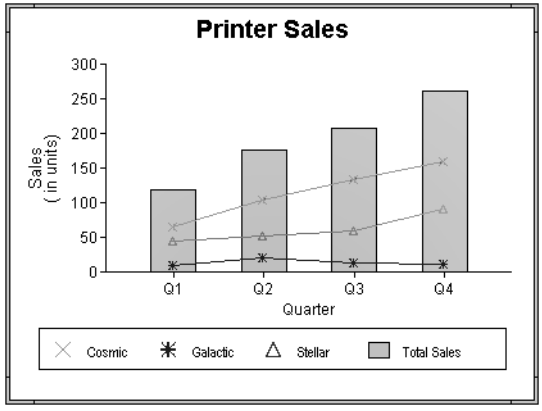
Examples

To graph sales in each quarter and overlay the sales of each individual printer, specify the graph's data as in “Graphing unit sales of each printer” on page 630, but use the following expression in the Series box:

```
"Total Sales", "@overlay~t + product"
```

Figure 24-18 shows the resulting graph.

Figure 24-18: Graph with overlay for individual product sales by quarter



To graph unit sales of printers by quarter and overlay the largest sale made in each quarter, change the Value expression to this:

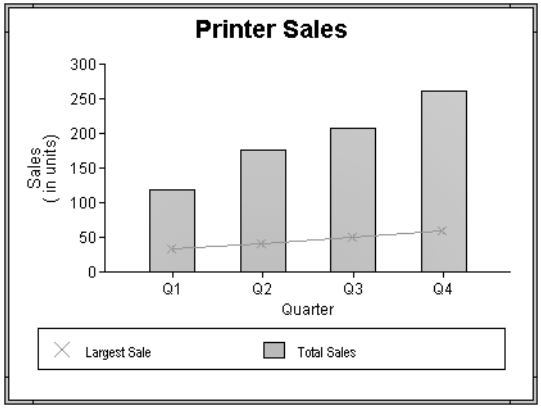
```
sum(units for graph), max(units for graph)
```

Change the Series expression to this:

```
"Total Sales", "@overlay~tLargets Sale"
```

Figure 24-19 shows the resulting graph.

Figure 24-19: Graph with overlay for largest sale by quarter



Using the Graph presentation style

Instead of embedding a graph in a DataWindow object, you can use the Graph presentation style to create a DataWindow object that is only a graph: the underlying data is not displayed.

❖ **To use the Graph presentation style:**

1 Select File>New from the menu bar.

The New dialog box displays.

2 Select the DataWindow tab, select the Graph presentation style, and click OK.

3 Specify the data source for the DataWindow object.
If you want data to be retrieved into the Preview view automatically, select the Retrieve on Preview check box.

For more information, see Chapter 17, “Defining DataWindow Objects.”

4 Enter the definitions for the series, categories, and values, as described in “Associating data with a graph” on page 624, and click Next.

Note that when using the Graph presentation style, the graph always graphs all rows; you cannot specify page or group.

5 Enter a title for the graph, select a graph type, and click Next.

6 Review your specifications and click Finish.

A model of the graph displays in the Design view.

7 Specify the properties of the graph, as described in "Defining a graph's properties" next.

8 Save the DataWindow object in a library.

9 Associate the graph DataWindow object with a DataWindow control in a window or user object.

During execution, the graph fills the entire control.

Defining a graph's properties

This section describes properties of a graph that are used whether the graph is in a DataWindow object or in a window. To define the properties of a graph, you use the graph's Properties view. For general information about the property pages, see "Using the graph's Properties view" on page 622.

Using the General page in the graph's Properties view

You name a graph and define its basic properties on the General page in the graph's Properties view.

❖ **To specify the basic properties of a graph:**

- Select Properties from the graph's pop-up menu and then select the General page in the Properties view

About the model graph in the Design view

As you modify a graph's properties, PocketBuilder updates the model graph shown in the Design view so that you can get an idea of the graph's basic layout:

- PocketBuilder uses the graph title and axis labels you specify
- PocketBuilder uses sample data (not data from your DataWindow object) to illustrate series, categories, and values

In Preview view, PocketBuilder displays the graph with data.

Naming a graph

You can modify a graph in scripts during execution. To reference a graph during execution, you use its name.

❖ **To name a graph:**

- On the General page for the graph, assign a meaningful name to the graph in the Name box

Defining a graph's title

The graph title displays at the top of the graph.

❖ **To specify a graph's title:**

- On the General page for the graph, enter a title in the Title box

---

Multiline titles
You can force a new line in a title by embedding ~n.

For information about specifying properties for the title text, see "Specifying text properties for titles, labels, axes, and legends" on page 639.

Specifying the type of graph

You can change the graph type at any time in the development environment. (To change the type at runtime, modify a graph's GraphType property.)

❖ **To specify the graph type:**

- On the General properties page for the graph, select a graph type from the Graph Type drop-down list

Using legends

A legend provides a key to your graph's series.

❖ **To include a legend for a series in a graph:**

- On the General properties page for the graph, specify where you want the legend to appear by selecting a value in the Legend drop-down list

For information on specifying text properties for the legend, see "Specifying text properties for titles, labels, axes, and legends" on page 639.

Specifying a border

You can specify the border that PocketBuilder places around a graph.

❖ **To specify a border for a graph:**

- On the General properties page for the graph, select the type of border to use from the Border drop-down list

Specifying point of view in 3D graphs

If you are defining a 3D graph, you can specify the point of view that PocketBuilder uses when displaying the graph.

❖ **To specify a 3D graph's point of view:**

1 On the General properties page for the graph, adjust the point of view along the three dimensions of the graph:

- To change the perspective, move the Perspective slider
- To rotate the graph, move the Rotation slider
- To change the elevation, move the Elevation slider

2 Define the depth of the graph by entering a value in the Depth box.

The value you enter for the depth is a percentage of the width of the graph. You can use the spin control to set this value.

Sorting data for series and categories

You can specify how to sort the data for series and categories. By default, the data is sorted in ascending order.

❖ **To specify how to sort the data for series and categories in a graph:**

1 Select Properties from the graph's pop-up menu and then select the Axis page in the Properties view.

2 Select the axis for which you want to specify sorting.

3 In the Sort drop-down list, select Ascending (order), Descending (order), or Unsorted.

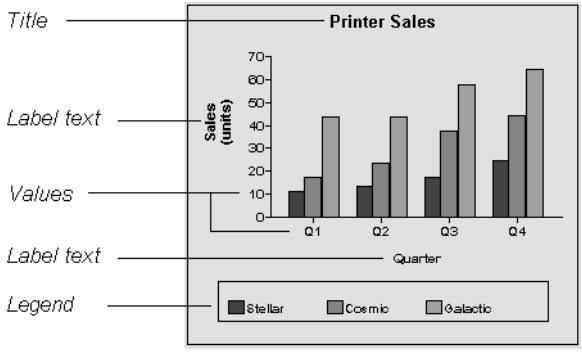
The Sort drop-down list is at the bottom of the Axis page in the Properties view for the graph.

Specifying text properties for titles, labels, axes, and legends

A graph can have four text elements:

- Title
- Labels for the axes
- Text that shows the values along the axes
- Legend

Figure 24-20: Text elements of a graph



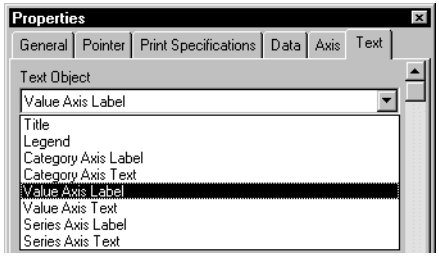
You can specify properties for each text element.

❖ **To specify text properties for the title, labels, axis values, and legend of a graph:**

1 Select Properties from the graph's pop-up menu and then select the Text page in the Properties view.

2 Select a text element from the list in the Text Object drop-down list.

Figure 24-21: Selecting text elements to change their properties



3 Specify the font and its characteristics.

Using Auto Size

With Auto Size in effect, PocketBuilder resizes the text appropriately whenever the graph is resized. With Auto Size disabled, you specify the font size of a text element explicitly.

❖ **To have PocketBuilder automatically size a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Select the Autosize check box (this is the default).

❖ **To specify a font size for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Clear the Autosize check box.

3 Select the Font size in the Size drop-down list.

Rotating text

For all the text elements, you can specify the number of degrees by which you want to rotate the text.

❖ **To specify rotation for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Specify the rotation you want in the Escapement box, using tenths of a degree (450 means 45 degrees)

Changes you make here are shown in the model graph in the Design view and in the Preview view.

Using display formats

Display formats are masks in which certain characters have special significance that aid in the presentation of information to the user. For more information about defining display formats, see Chapter 21, “Displaying and Validating Data.”

❖ **To use a display format for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Type a display format in the Format box or choose one from the pop-up menu.

To display the pop-up menu, click the button to the right of the Format box.

Modifying display expressions

You can specify an expression for the text that is used for each graph element. The expression is evaluated at execution time.

❖ **To specify an expression for a text element in a graph:**

1 On the Text properties page for the graph, select a text element from the list in the Text Object drop-down list.

2 Click the button next to the Display Expression box.

The Modify Expression dialog box displays.

3 Specify the expression.

You can paste functions, column names, and operators. Included with column names in the Columns box are statistics about the columns, such as counts and sums.

4 Click OK to return to the graph's Properties view.

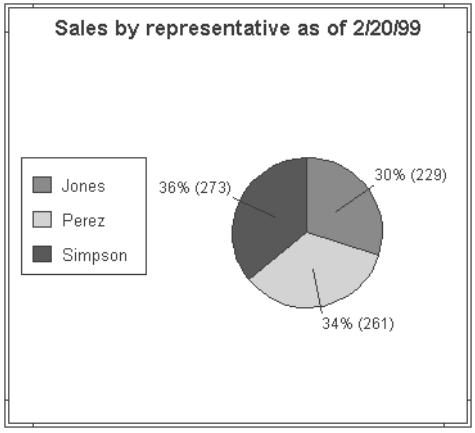
Example

By default, when you generate a pie graph, PocketBuilder puts the title at the top and labels each slice of the pie with the percentage that slice represents of the whole. Percentages are accurate to two decimal places.

Figure 24-22 has been enhanced as follows:

- The current date displays in the title
- The percentages are rounded to integers
- The raw data for each slice is shown in addition to the percentages

Figure 24-22: Pie chart with display enhancements



To enhance the pie chart, the display expressions were modified for the title and pie graph labels as follows:

Element	Original expression	Modified expression
Title	`title`	`title + " as of " + date(today())`
Pie graph labels	`if(seriescount > 1, series, string (percentofseries, "0.00%"))`	`if(seriescount > 1, series, string(percentofseries,"0% ") + " (" + value + ")" )`

Specifying overlap and spacing

With bar and column charts, you can specify the properties in Table 24-8.

Table 24-8: Overlap and spacing properties for bar and column charts

Property	Meaning
Overlap	The percentage by which bars or columns overlap each other. The default is 0 percent, meaning no overlap.
Spacing	The amount of space to leave between bars or columns. The default is 100 percent, which leaves a space equal to the width of a bar or column.

❖ **To specify overlap and spacing for the bars or columns in a graph:**

1 Select Properties from the graph's pop-up menu and then select the Graph tab.

2 Specify a percentage for Overlap (% of width) and Spacing (% of width).

Specifying axis properties

Graphs have two or three axes. You specify the axes' properties in the Axis page in the graph's Properties view.

❖ **To specify properties for an axis of a graph:**

1 Select Properties from the graph's pop-up menu and then select the Axis page in the Properties view.

2 Select the Category, the Value, or the Series axis from the Axis drop-down list.

If you are not working with a 3D graph, the Series Axis options are disabled.

3 Specify the properties as described next.

Specifying text properties

You can specify the characteristics of the text that displays for each axis. Table 24-9 shows the two kinds of text associated with an axis.

Table 24-9: Text types associated with each axis of a graph

Type of text	Meaning
Text	Text that identifies the values for an axis.
Label	Text that describes the axis. You specify the label text in a painter. You can use ~n to embed a new line within a label.

For information on specifying properties for the text, see "Specifying text properties for titles, labels, axes, and legends" on page 639.

Specifying datatypes

The data graphed along the Value, Category, and Series axes has an assigned datatype. The Series axis always has the datatype String. The Value and Category axes can have the datatypes listed in Table 24-10.

Table 24-10: Datatypes for Value and Category axes

Axis	Possible datatypes
Both axes (for scatter graph)	Number, Date, Time
Value (other graph types)	Number, Date, DateTime, Time
Category (other graph types)	String, Number, Date, DateTime, Time

For graphs in DataWindow objects, PocketBuilder automatically assigns the datatypes based on the datatype of the corresponding column; you do not specify them.

For graphs in windows, you specify the datatypes yourself. Be sure you specify the appropriate datatypes so that when you populate the graph (using the AddData function), the data matches the datatype.

Scaling axes

You can specify the properties listed in Table 24-11 to define the scaling used along numeric axes.

Table 24-11: Properties for scaling on numeric axes

Property	Meaning
Autoscale	If selected (the default), PocketBuilder automatically assigns a scaling for the numbers along the axis.
Round To, Round To Unit	Specifies how to round the end points of the axis. This rounds only the range displayed along the axis; it does not round the data itself. You can specify a number and a unit. The unit is based on the datatype; you can specify Default as the unit to have PocketBuilder decide for you. For example, if the Value axis is a Date column, you can specify that you want to round the end points of the axis to the nearest five years. In this case, if the largest data value is the year 1997, the axis extends up to 2000.
Minimum Value, Maximum Value	The smallest and largest numbers to appear on the axis (disabled if you have selected Autoscale).
Scale Type	Specifies linear or logarithmic (common or natural) scaling.

Using major and minor divisions

You can divide axes into divisions. Each division is identified by a tick mark, which is a short line that intersects an axis. In the Sales by Printer graphs shown in “Examples” on page 628, the graph's Value axis is divided into major divisions of 50 units each. PocketBuilder divides the axes automatically into major divisions.

❖ **To define divisions for an axis of a graph:**

1 To divide an axis into a specific number of major divisions, type the number of divisions you want in the MajorDivisions box.

Leave the number 0 to have PocketBuilder automatically create divisions. PocketBuilder labels each tick mark in major divisions. If you do not want each tick mark labeled, enter a value in the DisplayEveryNLabels box. For example, if you enter 2, PocketBuilder labels every second tick mark for the major divisions.

2 To use minor divisions, which are divisions within each major division, type the appropriate number in the MinorDivisions box.

To use no minor divisions, leave the number 0.

When using logarithmic axes
If you want minor divisions, specify 1; otherwise, specify 0.

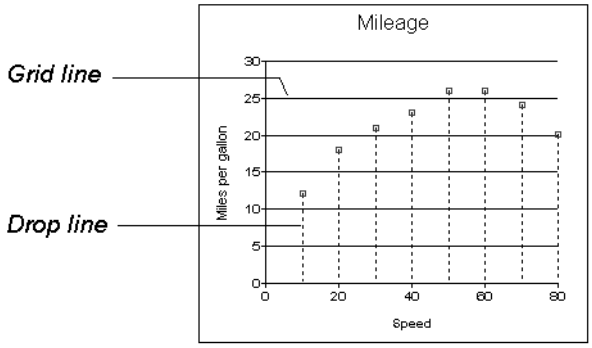
Representing divisions with grid and drop lines

You can specify lines to represent the divisions as described in Table 24-12 and illustrated in Figure 24-23.

Table 24-12: Representing graph divisions with grid and drop lines

Line	Meaning
Grid line	A line that extends from a tick mark across the graph. Grid lines make graphs easier to read.
Drop line	A line that extends vertically from a data point to its axis (not available for all graph types).

Figure 24-23: Grid and drop lines in a graph



Using line styles

You can define line styles for the components of a graph listed in Table 24-13.

Table 24-13: Components of a graph that can have line styles

Component	Meaning
PrimaryLine	The axis itself
SecondaryLine	The axis parallel to and opposite the primary axis
OriginLine	A grid line that represents the value zero
Frame	The frame for the axis in 3D graphs (disabled for 2D graphs)

Using graphs in windows

In addition to using graphs in DataWindow objects, you can also place graphs directly in windows and visual user objects. You define properties for a graph control in the Window painter and use scripts to populate the graph with data and to modify properties for the graph during execution.

This section describes procedures unique to using graphs in windows and visual user objects. For general graph properties, see “Defining a graph's properties” on page 637.

Placing a graph in a window

This procedure for placing a graph in a window in the Window painter can also be used for placing a graph on a user object in the User Object painter.

❖ **To place a graph in a window:**

1. Open the Window painter and select the window that will contain the graph.
2. Select Insert>Control>Graph from the menu bar.
3. Click where you want the graph.

   PocketBuilder displays a model of the graph in the window.
4. Specify properties for the graph, such as type, title, text properties, and axis properties.

   See “Defining a graph's properties” on page 637.
5. Write one or more scripts to populate the graph with data.

   See the chapter on manipulating graphs in the *Resource Guide*.

Using the graph's Properties view in the Window painter

A graph's Properties view in the Window and User Object painters is similar to the one in the DataWindow painter but the Properties view in the Window and User Object painter:

- Does not have buttons for specifying property conditional expressions next to properties
- Does not have Data and Position property pages
- Has an Other properties page, which you use to specify position properties for the graph control

For more information, see “Using the graph's Properties view” on page 622.

PART 6

Testing and Running Your Application

This part describes how to test an application from within PocketBuilder on the desktop: in debug mode, where you can set breakpoints and examine the state of your application as it executes, and in test mode, where the application runs until you stop it or an error occurs. It also describes how to generate trace information that you can use to improve your application's performance.

The last chapter in this part describes how to build and package an application for distribution to users.

CHAPTER 25

Testing and Debugging Applications

About this chapter

This chapter describes how to test and debug an application in PocketBuilder on the desktop. You can also test PocketBuilder applications on a Windows CE emulator running on the desktop.

Contents

Overview of debugging and testing applications

After you build all or part of an application and compile and save its objects, you can run the application. The PocketBuilder development environment provides two ways to run an application on the desktop: in debug mode and in test mode.

Debug mode

In debug mode, you can insert breakpoints (stops) in scripts and functions, single-step through code, and display the contents of variables to locate logic errors that will result in errors during execution.

Test mode

In the desktop test mode, the application responds to user interaction and runs until you stop it or until a runtime error occurs.

You can also collect trace information while you run your application in test mode, then use the trace data to profile your application. For more information, see Chapter 26, “Tracing and Profiling Applications.”

You can also test an application by deploying and running it on a supported emulator or device.

Troubleshooting advantages by deployment platform

Table 25-1 shows the advantages and disadvantages of troubleshooting an application, depending on where and how you deploy it.

Table 25-1: Troubleshooting advantages by deployment platform

Platform	Advantage	Disadvantage
Running from the IDE	• Fastest turnaround time • Full debugger support	• Least like PDA environment • Different processor, VM, and DLLs • Different database configuration
Running in the Emulator	• Somewhat like the PDA • Lets you test applications when a PDA is not available	• Emulators have bugs of their own • Can be cumbersome to install and use
Running in the PDA	• Real target environment • Real database environment	• Difficult to debug • No tracing or remote debugging currently available

Debugging an application

Sometimes an application does not behave the way you think it will. Perhaps a variable is not being assigned the value you expect, or a script does not perform as desired. In these situations, you can examine your application by running it in debug mode.

When you run the application in debug mode, PocketBuilder stops execution before it executes a line containing a breakpoint (stop). You can then step through the application and examine its state.

❖ **To debug an application:**

1 Open the debugger.

2 Set breakpoints at places in the application where you have a problem.

3 Run the application in debug mode.

See “Starting the debugger” next.

4 When execution is suspended at a breakpoint, look at the values of variables, examine the properties of objects in memory and the call stack, or change the values of variables.

5 Step through the code line by line.

6 As needed, add or modify breakpoints as you run the application.

7 When you uncover a problem, fix your code and run it in the debugger again.

Starting the debugger

❖ **To open the debugger:**

- Do one of the following:
  - In the System Tree, highlight a target and select Debug from the pop-up menu
  - Click the Debug or Select and Debug button on the PowerBar
  - Select Run>Debug or Run>Select and Debug from the menu bar

  The Debug button opens the debugger for the last target run in debug or test mode. The name of the current target is displayed in the Debug button tool tip. The Select and Debug button opens a dialog box that lets you select the target to be debugged.

Views in the debugger

The debugger contains several views. Each view shows a different kind of information about the current state of your application or the debugging session. Table 25-2 summarizes what each view shows and what you can do from that view.

Table 25-2: Views in the debugger

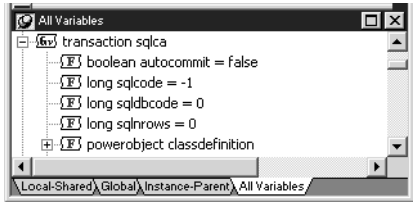
View	What it shows	What you can do
Breakpoints	A list of breakpoints with indicators showing whether the breakpoints are currently active or inactive	Set, enable, disable, and clear breakpoints, set a condition for a breakpoint, and show source for a breakpoint in the Source view.
Call Stack	The sequence of function calls leading up to the function that was executing at the time of the pause for a breakpoint, shown as the script and line number from which the function was called	Examine the context of the application at any line in the call stack.

View	What it shows	What you can do
Objects in Memory	An expandable list of objects currently in memory	View the names and memory locations of instances of each memory object, and property values of each instance.
Source	The full text of a script	Go to a specific line in a script, find a string, open another script, including ancestor and descendent scripts, and manage breakpoints.
Source Browser	An expandable hierarchy of objects in your application	Select any script in your application and display it in the Source view.
Source History	A list of the scripts that have been displayed in the Source view	Select any script in the Source History and display it in the Source view.
Variables	An expandable list of all the variables in scope	Select which kinds of variables are shown in the view, change the value of a variable, and set a breakpoint when a variable changes.
Watch	A list of variables you have selected to watch as the application proceeds	Change the value of a variable, set a breakpoint when a variable changes, and add an arbitrary expression to the Watch view.

Changing Variable views

The default debugger layout contains a separate view for each variable type in a stacked pane. You can combine two or more Variables views in a single pane. For example, you might want to combine local and global variables in a single view that you keep at the top of the stacked pane.

Figure 25-1: Variables views in a stacked pane



❖ **To display multiple variable types in a single view:**

1 Display the pop-up menu for a pane that contains a Variables view you want to change.

2 Click the names of the variable types you want to display.

A check mark displays next to selected variable types. The pop-up menu closes each time you select a variable type or clear a check mark, so you need to reopen the menu to select an additional variable type.

When you select or clear variable types, the tab for the pane changes to show the variable types displayed on that pane.

Setting breakpoints

A breakpoint is a point in your application code where you want to interrupt the normal execution of the application while you are debugging. If you suspect a problem is occurring in a particular script or function call, set a breakpoint at the beginning of the script or at the line where the function is called.

When you close the debugger, any breakpoints you set are written to your PocketBuilder initialization file and are available when you reopen the debugger.

Setting a simple breakpoint

This procedure describes setting a breakpoint in the Source view in the debugger. You can also set a breakpoint by selecting Add Breakpoint from the pop-up menu in the Script view when you are not running the debugger.

❖ **To set a breakpoint on a line in a script:**

1 Display the script in a Source view and place the cursor where you want to set the breakpoint.

For how to change the script shown in the Source view, see "Using the Source view" on page 663.

2 Double-click the line or select Add Breakpoint from the pop-up menu.

PocketBuilder sets a breakpoint and a red circle displays at the beginning of the line. If you select a line that does not contain executable code, PocketBuilder sets the breakpoint at the beginning of the next executable statement.

Setting special breakpoints

Breakpoints can be triggered when a statement has been executed a specified number of times (an occasional breakpoint), when a specified expression is true (a conditional breakpoint), or when the value of a variable changes.

You use the Edit Breakpoints dialog box to set and edit occasional and conditional breakpoints. You can also use it to set a breakpoint when the value of a variable changes. The Edit Breakpoints dialog box opens when you:

- Click the Edit Stop button on the PainterBar
- Select Breakpoints from the pop-up menu in the Source, Variables, Watch, or Breakpoints view
- Select Edit>Breakpoints from the menu bar
- Double-click a line in the Breakpoints view

Setting occasional and conditional breakpoints

If you want to check the progress of a loop without interrupting execution in every iteration, you can set an occasional breakpoint that is triggered only after a specified number of iterations. To specify that execution stops only when conditions you specify are met, set a conditional breakpoint. You can also set both occasional and conditional breakpoints at the same location.

- **If you specify an occurrence** Each time PocketBuilder passes through the specified location, it increments a counter by one. When the counter reaches the number specified, it triggers the breakpoint and resets the counter to zero.
- **If you specify a condition** Each time PocketBuilder passes through the specified location, it evaluates the expression. When the expression is true, it triggers the breakpoint.
- **If you specify both an occurrence and a condition** Each time PocketBuilder passes through the specified location, it evaluates the expression. When the expression is true, it increments the counter. When the counter reaches the number specified, it triggers the breakpoint and resets the counter to zero.

  For example, if you specify an occurrence of 3 and the condition `notisNull(val)`, PocketBuilder checks whether val is NULL each time the statement is reached. The breakpoint is triggered on the third occurrence of a non-NULL val, then again on the sixth occurrence, and so forth.

❖ **To set an occasional or conditional breakpoint:**

1 On the Location tab in the Edit Breakpoints dialog box, specify the script and line number where you want the breakpoint.

  You can select an existing location or select New to enter a new location.

Set a simple breakpoint first
You must specify a line that contains executable code. Set a simple breakpoint on the line before opening the Edit Breakpoints dialog box to make sure the format and line number are correct.

2 Specify an integer occurrence, a condition, or both.

The condition must be a valid boolean PowerScript expression. If it is not, the breakpoint is always triggered. PocketBuilder displays the breakpoint expression in the Edit Breakpoints dialog box and in the Breakpoints view. When PocketBuilder reaches the location where the breakpoint is set, it evaluates the breakpoint expression and triggers the breakpoint only when the expression is true.

Setting a breakpoint when a variable changes

You can interrupt execution every time the value of a variable changes. The variable must be in scope when you set the breakpoint.

❖ **To set a breakpoint when a variable changes:**

- Do one of the following:
  - Select the variable in the Variables view or Watch view and select Break on Change from the pop-up menu.
  - Drag the variable from the Variables view or Watch view to the Breakpoints view.
  - Select New on the Variable tab in the Edit Breakpoints dialog box and specify the name of a variable in the Variable box

The new breakpoint displays in the Breakpoints view and in the Edit Breakpoints dialog box if it is open. PocketBuilder watches the variable during execution and interrupts execution when the value of the variable changes.

Disabling and clearing breakpoints

If you want to bypass a breakpoint for a specific debugging session, you can disable it and then enable it again later. If you no longer need a breakpoint, you can clear it.

❖ **To disable a breakpoint:**

- Do one of the following:
  - Click the red circle next to the breakpoint in the Breakpoints view or Edit Breakpoints dialog box

- Select Disable Breakpoint from the pop-up menu in the Source view
- Select Disable from the pop-up menu in the Breakpoints view

The red circle next to the breakpoint is replaced with a white circle.

You can enable a disabled breakpoint from the pop-up menus or by clicking the white circle.

Disabling all breakpoints
To disable all breakpoints, select Disable All from the pop-up menu in the Breakpoints view.

❖ **To clear a breakpoint:**

- Do one of the following:
  - Double-click the line containing the breakpoint in the Source view
  - Select Clear Breakpoint from the pop-up menu in the Source view
  - Select Clear from the pop-up menu in the Breakpoints view
  - Select the breakpoint in the Edit Breakpoints dialog box and select Clear

The red circle next to the breakpoint disappears.

Clearing all breakpoints
To clear all breakpoints, select Clear All in the Edit Breakpoints dialog box or from the pop-up menu in the Breakpoints view.

Running in debug mode

After you have set some breakpoints, you can run the application in debug mode. The application executes normally until it reaches a statement containing a breakpoint. At this point it stops so that you can examine the application. After you do so, you can single-step through the application, continue execution until execution reaches another breakpoint, or stop the debugging run so that you can close the debugger and change the code.

❖ **To run an application in debug mode:**

1 If necessary, open the debugger by clicking the Debug or the Select and Debug button.

The Debug button opens the debugger for the target you last ran or debugged. Use the Select and Debug button if you want to debug a different target in the workspace.

2 Click the Start button in the PainterBar or select Debug>Start from the menu bar.

The application starts and runs until it reaches a breakpoint. PocketBuilder displays the debugger, with the line containing the breakpoint displayed in the Source view. The yellow arrow cursor indicates that this line contains the next statement to be executed. You can now examine the application using debugger views and tools.

For more information, see "Examining an application at a breakpoint" next and "Stepping through an application" on page 665.

❖ **To continue execution from a breakpoint:**

- Click the Continue button in the PainterBar or select Debug>Continue from the menu bar

Execution begins at the statement indicated by the yellow arrow cursor and continues until the next breakpoint is hit or until the application terminates normally.

❖ **To terminate a debugging run at a breakpoint:**

- Click the Stop Debugging button in the PainterBar or select Debug>Stop from the menu bar

PocketBuilder resets the application and all the debugger views to their state at the beginning of the debugging run. You can now begin another run in debug mode or close the debugger.

Cleaning up
When you terminate a debugging run or close the debugger without terminating the run, PocketBuilder executes the application's close event and destroys any objects, such as autoinstantiated local variables, that it would have destroyed if the application had continued to run and exited normally.

Examining an application at a breakpoint

When an application is suspended at a breakpoint, use the Variables, Watch, Call Stack, and Objects in Memory views to examine its state.

About icons used in debugging views
The Variables, Watch, and Objects in Memory views use many of the icons used in the PocketBuilder Browser as well as some additional icons:
I represents an Instance; F, a field; A, an array; and E, an expression.

Examining variable values

Using Variables views

Each Variables view shows one or more types of variables in an expandable outline. Double-click the variable names or click on the plus and minus signs next to them to expand and contract the hierarchy. If you open a new Variables view, it shows all variable types.

Table 25-3: Variable views in the debugger

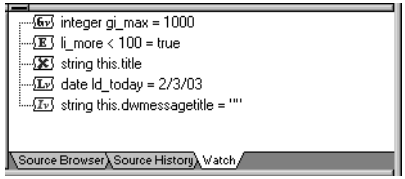
Variable type	What the Variables view shows
Local	Values of variables that are local to the current script or function
Global	Values of all global variables defined for the application and properties of all objects (such as windows) that are open
Instance	Properties of the current object instance (the object to which the current script belongs) and values of instance variables defined for the current object
Parent	Properties of the parent of the current instance
Shared	Objects, such as application, window, and menu objects, that have been opened, and the shared variables associated with them

Watching variables and expressions

The Watch view lets you monitor the values of selected variables and expressions as the application runs.

If the variable or expression is in scope, the Watch view shows its value. Empty quotes indicate that the variable is in scope but has not been initialized. An X in the Watch view indicates that the variable or expression is not in scope.

Figure 25-2: Watch view showing an out-of-scope variable



Setting variables and expressions in the Watch view

You can select variables you want to watch as the application runs by copying them from a Variables view. You can also set a watch on any PowerScript expression. When you close the debugger, any watch variables and expressions you set are saved.

❖ **To add a variable to the Watch view:**

1 Select the variable in the Variables view.

2 Do one of the following:

- Drag the variable to the Watch view
- Click the Add Watch button on the PainterBar
- Select Debug>Add Watch from the menu bar

PocketBuilder adds the variable to the watch list.

❖ **To add an expression to the Watch view:**

1 Select Insert from the pop-up menu.

2 Type any valid PowerScript expression in the New Expression dialog box and click OK.

PocketBuilder adds the expression to the watch list.

❖ **To edit a variable in the Watch view:**

1 Select the variable you want to edit.

2 Double-click the variable, or select Edit Variable from the pop-up menu.

3 Type the new value for the variable in the Modify Variable dialog box and click OK.

❖ **To edit an expression in the Watch view:**

1 Select the expression you want to edit.

2 Double-click the expression, or select Edit Expression from the pop-up menu.

3 Type the new expression in the Edit Expression dialog box and click OK.

❖ **To clear variables and expressions from the Watch view:**

1 Select the variable or expression you want to delete.

2 Do one of the following:

- Select Clear from the pop-up menu
- Click the Remove Watch button on the PainterBar
- Select Debug>Remove Watch from the menu bar

❖ **To clear all variables and expressions from the Watch view:**

- Select Clear All from the pop-up menu

Monitoring the call stack

The Call Stack view shows the sequence of function calls leading up to the script or function that was executing at the time of a pause at a breakpoint. Each line in the Call Stack view displays the name of the script and the line number from which the call was made. The yellow arrow shows the script and line where execution was suspended.

You can examine the context of the application at any line in the call stack.

❖ **To show a different context from the Call Stack view:**

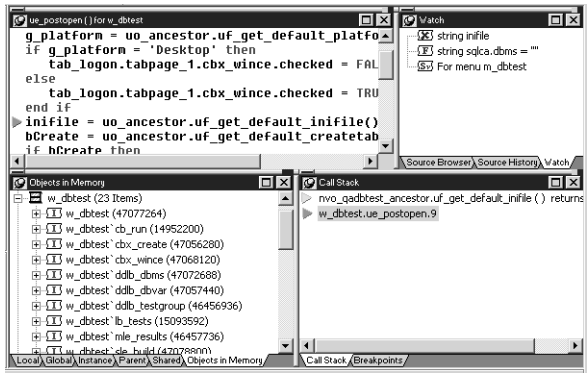
1 Select a line in the Call Stack view.

2 Do one of the following:

- Double-click the line
- Select Set Context from the pop-up menu
- Drag the line into the Source view

A green arrow indicates the script that you selected. The Source view shows the script and line number you selected, and the Variables and Watch views show the variables and expressions in scope in that context.

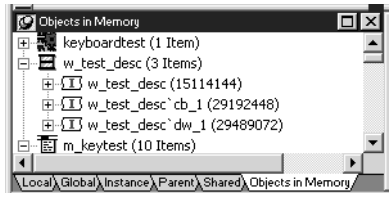
Figure 25-3: Showing a different context from the Call Stack view



Examining objects in memory

The Objects in Memory view shows an expandable list of objects currently in memory as well as the memory location of each object. Double-click the name of an object or click the plus sign next to it to view the property values of each object.

Figure 25-4: Objects in Memory view



Using the Source view

The Source view displays the full text of a script. As you run or step through the application, the Source view is updated to show the current script with a yellow arrow indicating the next statement to be executed.

Multiple Source views

You can open more than one source view. If there are multiple source views open, only the first one opened is updated to show the current script when the context of an application changes.

Copying from the Source view

When text is selected in the Source view, you can select Copy from the pop-up menu in the Source view to copy the string to the clipboard. You can then paste the string into another dialog box to search for the string, insert a watch, or add a conditional breakpoint.

Changing the Source view

From the pop-up menu, you can navigate backward and forward through the scripts that have been opened so far, open ancestor and dependent scripts, and go to a specific line in the current script. There are several other ways to change the script from other views or from the menu bar.

❖ **To change the script displayed in a Source view:**

- Do one of the following:
  - Drag the name of a script to the Source view from the Call Stack, Source Browser, or Source History views
  - Select a line and then select Open Source from the pop-up menu in the Breakpoints, Source Browser, or Source History views
  - Select Edit>Select Script from the menu bar

❖ **To find a specified string in the Source view:**

1. Select Find from the pop-up menu, or select Edit>Find from the menu bar.

   The Find Text dialog box opens.

2. Type the string in the Find box and check the search options you want.

Using the Source Browser view

The Source Browser shows all the objects in your application in an expandable hierarchy. It provides a view of the structure of the application and a quick way to open any script in the Source view.

❖ **To open a script from the Source Browser:**

1. Double-click the object that the script belongs to or click the plus sign next to the object to expand it.
2. Do one of the following:
   - Double-click the script
   - Select the script and select Open Source from the pop-up menu
   - Drag the script onto a Source view

   When you double-click or select Open Source, a new Source view opens if there was none open. If several Source views are open, the script displays in the view that was used last.

Using the Source History view

The Source History view lists all the scripts that have been opened in the current debugging session. Use the same techniques as in the Source Browser view to display a selected script in the Source view.

Source History limit
The Source History view shows up to 100 scripts and is not cleared at the end of each debugging run. It is cleared when you close the debugger. You can also clear the list from its pop-up menu.

Stepping through an application

When you have stopped execution at a breakpoint, you can use several commands to step through your application code and use the views to examine the effect of each statement. As you step through the code, the debugger views change to reflect the current context of your application and a yellow arrow cursor indicates the next statement to be executed.

Updating the Source view
When the context of your application changes to another script, the Source view is updated with the new context. If you have multiple Source views open, only the first one opened is updated.

Single-stepping through an application

You can use either Step In or Step Over to step through an application one statement at a time. They have the same result except when the next statement contains a call to a function. Use Step In if you want to step into a function and examine the effects of each statement in the function. Use Step Over to execute the function as a single statement.

❖ **To step through an application entering functions:**

- Click the Step In button in the PainterBar, or select Debug>Step In from the menu bar

❖ **To step through an application without entering functions:**

- Click the Step Over button in the PainterBar, or select Debug>Step Over from the menu bar

Stepping out of a function

If you step into a function where you do not need to step into each statement, use Step Out to continue execution until the function returns.

❖ **To step out of a function:**

- Click the Step Out button in the PainterBar, or select Debug>Step Out from the menu bar

Stepping through multiple statements

As you step through an application, you might reach sections of code that you are not interested in examining closely. The code might contain a large loop, or it might be well-tested code that you are confident is free of errors. You can use Run To Cursor to select a statement further down in a script or in a subsequent script where you want execution to stop.

❖ **To step through multiple statements:**

1. Click on the line in the script where you want to resume single stepping.
2. Click the Run To Cursor button in the PainterBar, or select Debug>Run To Cursor from the menu bar.

   PocketBuilder executes all intermediate statements and the yellow arrow cursor displays at the line where you set the cursor.

Bypassing statements

You can use Set Next Statement to bypass a section of code that contains a bug, or to test part of an application using specific values for variables. Execution continues from the statement where you place the cursor. Be cautious when you use Set Next Statement, because results might be unpredictable if, for example, you skip code that initializes a variable.

❖ **To set the next statement to be executed:**

1. Click on the line in the script where you want to continue execution.
2. Click the Set Next Statement button in the PainterBar, or select Debug>Set Next Statement from the menu bar.
3. If necessary, change the values of variables.
4. Continue execution using Continue, Step In, or Step Over.

   If you select Continue, PocketBuilder begins execution at the statement you specified and continues to the next breakpoint. If you select Step In or Step Over, PocketBuilder sets the next statement and displays the yellow arrow cursor at the line where you set the cursor.

Changing a variable's value

As you step through the application, you can change the values of variables that are in scope. You might want to do this to examine different flows through the application, to simulate a condition that is difficult to reach in normal testing, or if you are skipping code that sets a variable's value.

Limitations
You cannot change the values of enumerated variables.

❖ **To change the value of a variable:**

1 Select the variable in the Variables view or the Watch view.

2 From the pop-up menu, select Edit Variable.

3 Type a value for the variable or select the Null check box and click OK.

The value you enter must conform to the type of the variable. If the variable is a string, do not enclose the string in quotes. When you continue execution, the new value is used.

Fixing your code

If you find an error in a script or function during a debugging session, you must close the debugger before you fix it. After you have fixed the problem, you can reopen the debugger and run the application again in debug mode. The breakpoints and watchpoints set in your last session are still defined.

Debugging windows opened as local variables

One way to open a window is by declaring a local variable of type window and opening it through a string. For example:

```
window mywin
string named_window
named_window = sle_1.Text
Open(mywin, named_window)
```

The problem

Normally, you cannot debug windows opened this way after the script ends because the local variable (mywin in the preceding script) goes out of scope when the script ends.

The solution

If you want to debug windows opened this way, you can declare a global variable of type window and assign it the local variable. If, for example, you declare GlobalWindow as a global variable of type window, you could add the following line to the end of the preceding script:

```
GlobalWindow = mywin
```

You can look at and modify the opened window through the global variable. When you have finished debugging the window, you can remove the global variable and the statement assigning the local to the global.

Just-in-time debugging

If you are running your application in test mode (using the Run button) and you notice that the application is behaving incorrectly, just-in-time debugging lets you switch to debug mode without terminating the application.

When you open the debugger while running an application on the desktop, the application does not stop executing. The Source, Variables, Call Stack, and Objects in Memory views are all empty because the debugger does not have any context. To suspend execution and examine the context in a problem area, open an appropriate script and set breakpoints, then initiate the action that calls the script.

If just-in-time debugging is enabled and a system error occurs while an application is running in test mode, the debugger opens automatically, showing the context where the error occurred.

You can also use the DebugBreak function to break into the debugger.

You must enable just-in-time debugging before you run your application to take advantage of this feature.

❖ **To enable just-in-time debugging:**

1 Select Tools>System Options.

2 Check the Just In Time Debugging check box and click OK.

❖ **To debug an application while running in test mode:**

1 Enable just-in-time debugging.

2 Run the application on the desktop.

3 Click the minimized PocketBuilder icon on the Windows Taskbar.

4 Click the Debug button in the dialog box that displays.

5 Open a script in the Source view and set breakpoints.

The application is suspended when it hits a breakpoint and the Source, Variable, Call Stack, and Objects in Memory views show the current context. You can now debug the application.

Generating a trace file without timing information

If you want to generate an activity log with no timing information in a text file, you can turn on PKDebug tracing in the System Options dialog box. The PKDebug trace file contains a log showing which object functions and instructions and system DLL functions were executed in chronological order.

❖ **To generate a simple trace file:**

1 Select Tools>System Options and check the Enable PKDebug Tracing check box.

2 Check the Prompt Before OverWriting PKDebug Output File box if you want to retain existing trace output when you run or debug the application.

3 (Optional) Specify a pathname for the PKDebug output file.

4 Run your application.

If you do not check the Prompt Before OverWriting PKDebug Output File box, PocketBuilder overwrites the existing trace file every time you run the application or click the Start button in the debugger. If you check the box, it displays a response window. You can choose to overwrite the file, append new trace output to the existing file, or cancel the run or debug session.

If you want to retain the trace file, save it with a different name before running the application, or specify a new file name in the System Options dialog box.

If you do not specify an output file path, PocketBuilder creates an output file in the same directory as the PocketBuilder executable file. The output file has the same name as the PocketBuilder executable with the extension *DBG*. If you do not have write permission to this directory, you must specify a value for the output file path.

Turning PKDebug off
Running your application with PKDebug on will slow down execution. Be sure to clear the Enable PKDebug Tracing check box on the System Options dialog box if you do not need this trace information.

Testing an application on the desktop

In addition to debugging an application, you can run it on the desktop in test mode. In test mode, the application responds to user interaction and continues to run until you exit the application or a runtime error occurs. You can rely on the default runtime error reporting by PocketBuilder or write a script that specifies your own error processing.

You can also generate a diagnostic trace of your application's execution. For how to analyze your application's logic and performance, see Chapter 26, "Tracing and Profiling Applications."

Running the application on the desktop

❖ **To run an application on the desktop:**

- Do one of the following:
  - In the System Tree, highlight a target and select Run from the pop-up menu
  - Click the Run or the Select and Run button on the PowerBar
  - Select Run>Run or Run>Select and Run from the menu bar

  The Run button runs the last run or debugged target. The name of the current target is displayed in the Run button tool tip. The Select and Run button opens a dialog box that lets you select the target to run.

  PocketBuilder becomes minimized and a button displays on the Taskbar. Your application executes.

❖ **To stop a running application:**

- End the application normally, or double-click the minimized PocketBuilder icon to open a response window from which you can terminate the application.

Handling errors during execution

A serious error during execution (such as attempting to access a window that has not been opened) will trigger the SystemError event in the Application object if you have not added exception handling code to take care of the error.

If there is no SystemError script

If you do not write a SystemError script to handle these errors, PocketBuilder displays a message box containing the following information:

- The number and text of the error message
- The line number, event, and object in which the error occurred

There is also an OK button that closes the message box and stops the application.

If there is a SystemError script

If there is a script for the SystemError event, PocketBuilder executes the script and does not display the message box. Whether or not you have added TRY/CATCH blocks to your code to trap errors, it is a good idea to build an application-level script for the SystemError event to trap and process any runtime errors that have not been handled, as described in "Using the Error object" next.

For more information about handling exceptions, see the *Resource Guide*.

Using the Error object

In the script for the SystemError event, you can access the built-in Error object to determine which error occurred and where it occurred. The Error object contains the properties shown in Table 25-4.

Table 25-4: Properties of the Error object

Property	Data type	Description
Number	Integer	Identifies the error.
Text	String	Contains the text of the error message.
WindowMenu	String	Contains the name of the window or menu in which the error occurred.
Object	String	Contains the name of the object in which the error occurred. If the error occurred in a window or menu, the Object property will be the same as the WindowMenu property.
ObjectEvent	String	Contains the event for which the error occurred.
Line	Integer	Identifies the line in the script at which the error occurred.

Defining your own Error object

You can customize your own version of the Error object by defining a class user object inherited from the built-in Error object. You can add properties and define object-level functions for your Error object to allow for additional processing. In the Application painter, you can then specify that you want to use your user object inherited from Error as the global Error object in your application. For more information, see "Building a standard class user object" on page 315.

Runtime error numbers

Table 25-5 lists the runtime error numbers returned in the Number property of the Error object and the meaning of each number.

Table 25-5: Runtime errors

Number	Meaning
1	Divide by zero.
2	Null object reference.
3	Array boundary exceeded.
4	Enumerated value is out of range for function.
5	Negative value encountered in function.
6	Invalid DataWindow row/column specified.
7	Unresolvable external when linking reference.
8	Reference of array with null subscript.
9	DLL function not found in current application.
10	Unsupported argument type in DLL function.
11	Object file is out of date and must be converted to current version.
12	DataWindow column type does not match GetItem type.
13	Unresolved property reference.
14	Error opening DLL library for external function.
15	Error calling external function *name*.
16	Maximum string size exceeded.
17	DataWindow referenced in DataWindow object does not exist.
18	Function does not return value.
19	Cannot convert *name* in Any variable to *name*.
20	Database command has not been successfully prepared.
21	Bad runtime function reference.
22	Unknown object type.
23	Cannot assign object of type *name* to variable of type *name*.
24	Function call does not match its definition.
25	Double or Real expression has overflowed.

Number	Meaning
26	Field *name* assignment not supported.
27	Cannot take a negative to a noninteger power.
29	Nonarray expected in ANY variable.
30	External object does not support data type *name.*
31	External object data type *name* not supported.
32	Name not found calling external object function *name*.
33	Invalid parameter type calling external object function *name.*
34	Incorrect number of parameters calling external object function *name*.
35	Error calling external object function *name.*
36	Name not found accessing external object property *name.*
37	Type mismatch accessing external object property *name.*
38	Incorrect number of subscripts accessing external object property *name.*
39	Error accessing external object property *name*.
40	Mismatched ANY data types in expression.
41	Illegal ANY data type in expression.
42	Specified argument type differs from required argument type at runtime in DLL function *name.*
43	Parent object does not exist.
44	Function has conflicting argument or return type in ancestor.
45	Internal table overflow; maximum number of objects exceeded.
46	Null object reference cannot be assigned or passed to a variable of this type.
47	Array expected in ANY variable.
48	Size mismatch in array to object conversion.
49	Type mismatch in array to object conversion.
51	Bad argument list for function/event.
58	Object instance does not exist.
59	Invalid column range.
60	Invalid row range.
61	Invalid conversion of *number* dimensional array to object.
62	Server busy.
63	Function/event with no return value used in expression.
64	Object array expected in left side of assignment.
65	Dynamic function not found. Possible causes include: pass by value/reference mismatch.
66	Invalid subscript for array index operation.

Number	Meaning
67	NULL object reference cannot be assigned or passed to an autoinstantiate.
68	NULL object reference cannot be passed to external DLL function name.
69	Function *name* cannot be called from a secured runtime session.
70	External DLL function *name* cannot be called from a secured runtime session.
71	General protection fault occurred.
72	*name* failed with an operating system error code of *number.*
73	Reference parameters cannot be passed to an asynchronous shared/remote object method.
74	Reference parameters cannot be passed to a shared object method.
76	Passing NULL as a parameter to external function *name*.
77	Object passed to shared/remote object method is not a nonvisual user object.
79	The argument to *name* must be an array.
81	Function argument file creator must be a four character string.
82	Function argument file type must be a four character string.
83	Attempt to invoke a function or event that is not accessible.
84	Wrong *number* of arguments passed to function/event call.
85	Error in reference argument passed in function/event call.
86	Ambiguous function/event reference.
88	Cannot ask for ClassDefinition Information on open painter: *name*.
90	Cannot assign array of type *name* to variable of type array of *name.*
91	Cannot convert *name* in Any variable to *name*. Possible cause uninitialized value.
92	Required property name is missing.
96	Exception Thrown has not been handled.
97	Cannot save *name* because of a circular reference problem.
98	Obsolete object reference.

Some errors terminate the application immediately. They do not trigger the SystemError event.

SystemError event scripts

A typical script for the SystemError event includes a CHOOSE CASE control structure to handle specific errors. To stop the application, include a HALT statement in the SystemError script.

Caution
You can continue your application after a SystemError event, but doing so can cause unpredictable and undesirable effects. Where the application will resume depends on what caused the error. Typically, you are better off reporting the problem to the user, then stopping the application with HALT.

❖ **To test the SystemError event script:**

1 Assign values to the properties of the Error object with the PopulateError function.

2 Call the SignalError function to trigger the SystemError event.

The script for the SystemError event executes.

CHAPTER 26

Tracing and Profiling Applications

About this chapter

This chapter describes how to generate trace information that you can use to improve your application's performance.

Contents

About tracing and profiling an application

You use tracing and profiling to debug and tune an application. When you run an application on the desktop or on a handheld device, you can generate an execution trace file. You use the trace file to create a profile of your application.

The profile shows you which functions and events were called by which other functions and events, how often they were called, when garbage collection occurred, when objects were created and destroyed, and how long each activity took to complete. This information helps you find errors in the application's logic and identify areas that you should rewrite to improve performance.

PKDebug tracing

You can also generate a simple text trace file without timer values by checking Enable PKDebug Tracing in the System Options dialog box.

For more about PKDebug, see "Generating a trace file without timing information" on page 706.

When you can trace an application

You can create a trace file when you run an application in the design-time environment or from a standalone executable at runtime.

When you run an application with tracing turned on, PocketBuilder records a timer value in a data file every time a specific activity occurs. You control when logging begins and ends and which activities are recorded.

Creating profiles

After you have generated a trace file, you can create several different profiles or views of the application by extracting different types of information from the trace file.

PocketBuilder provides three profiling tools that create profiles (views) of the application for you, but you can also create your own analysis tools.

Using profiling to tune an application

Examining the profiles generated by the profiling tools tells you where the application is spending the most time. You can also find routines that are being called too often, routines being called that you did not expect to call, or routines that are not being called at all. Follow these suggestions for tuning an application:

- The database connection process is often slow. Although you might not be able to speed this up, you might be able to enhance the user's perception of performance by moving the database connection process to a different place in your application.
- Use profiling to tune algorithms. Fixing algorithmic yields greater performance enhancements than changing single lines of code.
- Enhancing the efficiency of a function is not as effective as removing unneeded calls to that function.
- Focus on optimizing the routines that are called most often.
- If you cannot speed up a routine, consider adding some user feedback such as a progress bar.

Collecting trace information

There are three ways to collect trace information. You can use:

- The Profiling tab on the System Options dialog box
- A window similar to the Profiling tab
- Trace objects and functions

Use the Profiling tab if you want to trace an entire application run in the development environment. For more information, see "Tracing an entire application in PocketBuilder" on page 681.

Use a window or trace objects and functions if you want to create a trace file for selected parts of the application or the entire application, either in the development environment or when running an executable file. See "Using a window" on page 681 and "Collecting trace information using PowerScript functions" on page 687.

Collection time
The timer values in the trace file exclude the time taken to collect the trace data. Because an application can be idle (while displaying a MessageBox, for example), percentage metrics are most meaningful when you control tracing programmatically to minimize idle time. Percentages are less meaningful when you create a trace file for a complete application.

Whichever method you use, you can specify:

- The name and location of the trace file and optional labels for blocks of trace data
- The kind of timer used in the trace file
- The activities you want recorded in the trace file

Trace file names and labels

The default name of the trace file is the name of the application with the extension *PKP*. The trace file is saved in the directory where the PKL or executable file resides and overwrites any existing file of the same name. If you run several different tests on the same application, you should change the trace file name for each test.

You can also associate a label with the trace data. If you are tracing several different parts of an application in a single test run, you can associate a different label with the trace data (the trace block) for each part of the application.

Timer kinds

There are three kinds of timer: clock, process, and thread. If your analysis does not require timing information, you can omit timing information from the trace file to improve application performance.

If you do not specify a timer kind, the time at which each activity begins and ends is recorded using the clock timer, which measures an absolute time with reference to an external activity, such as the machine's start-up time. The clock timer measures time in microseconds. Depending on the speed of your machine's central processing unit, the clock timer can offer a resolution of less than one microsecond. A timer's resolution is the smallest unit of time the timer can measure.

For tracing activities on the desktop, you can use process or thread timers, which measure time in microseconds with reference to when the process or thread being executed started. Clock timers and thread timers are the only kinds of timers supported on handheld devices. If you select the process timer for an application running on a device, the thread timer is used instead.

Both process and thread timers exclude the time taken by any other running processes or threads so that they give you a more accurate measurement of how long the process or thread is taking to execute, but both have a lower resolution than the clock timer.

Trace activities

You can choose to record in the trace file the time at which any of the following activities occurs. If you are using the System Options dialog box or a window, you select the check boxes for the activities you want. If you are using PowerScript functions to collect trace information, you use the TraceActivity enumerated type to identify the activity.

Table 26-1: Trace activities

Trace Activities check box	What is recorded	TraceActivity value
Routine Entry/Exit	Routine entry or exit	ActRoutine!
Routine Line Hits	Execution of any line in any routine	ActLine!
Embedded SQL	Use of an embedded SQL verb	ActESQL!
Object Creation/ Destruction	Object creation or destruction	ActObjectCreate!, ActObjectDestroy!
User Defined Activities	A user-defined activity that records an informational message	ActUser!
System Errors	A system error or warning	ActError!
Garbage Collection	Garbage collection	ActGarbageCollect!

Trace Activities check box	What is recorded	TraceActivity value
Not available	Routine entry and exit, embedded SQL verbs, object creation and destruction, and garbage collection	ActProfile!
Not available	All except ActLine!	ActTrace!

When you begin and end tracing, an activity of type ActBegin! is automatically recorded in the trace file. User-defined activities, which you use to log informational messages to the trace file, are the only trace activities enabled by default.

Tracing an entire application in PocketBuilder

Use the Profiling tab on the System Options dialog box if you want to collect trace data for an entire application run in the PocketBuilder development environment.

❖ **To trace an entire application in PocketBuilder:**

1 Select Tools>System Options from the PowerBar and select the Profiling tab.

2 Specify a name for the trace file, select the trace options you want, and click OK.

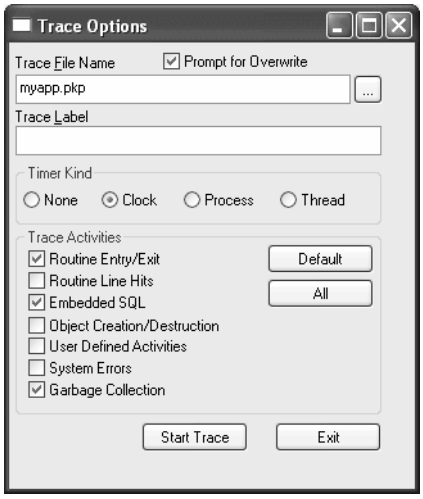
When you run the application, the activities you selected are logged in the trace file.

Using a window

You can create a window that is similar to the Profiling tab on the System Options dialog box and add it to any application that is under development, so that you can start and stop tracing when testing specific actions.

This window lets you specify a trace file name, label, and timer kind, as well as which activities you want to trace:

Figure 26-1: Window for selecting tracing and profiling options



The following instance variables are defined for the window:

```
TimerKind itk_kind
string is_title = 'Trace Options '
string is_starttext
```

The Open event for the window sets some defaults:

```
application lapp_current
lapp_current = getapplication()
itk_kind = Clock!
is_starttext = cb_startstop.text
sle_filename.text = classname(lapp_current)+'.pkp'
```

The following code shows the script for the Clicked event of the Start Trace button. The text for the button is set to Start Trace in the painter. When the user clicks Start Trace, the button label changes to Stop Trace. The Clicked event script checks the text on the button before either starting or stopping tracing. This script uses the functions described in “Collecting trace information using PowerScript functions” on page 687:

```
errorreturn le_errorreturn
integer li_key

// Check that the button label is Start Trace
// and a trace file name has been entered
if this.text = is_starttext then
   if len(trim(sle_filename.text)) = 0 then
       messagebox(parent.title, &
```

```
      'Trace file name is required',information!)
    sle_filename.setfocus()
    return
end if

// If Prompt for overwrite is checked and the
// file exists, pop up a response window
if cbx_prompt.checked and &
   fileexists(sle_filename.text) then
   li_key = messagebox(parent.title, &
      'OK to overwrite '+sle_filename.text, &
      question!,okcancel!,1)
   if li_key = 2 then return
end if

// Open the trace file
TraceOpen( sle_filename.text, itk_kind )

// Enable tracing for checked activities
// For each activity, check for errors and close
// the trace file if an error occurs
if cbx_embeddedsql.checked then
   le_errorreturn = TraceEnableActivity( ActESql! )
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
         'TraceEnableActivity( ActESql! )')
       le_errorreturn = Traceclose()
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceClose')
      end if
      return
   end if
end if

if cbx_routineentry.checked then
   le_errorreturn =TraceEnableActivity(ActRoutine!)
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
         'TraceEnableActivity( ActRoutine! )')
      Traceclose()
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceClose')
      end if
      return
   end if
end if
```

```
if cbx_userdefined.checked then
   le_errorreturn = TraceEnableActivity( ActUser! )
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
         'TraceEnableActivity( ActUser! )')
      Traceclose()
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceClose')
      end if
      return
   end if
end if

if cbx_systemerrors.checked then
   le_errorreturn = TraceEnableActivity(ActError! )
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
         'TraceEnableActivity( ActError! )')
      Traceclose()
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceClose')
      end if
      return
   end if
end if

if cbx_routineline.checked then
   le_errorreturn = TraceEnableActivity( ActLine! )
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
         ' TraceEnableActivity( ActLine! )')
      Traceclose()
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceClose')
      end if
      return
   end if
end if
if cbx_objectcreate.checked then
   le_errorreturn = &
      TraceEnableActivity( ActObjectCreate! )
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn, &
         'TraceEnableActivity( ActObject! )')
      Traceclose()
```

```
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn,'TraceClose')
            end if
            return
         end if
         le_errorreturn = &
            TraceEnableActivity( ActObjectDestroy! )
         if le_errorreturn <> Success! then
            of_errmsg(le_errorreturn, &
              'TraceEnableActivity(ActObjectDestroy!)')
            Traceclose()
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn,'TraceClose')
            end if
            return
         end if
      end if

      if cbx_garbagecoll.checked then
         le_errorreturn = &
            TraceEnableActivity( ActGarbageCollect! )
         if le_errorreturn <> Success! then
            of_errmsg(le_errorreturn, &
             'TraceEnableActivity(ActGarbageCollect! )')
            Traceclose()
            if le_errorreturn <> Success! then
               of_errmsg(le_errorreturn,'TraceClose')
            end if
            return
         end if
      end if

      // Start tracing
      le_errorreturn =TraceBegin( sle_tracelabel.text )
      if le_errorreturn <> Success! then
         of_errmsg(le_errorreturn,'TraceBegin')
         return
      end if

      // Change the title of the window and the
      // text of the Start Trace button
      parent.title = is_title + '(Tracing)'
      this.text = 'Stop &Tracing'
   // If the button label is Stop Trace, stop tracing
   // and close the trace file
   else
```

```
   le_errorreturn =TraceEnd()
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn,'TraceEnd')
      return
   end if

   le_errorreturn =TraceClose()
   if le_errorreturn <> Success! then
      of_errmsg(le_errorreturn,'TraceClose')
   end if
   this.text = is_starttext
   parent.title = is_title
end if
```

The window uses two functions to handle error messages: of_errmsg and of_converterror.

of_errmsg function

The of_errmsg function displays a message box:

```
// of_errmsg
Messagebox( this.title,'Error executing '+ as_msg + &
   '. Error code : '+ of_converterror(ae_error) )
```

of_converterror function

The of_converterror function converts the ErrorReturn parameter to a string:

```
// of_converterror: convert enumerated type
// ErrorReturn parameter to text.
String ls_result

choose case a_error
   Case Success!
      ls_result =  "Success!"
   Case FileCloseError!
      ls_result =  "FileCloseError!"
   Case FileOpenError!
      ls_result =  "FileOpenError!"
   Case FileReadError!
      ls_result =  "FileReadError!"
   Case FileWriteError!
      ls_result =  "FileWriteError!"
   Case FileNotOpenError!
      ls_result =  "FileNotOpenError!"
   Case FileAlreadyOpenError!
      ls_result =  "FileAlreadyOpenError!"
   Case FileInvalidFormatError!
      ls_result =  "FileInvalidFormatError!"
   Case FileNotSetError!
      ls_result =  "FileNotSetError!"
```

```
    Case EventNotExistError!
       ls_result =  "EventNotExistError!"
    Case EventWrongPrototypeError!
       ls_result =  "EventWrongPrototypeError!"
    Case ModelNotExistsError!
       ls_result =  "ModelNotExistsError!"
    Case ModelExistsError!
       ls_result =  "ModelExistsError!"
    Case TraceStartedError!
       ls_result =  "TraceStartedError!"
    Case TraceNotStartedError!
       ls_result =  "TraceNotStartedError!"
    Case TraceNoMoreNodes!
       ls_result =  "TraceNoMoreNodes!"
    Case TraceGeneralError!
       ls_result =  "TraceGeneralError!"
    Case FeatureNotSupportedError!
       ls_result =  "FeatureNotSupportedError!"
    Case else
       ls_result =  "Unknown Error Code"
end choose
return ls_result
```

Collecting trace information using PowerScript functions

You use the PowerScript system functions listed in Table 26-2 to collect information in a trace file. Each of these functions returns a value of type ErrorReturn, an enumerated datatype.

Table 26-2: PowerScript trace functions

Use this PowerScript function	To do this
TraceOpen	Open a named trace file and set the timer kind.
TraceEnableActivity	Enable logging of the specified activity.
TraceBegin	Start logging all enabled activities. You can pass an optional label for the trace block.
TraceError	Log a severity level and error message to the trace file.
TraceUser	Log a reference number and informational message to the trace file.
TraceEnd	Stop logging all enabled activities.
TraceDisableActivity	Disable logging of the specified activity.
TraceClose	Close the open trace file.

In general, you call the functions in the order shown in the table. That is, you must call TraceOpen before you call any other trace functions. You call TraceClose when you have finished tracing.

TraceEnableActivity and TraceDisableActivity can be called only when a trace file is open but tracing has not begun or has stopped—that is, before you call TraceBegin or after you call TraceEnd.

TraceUser and TraceError can be called only when the trace file is open and tracing is active—that is, after you call TraceBegin and before you call TraceEnd.

About TraceUser and TraceError

You can use TraceUser to record specific events in the trace file, such as the beginning and end of a body of code. You can also record the execution of a statement you never expected to reach, such as the DEFAULT statement in a CHOOSE CASE block. TraceError works just like TraceUser, but you can use it to signal more severe problems.

Both TraceUser and TraceError take a number and text string as arguments. You can use a simple text string that states what activity occurred, or you can build a string that provides more diagnostic information by including some context, such as the current values of variables. Run the application with only ActUser! or ActError! tracing turned on and then use the Profiling Trace View to pinpoint problems quickly.

Example: trace data collection

In this example, the user selects a timer kind from a drop-down list and enters a name for the trace file in a single-line edit box. Typically you would use the ErrorReturn return value from every trace call to return an error message if the call fails. For brevity, the example shows this only for the TraceOpen call.

Several trace activities are disabled for a second trace block. The activities that are *not* specifically disabled remain enabled until TraceClose is called.

```
ErrorReturn le_err
integer li_key
TimerKind ltk_kind

CHOOSE CASE ddlb_timerkind.Text
   CASE "None"
      ltk_kind = TimerNone!
   CASE "Clock"
      ltk_kind = Clock!
   CASE "Process"
      ltk_kind = Process!
   CASE "Thread"
      ltk_kind = Thread!
END CHOOSE
```

```
// Open the trace file and return an error message
// if the open fails
le_err = TraceOpen( sle_fileName.Text, ltk_kind )
IF le_err <> Success! THEN &
   of_errmsg(le_err, 'TraceOpen failed')
   RETURN
END IF

// Enable trace activities. Enabling ActLine!
// enables ActRoutine! implicitly
TraceEnableActivity(ActESQL!)
TraceEnableActivity(ActUser!)
TraceEnableActivity(ActError!)
TraceEnableActivity(ActLine!)
TraceEnableActivity(ActObjectCreate!)
TraceEnableActivity(ActObjectDestroy!)
TraceEnableActivity(ActGarbageCollect!)

TraceBegin("Trace_block_1")
// first block of code to be traced
// this block has the label Trace_block_1
…

TraceEnd()

// disable trace activities not needed for
// second block
TraceDisableActivity(ActLine! )
TraceDisableActivity(ActObjectCreate!)
TraceDisableActivity(ActObjectDestroy!)
TraceDisableActivity(ActGarbageCollect!)

TraceBegin("Trace_block_2")
// second block of code to be traced
…

TraceEnd()
TraceClose()
```

Analyzing trace information using profiling tools

After you have created a trace file, the easiest way to analyze it is to use the profiling tools provided on the Tool tab of the New dialog box. There are three tools:

- The Profiling Class View shows information about the objects that were used in the application
- The Profiling Routine View shows information about all the routines (functions and events) that were used in the application
- The Profiling Trace View shows the elapsed time taken by each activity in chronological order

Using the profiling tools as a model

Even if you want to develop your own analysis tools, using the profiling tools is a good way to learn about these models and about profiling in PocketBuilder. When you are ready to develop your own tools, see "Analyzing trace information programmatically" on page 696 for an overview of the approaches you can take.

Viewing profiles from a handheld device

The profiling tools included with PocketBuilder work only on the desktop. If you use the profiling tools to view a trace file from a deployment device, you must copy the executable for the application that created the trace file to a desktop directory. The desktop directory must have the same path name—with the exception of the initial drive letter—as the path where the executable resides on the device. For example, for an application residing in the \Program Files\Tests directory on a device, you must copy the executable to \Program Files\Tests on the current drive on the desktop.

You must also copy the PKP trace file from the handheld device. Although you can copy the PKP file to any location on the desktop, when you select the PKP file in a profiling tool you also set the current drive; therefore you should copy the PKP file and the executable file to the same drive on the desktop.

Profiling Class View

The Class view uses a TreeView control to display statistics for PocketBuilder objects, their functions, and their events. It displays statistics only for those objects that were active while tracing was enabled. The Class view has three tabs:

- **Numbers** Shows statistics only
- **Source** Shows statistics and source code for those routines that originated in PowerScript source
- **Graph** Shows statistics in a bar graph

For each object, the Class view shows all the routines called from each class with the number of times each routine was called (hit) as well as timing information for each call. Embedded SQL commands are shown as being called from a pseudo class called ESQL.

The Class view includes both PocketBuilder system-level objects (such as DataWindow and SystemFunction) and user-defined classes (such as windows and user objects). Each top-level node is a PocketBuilder class. As you expand the TreeView control, each node represents a routine and each subnode represents a called routine.

The Class view uses the call graph model to show cumulative statistics for objects, routines, and called routines. The information displayed on the right side of the display differs depending on the current node on the left.

Table 26-3: Statistics displayed in the Profiling Class View by node

Current node	Statistics displayed
Application	Statistics for each object
Object	Cumulative statistics for the object and detailed statistics for the object's routines
Routine	Cumulative statistics for the routine and detailed statistics for called routines

You can sort items on the right by clicking the heading.

Class view metrics

The Class view displays five metrics. The profiling tool accesses these metrics from instances of the ProfileCall and ProfileRoutine objects. The time scale you specified in the Preferences dialog box determines how times are displayed.

Table 26-4: Metrics in the Profiling Class View

Metric	What it means
Hits	The number of times a routine executed in a particular context.
Self	The time spent in the routine or line itself. If the routine or line was executed more than once, this is the total time spent in the routine or line itself; it does not include time spent in routines called by this routine.
%Self	Self as a percentage of the total time the calling routine was active.

Metric	What it means
Self+Called	The time spent in the routine or line and in routines or lines called from the routine or line. If the routine or line was executed more than once, this is the total time spent in the routine or line and in called routines or lines.
%Self+Called	Self+Called as a percentage of the total time that tracing was enabled.

About percentages
The percentages captured in the trace file are based on the total time tracing was enabled. Because an application can be idle (while displaying a MessageBox, for example), percentage metrics are most meaningful when you control tracing programmatically, which can help to minimize idle time. Percentages are least meaningful when you create a trace file for a complete application.

Profiling Routine View

The Routine view displays statistics for a routine, its calling routines, and its called routines. It uses multiple DataWindow objects to display information for a routine:

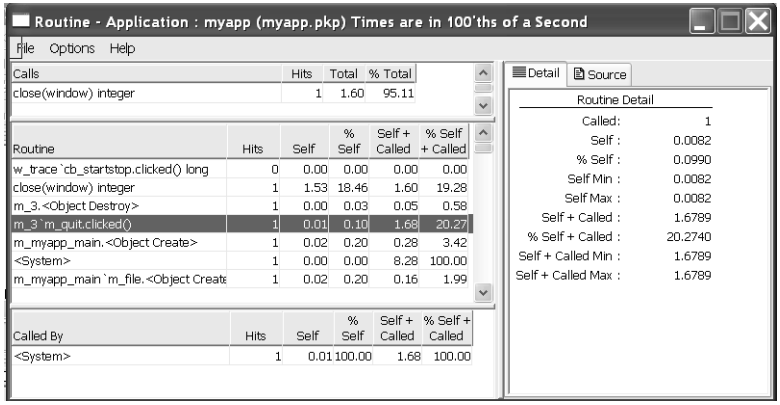
- **Called routines** The top DataWindow lists functions and events called by the current routine.
- **Current routine** The middle DataWindow and the DataWindow on the right highlight the current routine and show detailed statistics.
- **Calling routines** The bottom DataWindow lists functions and events that call the routine displayed in the middle DataWindow.

The Routine view has two tabs:

- **Detail** Shows statistics only
- **Source** Shows statistics and source code for those routines that originated in PowerScript source

The Routine view uses the call graph model to show the call chain and cumulative statistics for routines and called routines.

Figure 26-2: Profiling Routine View



You can specify the current routine by clicking in the various DataWindows.

Table 26-5: Specifying the current routine in the Profiling Routine View

To do this	Click here
Establish a new current routine in the current routine DataWindow	On the routine. The profiling tool updates the top and bottom DataWindows with information on called and calling routines.
Select a calling routine as the new routine	On the routine in the top DataWindow. The profiling tool makes it the current routine in the middle DataWindow.
Select a called routine as the new routine	On the routine in the bottom DataWindow. The profiling tool makes it the current routine in the middle DataWindow.

You can sort items by clicking the column headings.

Routine view metrics

The Routine view displays nine metrics. The profiling tool accesses these metrics from instances of the ProfileCall and ProfileRoutine objects. The time scale you specified in the Preferences dialog box determines how times are displayed.

Table 26-6: Metrics in the Profiling Routine View

Metric	What it means
Hits (Called on Detail tab)	The number of times a routine executed in a particular context.
Self	The time spent in the routine or line itself. If the routine or line was executed more than once, this is the total time spent in the routine or line itself; it does not include time spent in routines called by this routine.
%Self	Self as a percentage of the total time the calling routine was active.

Metric	What it means
Self Min	The shortest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
Self Max	The longest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
Self+Called	The time spent in the routine or line and in routines or lines called from the routine or line. If the routine or line was executed more than once, this is the total time spent in the routine or line and in called routines or lines.
%Self+Called	Self+Called as a percentage of the total time that tracing was enabled.
Self+Called Min	The shortest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.
Self+Called Max	The longest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.

Profiling Trace View

The Trace view uses a TreeView control to display the events and functions in the trace file. The initial display shows top-level routines. Each node expands to show the sequence of routine execution. The fully expanded TreeView shows the complete sequence of executed instructions for the trace file.

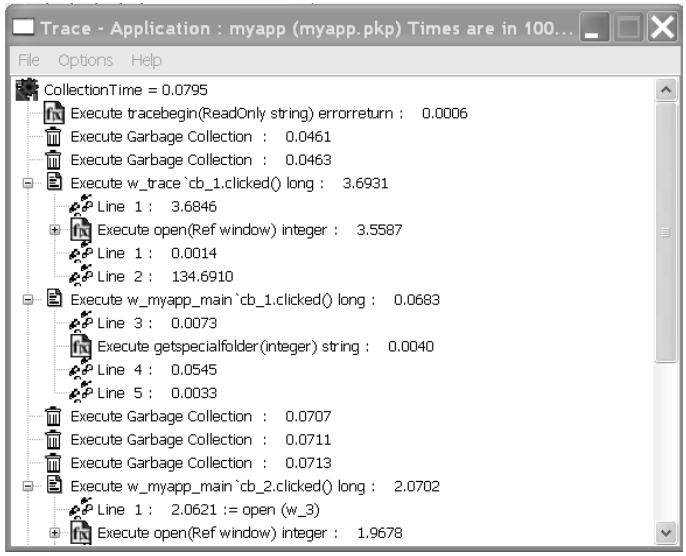
The Trace view uses the trace tree model to show the sequence of execution. It includes statistics and (for those routines that originated in PowerScript source) source code.

You can use the Trace View Options section of the Preferences dialog box to control the display:

- **System routines** This option controls whether the Trace view includes information for lines that execute PocketBuilder system routines.
- **Line information** This option controls whether the Trace view includes line numbers.

The following screen shows a Trace view with several nodes expanded. The number to the right of each item is the execution time for that item.

Figure 26-3: Profiling Trace View



Trace view metrics

The Trace view displays two metrics. The profiling tool accesses these metrics from instances of the TraceTree and TraceTreeNode objects.

Table 26-7: Metrics in the Profiling Trace View

Entry	What it means
Routine or line number	The routine or line number that was executed.
Execution time	Total execution time for the Tree view entry. This is total time from the start of the entry to the end of the entry. For example, if you call the MessageBox function, this value reflects the elapsed time from when the message box was opened until the user provided some kind of response.

About preferences

The specifications you make in the Preferences dialog box control whether the Trace view displays system functions and line numbers.

Setting call aggregation preferences

You can control how the profiling tools display information using the Preferences dialog box. To open it, select Options>Preferences from any profiling view's menu bar.

In both Class and Routine views, you can choose how functions and events that are called more than once are displayed. Select the Aggregate Calls check box if you want the view to display a single line for each called function or event that represents a sum of all the times it was called. If you do not select the check box, the view displays statistics for each call on a separate line.

For example, if aggregation is enabled and a function calls another function five times, you see one entry with five hits; with no aggregation, you see five separate entries for the same function.

Internally, the profiling tool controls aggregation by using the AggregateDuplicateRoutineCalls boolean argument to the OutgoingCallList and IncomingCallList functions on the ProfileRoutine object.

Analyzing trace information programmatically

PocketBuilder provides three ways to analyze trace information using built-in system objects and functions:

- Analyze performance by building a call graph model

  A call graph model contains information about all the routines in the trace file: how many times each routine was called, which routines called it and which routines it called, and the execution time taken by the routine itself and any routines it called.

- Analyze program structure and logical flow by building a trace tree model

  A trace tree model contains information about all recorded activities in the trace file in chronological order, with the elapsed time for each activity.

- Access the data in the trace file directly

  Trace objects and functions let you build your own model and analysis tools by giving you access to all the data in the trace file.

The profiling tools use the first two ways. The Class and Routine views are based on a call graph model, and the Trace view is based on a trace tree model. No matter which way you analyze trace files, you can analyze them on the desktop only—the profiling functions are not supported on deployment devices.

Supporting files needed
To create a profile from a trace file, PocketBuilder must also access the PKL, PKD, or executable file used to create the trace file. If you create the trace file on a handheld device, you must copy the trace file and the executable file to the desktop. The PKL and PKD files must also be available on the desktop.

Analyzing performance with a call graph model

You use the PowerScript functions and PocketBuilder objects listed in Table 26-8 to analyze the performance of an application. You can use these functions only on the desktop.

Table 26-8: Functions for analyzing performance

Use this function	With this object	To do this
SetTraceFileName	Profiling	Set the name of the trace file to be analyzed.
BuildModel	Profiling	Build a call graph model based on the trace file. You can pass optional parameters that let you track the progress of the build.
RoutineList	Profiling and ProfileClass	Get a list of routines in the model or in a class.
ClassList	Profiling	Get a list of classes in the model.
SystemRoutine	Profiling	Get the name of the routine node that represents the root of the model.
IncomingCallList	ProfileRoutine	Get a list of routines that called a specific routine.
OutgoingCallList	ProfileRoutine and ProfileLine	Get a list of routines called by a specific routine or from a specific line.

Use this function	With this object	To do this
LineList	ProfileRoutine	Get a list of lines in the routine in line order.
DestroyModel	Profiling	Destroy the current performance analysis model and all the objects associated with it.

Each of these functions returns a value of the enumerated datatype ErrorReturn. The objects contain information such as the number of times a line or routine was executed, and the amount of time spent in a line or routine and in any routines called from that line or routine.

Using the BuildModel function to build a call graph model

The call graph model that you create with the BuildModel function contains all the routines in the trace file and can take a long time to build. If you want to monitor the progress of the build or you want to be able to interrupt it while the model is being built, you can pass optional arguments to BuildModel.

BuildModel arguments

BuildModel takes three arguments: the name of an object of type PowerObject, the name of a user event, and a long value representing how often the user event should be triggered as a percentage of the build completed.

The user event returns a boolean value and has two arguments: the number of the current activity, and the total number of activities in the trace file.

Destroying existing models

Before you call BuildModel, you can call DestroyModel to clean up any objects remaining from an existing model.

Example: building a call graph model

In the following example, the user event argument to BuildModel is called ue_progress and is triggered each time five percent of the activities have been processed. The progress of the build is shown in a window called w_progress that has a cancel button.

```
Profiling lpro_model
lpro_model = CREATE Profiling
ib_cancel = FALSE
lpro_model.SetTraceFileName(is_fileName )

open(w_progress)
// call the of_init window function to initialize
// the w_progress window
w_progress.of_init(lpro_model.numberofactivities, &
   'Building Model', this, 'ue_cancel')
```

```
// Build the call graph model
lpro_model.BuildModel(this, 'ue_progress', 5)

// clicking the cancel button in w_progress
// sets ib_cancel to TRUE and
// returns FALSE to ue_progress
IF ib_cancel THEN &
   close(w_progress)
   RETURN -1
END IF
```

Extracting information from the call graph model

After you have built a call graph model of the application, you can extract detailed information from it.

For routines and lines, you can extract the timing information shown in Table 26-9 from the ProfileRoutine and ProfileLine objects.

Table 26-9: Timing information in the call graph model

Property	What it means
AbsoluteSelfTime	The time spent in the routine or line itself. If the routine or line was executed more than once, this is the total time spent in the routine or line itself.
MinSelfTime	The shortest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
MaxSelfTime	The longest time spent in the routine or line itself. If the routine or line was executed only once, this is the same as AbsoluteSelfTime.
AbsoluteTotalTime	The time spent in the routine or line and in routines or lines called from the routine or line. If the routine or line was executed more than once, this is the total time spent in the routine or line and in called routines or lines.
MinTotalTime	The shortest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.
MaxTotalTime	The longest time spent in the routine or line and in called routines or lines. If the routine or line was executed only once, this is the same as AbsoluteTotalTime.
PercentSelfTime	AbsoluteSelfTime as a percentage of the total time tracing was active.
PercentTotalTime	AbsoluteTotalTime as a percentage of the total time tracing was active.

Example: extracting information from a call graph model

The following function extracts information from a call graph model about the routines called from a specific routine. You would use similar functions to extract information about the routines that called the given routine and about the routine itself.

The function takes a ProfileCall object and an index as arguments and returns a structure containing the number of times the called routine was executed and execution times for the called routine.

```
str_func_detail lstr_result
ProfileClass lproclass_class
ProfileRoutine lprort_routine

// get the name of the called routine
// from the calledroutine property of
// the ProfileCall object passed to the function
lprort_routine = a_pcall.Calledroutine
lstr_result.Name = ""
lproclass_class = a_pcall.Class
IF isValid(lproclass_class) THEN &
   lstr_result.Name += lproclass_class.Name + "."
lstr_result.name += a_pcall.Name

lstr_result.hits = a_pcall.HitCount
lstr_result.selfTime = a_pcall. &
   AbsoluteSelfTime * timeScale
lstr_result.totalTime = a_pcall. &
   AbsoluteTotalTime * timeScale
lstr_result.percentSelf = a_pcall.PercentSelfTime
lstr_result.percentTotal= a_pcall.PercentTotalTime
lstr_result.index = al_index

RETURN lstr_result
```

Analyzing structure and flow using a trace tree model

You use the PowerScript functions and PocketBuilder objects listed in Table 26-10 to build a nested trace tree model of an application. You can use these functions only on the desktop.

Table 26-10: Functions for analyzing program structure and flow

Use this function	With this object	To do this
SetTraceFileName	TraceTree	Set the name of the trace file to be analyzed.
BuildModel	TraceTree	Build a trace tree model based on the trace file. You can pass optional parameters that let you track the progress of the build.
EntryList	TraceTree	Get a list of the top-level entries in the trace tree model.
GetChildrenList	TraceTreeRoutine, TraceTreeObject, and TraceTreeGarbageCollect	Get a list of the children of the routine or object—that is, all the routines called directly by the routine, or the destructor called as a result of the object's deletion.
DestroyModel	TraceTree	Destroy the current trace tree model and all the objects associated with it.

Each of these functions returns a value of type ErrorReturn.

Each TraceTreeNode object returned by the EntryList and GetChildrenList functions represents a single node in the trace tree model and contains information about the parent of the node and the type of activity it represents.

Inherited objects

The following objects inherit from TraceTreeNode and contain additional information, including timer values:

TraceTreeError
TraceTreeESQL
TraceTreeGarbageCollect
TraceTreeLine
TraceTreeObject
TraceTreeRoutine
TraceTreeUser

Using *BuildModel* to build a trace tree model

You use the same approach to building a trace tree model as you do to building a call graph model, except that you build a model of type TraceTree instead of type Profiling.

For example:

```
TraceTree ltct_treemodel
ltct_treemodel = CREATE TraceTree
ltct_treeModel.SetTraceFileName(is_fileName )

ltct_treeModel.BuildModel(this, 'ue_progress', 1)
```

For more about using BuildModel, see "Using the BuildModel function to build a call graph model" on page 698.

Extracting information from the trace tree model

To extract information from a tree model, you can use the EntryList function to create a list of top-level entries in the model and then loop through the list, extracting information about each node. For each node, determine its activity type using the TraceActivity enumerated datatype, and then use the appropriate TraceTree object to extract information.

Example: trace tree model

The following simple example extracts information from an existing trace tree model and stores it in a structure:

```
TraceTreeNode ltctn_list[], ltctn_node
long ll_index, ll_limit
string ls_line
str_node lstr_node

ltct_treemodel.EntryList(ltctn_list)
ll_limit = UpperBound(ltctn_list)
FOR ll_index = 1 to ll_limit
   ltctn_node = ltctn_list[ll_index]
   of_dumpnode(ltctn_node, lstr_node)
   // insert code to handle display of
   // the information in the structure here
   …
NEXT
```

The of_dumpnode function takes a TraceTreeNode object and a structure as arguments and populates the structure with information about each node. The following code shows part of the function:

```
string ls_exit, ls_label, ls_routinename
long ll_node_cnt
TraceTreeNode ltctn_list[]
errorreturn l_err

astr_node.Children = FALSE
```

```
astr_node.Label = ''
IF NOT isvalid(atctn_node) THEN RETURN
CHOOSE CASE atctn_node.ActivityType
   CASE ActRoutine!
     TraceTreeRoutine ltctrt_routin
     ltctrt_routine = atctn_node
     IF ltctrt_routine.Classname = '' THEN &
        ls_routinename = ltctrt_routine.ClassName + "."
     END IF
     ls_routinename += ltctrt_routine.Name
     ltctrt_routine.GetChildrenList(ltctn_list)
     ll_node_cnt = UpperBound(ltctn_list)

     ls_label = "Execute " + ls_routinename + ' :' + &
         space(ii_offset) + String(l_timescale * &
         (ltctrt_routine.ExitTimerValue - &
         ltctrt_routine.EnterTimerValue), '0.000000')
     astr_node.Children = (ll_node_cnt > 0)
     astr_node.Label = ls_label
     astr_node.Time = ltctrt_routine.EnterTimerValue
     RETURN
   CASE ActLine!
     TraceTreeLine tctln_treeLine
     tctln_treeLine = atctn_node
     ls_label = LINEPREFIX + &
        String(tctln_treeLine.LineNumber )
     astr_node.time = tctln_treeLine.Timervalue
     ...
     // CASE statements omitted
     ...
   CASE ELSE
     ls_label = "INVALID NODE"
   END CHOOSE

   astr_node.label = ls_label
   RETURN
```

Accessing trace data directly

You use the PowerScript functions and PocketBuilder objects listed in Table 26-11 to access the data in the trace file directly so that you can develop your own analysis tools.

Table 26-11: Functions for direct access to trace data

Use this function with the TraceFile object	To do this
Open	Opens the trace file to be analyzed.
NextActivity	Returns the next activity in the trace file. The value returned is of type TraceActivityNode.
Reset	Resets the next activity to the beginning of the trace file.
Close	Closes the open trace file.

With the exception of NextActivity, each of these functions returns a value of type ErrorReturn. Each TraceActivityNode object includes information about the category of the activity, the timer value when the activity occurred, and the activity type.

Timer values

The category of the activity is either TraceIn! or TraceOut! for activities that have separate beginning and ending points, such as routines, garbage collection, and tracing itself. Each such activity has two timer values associated with it: the time when it began and the time when it completed.

Activities that have only one associated timer value are in the TraceAtomic! category: ActLine!, ActUser!, and ActError! are all atomic activities.

Inherited objects

The following objects inherit from TraceActivityNode and contain data about the associated activity type:

TraceBeginEnd
TraceError
TraceESQL
TraceGarbageCollect
TraceLine
TraceObject
TraceRoutine
TraceUser

TraceTreeNode and TraceActivityNode objects
The objects that inherit from TraceActivityNode are analogous to those that inherit from TraceTreeNode, and you can use similar techniques when you write applications that use them.

For a list of activity types, see “Trace activities” on page 680.

Using the TraceFile object

To access the data in the trace file directly, you create a TraceFile object, open a trace file, and then use the NextActivity function to access each activity in the trace file sequentially. For each node, determine what activity type it is by examining the TraceActivity enumerated datatype, and then use the appropriate trace object to extract information.

Example: direct access to trace data

The following example creates a TraceFile object, opens a trace file called ltcf_file, and then uses a function called of_dumpActivityNode to report the appropriate information for each activity depending on its activity type.

```
string ls_fileName
TraceFile ltcf_file
TraceActivityNode ltcan_node
string ls_line

ls_fileName = sle_filename.Text
ltcf_file = CREATE TraceFile
ltcf_file.Open(ls_fileName)
ls_line = "CollectionTime = " + &
   String(Truncate(ltcf_file.CollectionTime, 6)) &
   + "~r~n" + "Number of Activities = " + &
   String(ltcf_file.NumberOfActivities) + "~r~n" + &
   "Time Stamp " + "Activity" + "~r~n"

mle_output.text = ls_line

ltcan_node = ltcf_file.NextActivity()
DO WHILE IsValid(ltcan_node)
   ls_line += of_dumpActivityNode(ltcan_node)
   ltcan_node = ltcf_file.NextActivity()
LOOP

mle_output.text = ls_line
ltcf_file.Close()
```

The following code shows part of of_dumpActivityNode:

```
string lstr_result

lstr_result = String(Truncate(atcan_node. &
   TimerValue, 6)) + " "
CHOOSE CASE atcan_node.ActivityType
   CASE ActRoutine!
      TraceRoutine ltcrt_routine
      ltcrt_routine = atcan_node
      IF ltcrt_routine.IsEvent THEN
```

```
            lstr_result += "Event: "
         ELSE
            lstr_result += "Function: "
         END IF
         lstr_result += ltcrt_routine.ClassName + "." + &
            ltcrt_routine.name + "(" + &
            ltcrt_routine.LibraryName + ") " &
            + String(ltcrt_routine.ObjectId) + "~r~n"
      CASE ActLine!
         TraceLine ltcln_line
         ltcln_line = atcan_node
         lstr_result += "Line: " +     &
            String(ltcln_line.LineNumber) + "~r~n"
      CASE ActESQL!
         TraceESQL ltcsql_esql
         ltcsql_esql = atcan_node
         lstr_result += "ESQL: " + ltcsql_esql.Name &
             + "~r~n"

      // CASE statements and code omitted
      ...
      CASE ActBegin!
         IF atcan_node.Category = TraceIn! THEN
            lstr_result += "Begin Tracing~r~n"
         ELSE
            lstr_result += "End Tracing~r~n"
         END IF
      CASE ActGarbageCollect!
         lstr_result += "Garbage Collection~r~n"
      CASE else
         lstr_result += "Unknown Activity~r~n"
   END CHOOSE

   RETURN lstr_result
```

Generating a trace file without timing information

If you want to generate an activity log with no timing information in a text file, you can turn on PKDebug tracing in the System Options dialog box. The PKDebug trace file contains a log showing which object functions and instructions and system DLL functions were executed in chronological order.

❖ **To generate a simple trace file:**

1 Select Tools>System Options and check the Enable PKDebug Tracing check box.

2 Check the Prompt Before OverWriting PKDebug Output File box if you want to retain existing trace output when you run or debug the application.

3 (Optional) Specify a path name for the PKDebug output file.

4 Run your application.

If you do not check the Prompt Before OverWriting PKDebug Output File box, PocketBuilder overwrites the existing trace file every time you run the application or click the Start button in the debugger. If you check the box, it displays a response window. You can choose to overwrite the file, append new trace output to the existing file, or cancel the run or debug session.

If you want to retain the trace file, save it with a different name before running the application, or specify a new file name in the System Options dialog box.

If you do not specify an output file path, PocketBuilder creates an output file in the same directory as the PocketBuilder executable file. The output file has the same name as the PocketBuilder executable with the extension *DBG*. If you do not have write permission to this directory, you must specify a value for the output file path.

Turning PKDebug off
Running your application with PKDebug on slows down execution. Be sure to clear the Enable PKDebug Tracing check box on the System Options dialog box if you do not need this trace information.

CHAPTER 27

Packaging and Distributing an Application

About this chapter

This chapter describes how to create an executable version of your target and prepare a completed application for distribution to users.

Contents

Packaging an application

An application that you create in PocketBuilder includes one or more of the following pieces:

- An executable file (always required)
- Dynamic libraries
- Resources

To decide which of these pieces are required for your particular project, you need to know something about them.

The executable file

The executable file contains:

- Code that enables your application to run on the target platform.
- Compiled versions of objects from your application's libraries.
- Resources that your application uses, such as bitmaps.

You can choose to put all of your objects in the executable file or to split your application into one executable file and one or more dynamic library (PKD) files that contain objects that are linked at runtime.

About the Project painter
You use the Project painter to create an executable version of your target and a resource file that you can deploy to PocketPC devices, Smartphone devices, emulators, or the desktop. The Project painter allows you to streamline the generation of the files your target needs and to rebuild the target easily after you make changes to target objects. For how to create a new project using the Project painter, see "Creating a project" on page 717.

Dynamic libraries

You can deliver some (or even all) of the objects in your application in one or more dynamic libraries. Like executable files, dynamic libraries contain only compiled versions of objects, and they can include resources.

You can put any resources needed by a PKD's objects in the PKD file itself, so that the dynamic library is a self-contained unit that can easily be reused. If performance is your main concern, however, be aware that resources are loaded faster at runtime when they are included in the executable file.

Table 27-1 next lists several reasons why you might want to use dynamic libraries.

Table 27-1: Reasons to use dynamic libraries

Reason	Details
Modularity	They let you break up your application into smaller, more modular files that are easier to manage.
Maintainability	They enable you to deliver application components separately. To provide users with a bug fix, you can often give them the particular dynamic library that was affected.
Reusability	They make it possible for multiple applications to reuse the same components, because dynamic libraries can be shared among applications as well as among users.

Reason	Details
Flexibility	They enable you to provide your application with objects that it references only dynamically at runtime (such as a window object referenced through a string variable). You cannot put such objects in your executable file (unless they are DataWindow objects).
Efficiency	They can help a large application use memory efficiently because: • PocketBuilder does not load an entire dynamic library into memory at once. Instead, it loads individual objects from the dynamic library when needed. • Your executable file can remain small, making it faster to load and less obtrusive.

For more information about building dynamic libraries, see "Using dynamic libraries" on page 711.

Resources

Window, user, and DataWindow objects in a PocketBuilder application can use BMP, GIF, and ICO files as resources. These resources can be delivered with the application in several ways. For more information, see "Distributing resources" on page 713.

Using dynamic libraries

About dynamic libraries

You can store the objects used in your PocketBuilder application in more than one library and, when you run the application, dynamically load any objects that are not contained in the application's executable file. This allows you to break the application into smaller units that are easier to manage and makes the executable file smaller. You do this by using dynamic libraries. PocketBuilder builds PocketBuilder dynamic libraries (PKD files).

If you decide to use a dynamic library, you need to tell PocketBuilder which PocketBuilder library (PKL file) to create it from. PocketBuilder then places compiled versions of *all* objects from that PKL file into a PKD file with the same name. For example, the dynamic library built from *test.pkl* is named *test.pkd*.

Reducing the size of dynamic libraries

When PocketBuilder builds a dynamic library, it copies the compiled versions of all objects from the source PKL file into the dynamic library.

The easiest way to specify source libraries is simply to use your standard PocketBuilder libraries as source libraries. However, using this technique can make your dynamic libraries larger than they need to be, because they include all objects from the source library, not just the ones used in your application. You can create a PocketBuilder library that contains only the objects that you want in a dynamic library.

❖ **To create a source library to be used as a dynamic library:**

1 In the Library painter, place all the objects that you want in the dynamic library in one standard PKL file.

   If you need to create a new PKL file, select Entry>Library>Create from the menu bar, then drag or move the objects into the new library.

2 Make sure the application's library search path includes the new library.

Multiple dynamic libraries
You can use as many dynamic libraries as you want in an application. To do so, create a separate PKL file source library for each of them.

Specifying the dynamic libraries in your project

When you define your project, you tell PocketBuilder which of the libraries in the application's library search path will be dynamic by checking the PKD check box next to the library name in the Project painter.

Including additional resources for a dynamic library

When building a dynamic library, PocketBuilder does not inspect the objects; it simply copies the compiled form of the objects into the dynamic library. Therefore, if any of the objects in the library use resources (pictures and icons)—either specified in a painter or assigned dynamically in a script—and you do not want to provide these resources separately, you must list the resources in a PKR file. Doing so enables PocketBuilder to include the resources in the dynamic library when it builds it.

❖ **To reference additional resources:**

1 List the resources in a PKR file.

   How to list the resources is described in “Distributing resources” next.

2 Use the Resource File Name box in the Project painter workspace to reference the PKR file in the dynamic library.

Distributing resources

You can choose to distribute resources (pictures and icons) separately or include them in your executable file or dynamic library.

Including resources in the executable file

Whenever you create an executable file, PocketBuilder automatically examines the objects it places in that file to see if they explicitly reference any resources. It then copies all such resources right into the executable file.

PocketBuilder does not automatically copy in resources that are dynamically referenced (through string variables). To get such resources into the executable file, you must use a resource (PKR) file. This is simply a text file in which you list existing resources, including BMP, GIF, and ICO files.

Once you have a PKR file, you can tell PocketBuilder to read from it when creating the executable file to determine which additional resources to copy in. This might even include resources used by the objects in your dynamic libraries, if you decide to put most or all resources in the executable file for performance reasons.

Including DataWindow objects

You might occasionally want to include a dynamically referenced DataWindow object (one that your application knows about only through a string variable) in the executable file you are creating. To do this, you must list its name in a PKR file along with the names of the resources you want PocketBuilder to copy into that executable file.

You do not need to do this when creating a dynamic library, because PocketBuilder automatically includes every DataWindow object from the PKL source library in the PKD file.

Including resources in dynamic libraries

You might need to include resources directly in one or more dynamic libraries. PocketBuilder does not automatically copy any resources into a dynamic library that you create, even if they are explicitly referenced by objects in that file. You need to create a PKR file that tells PocketBuilder which resources you want in a particular PKD file.

Use a different PKR file for each dynamic library in which you want to include resources. When appropriate, you can even use this approach to generate a dynamic library that contains only resources and no objects. Simply start with an empty PKL file as the source.

Delivering resources as separate files

When you distribute an application, you can include the image files in addition to the application's executable file and any dynamic libraries. This can be useful if you expect to revise some of them in the future. However, it requires you to do some more work when you get ready to deliver your application, because you will not be able to use the CAB files generated by the Project painter without modification. For more information, see "Regenerating CAB files" on page 739.

Keep in mind also that this is not the fastest approach at runtime, because it requires more searching. Whenever your application needs a resource, it searches the executable file and then the dynamic libraries. If the resource is not found, the application searches for a separate file.

Make sure that your application can find where these separate files are stored; otherwise, it cannot display the corresponding resources. If you plan to install the files in a specific directory on the Pocket PC or Smartphone device, your scripts must reference the files using that path.

An example of delivering separate resources When a resource is referenced at runtime and it has not been included in the executable file or in a dynamic library, PocketBuilder looks for it in the device path provided for the resource. In test mode, you need only make sure the referenced files are in your machine's search path, but in applications that you deploy to a Pocket PC or Smartphone device, you must include the full path to the location where you install the referenced files.

In test mode on the desktop, if the referenced file is in the search path at runtime, the application can load it as needed. The desktop search path is as follows: current directory, Windows directory, Windows System directory, then all directories in the PATH environment variable.

Using PocketBuilder resource files

Instead of distributing resources separately, you can create a PKR file that lists all dynamically assigned resources. A PKR file is a Unicode or ANSI text file in which you list resource names (such as BMP, ICO, and GIF files) and DataWindow objects. To create a PKR file, use a text editor. List the name of each resource, one resource on each line, then save the list as a file with the extension *PKR*. Here are the contents of a sample PKR file:

```
ct_graph.ico
document.ico
codes.ico
button.bmp
```

```
next1.bmp
prior1.bmp
```

PocketBuilder compiles the listed resources into the executable file or a dynamic library file, so the resources are available directly at runtime.

Using DataWindow objects
If the objects in one PKL reference DataWindow objects (either statically or dynamically) that are in a different PKL, you must either specify a PocketBuilder resource file that includes the DataWindow objects, or define the library that includes them as a PKD that you distribute with your application. Unlike image files, you cannot distribute the objects separately.

When a resource such as a bitmap is referenced at runtime, PocketBuilder first looks in the executable file for it. Failing that, it looks in the PKD files that are defined for the application. Failing that, in test mode only (that is, on the desktop), it looks in directories in the search path for the file.

Using a resource file

❖ **To use a PocketBuilder resource file:**

1 Using a text editor, create a text file that lists all resource files referenced dynamically in your application.

You must include the path of the file if it is not in the current directory. See "Naming resources" next.

When creating a resource file for a dynamic library, list all resources used by the dynamic library, not just those assigned dynamically in a script.

2 Specify the resource files in the Project painter. The executable file can have a resource file attached to it, as can each of the dynamic libraries.

When PocketBuilder builds the project, it includes all resources specified in the PKR file in the executable file or dynamic library. You do not have to distribute your dynamically assigned resources separately; they are in the application.

Naming resources

If the resource file is in the current directory, you can simply list the file, such as:

```
FROWN.BMP
```

If the resource file is in a different directory, include the path to the file, such as:

```
C:\BITMAPS\FROWN.BMP
```

If the reference in a script uses a path, you must specify the same path in the PKR file. If the resource file is not qualified with a path in the script, it must not be qualified in the PKR file.

Paths in PKR files and scripts must match exactly
The file name specified in the PKR file must exactly match the way the resource is referenced in scripts.

For example, if the script reads:

```
p_logo.PictureName = "FROWN.BMP"
```

then the PKR file must read:

```
FROWN.BMP
```

If the PKR file says something like C:\MYAPP\FROWN.BMP and the script does not specify the path, PocketBuilder cannot find the resource at runtime.

PocketBuilder does a simple string comparison at runtime. In the preceding example, when PocketBuilder executes the script, it looks for the object identified by the string "`FROWN.BMP`" in the executable file. It cannot find it, because the resource is identified in the executable file by the string "`C:\MYAPP\FROWN.BMP`". In this case, the picture does not display at runtime; the control is empty in the window.

Resources not deployed to a path on the device
The resources that you include in a PKR are built into the executable or PKD; they are not deployed as separate files to the device or emulator. The fact that there is no *C:* drive on the device does not matter. "`C:\MYAPP\FROWN.BMP`" is simply a string that identifies the resource in the executable or PKD.

Including DataWindow objects in a PKR file

To include a DataWindow object in the list, enter the name of the library (with the extension PKL) followed by the DataWindow object name enclosed in parentheses. For example, here is a sample PKR file that includes DataWindow objects as well as other resources:

```
button.bmp
next1.bmp
prior1.bmp
logo.gif
dws.pkl(d_cust)
dws.pkl(d_custlist)
```

If the DataWindow library is not in the directory that is current when the executable is built, fully qualify the reference in the PKR file. For example: `c:\myapp\sales.PKl(d_emplist)`.

Creating a project

Table 27-2 lists wizards you can use to create a new project:

Table 27-2: Creating a PocketBuilder project

In the New dialog box	Do this
Target page	Use the PocketPC Application Creation wizard or the Smartphone Application Creation wizard to help you build an application from scratch
Project page	Double-click the Application icon to open the project painter where you can enter the required information without being prompted or use the Application wizard to help you set up a project object that will build an executable and optional dynamic libraries

Projects can be modified only in the Project painter
Unlike most other PowerBuilder objects, a project object cannot be edited in the Source editor.

The following procedure describes how to create a new project from the Project page.

❖ **To create a new project object from the Project page:**

1 Select File>New or click the New button in the PowerBar to open the New dialog box.

2 Select the Project tab.

3 Select the Application Wizard or Application and click OK.

- If you selected the Application wizard, complete the wizard screens to create a new project with the properties for which you are prompted, and then click Finish

  After you click Finish, the Project painter opens and displays your selections. You can also open the project generated by the wizard at any time to modify the properties you selected and build the project.

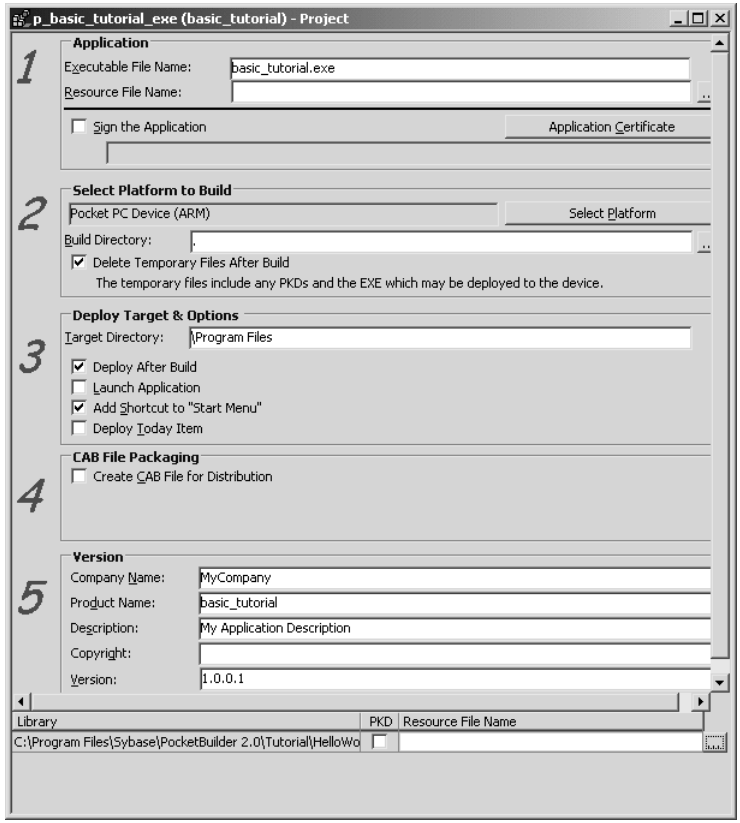
- If you selected Application, the Project painter opens so that you can specify properties of the project object

  After you close the Project painter, you can reopen the project from the System Tree at any time, modify the properties you selected, and deploy the project to the platform you select.

Defining the project

The Project painter allows you to streamline the generation of executable files and dynamic libraries for executable applications.

Figure 27-1: Displaying project properties in the Project painter



When you build a project object, you specify the following components of your application:

- Executable file name
- Which of the libraries you want to distribute as dynamic libraries
- Which PocketBuilder resource files (if any) should be used to build the executable file and the dynamic libraries
- Which platform you want to deploy to and what directories you want to use as your deployment directories

One platform per project
You can deploy your application to your desktop, a PocketPC device, a PocketPC emulator, a Smartphone device, or a Smartphone emulator. You can deploy to only one platform per project.

- Whether or not you want to select a certificate to use for signing your application
- Which deployment options you want to use in your project
- Whether or not you want to select a certificate to use for signing your application
- Version information for your application
- Whether or not you want to create a CAB file for distribution

Smartphone platforms
PocketBuilder uses the Microsoft Smartphone Cab Wizard (*cabwizsp.exe*) to generate CAB files for the Smartphone. The wizard executable is available in the Tools directory of the Smartphone 2003 SDK. You must place this file in the *PocketBuilder\Support\Cabwiz* directory before you can generate a CAB file for Smartphone platforms.

You can download the Smartphone 2003 SDK from the Microsoft Web site at http://www.microsoft.com/downloads/details.aspx?FamilyID=a6c4f799-ec5c-427c-807c-4c0f96765a81&DisplayLang=en.

- Whether or not you want to select a certificate to use for signing the CAB file

Pocket PC platforms
On the Pocket PC, you can sign only the application file itself. The Pocket PC platform currently does not support CAB file signing.

After you create a project, you might need to update it because your library list has changed or because you want to change your compilation options.

❖ **To define a project:**

1 Open a project in the Project painter by:

- Select File>Open from the PocketBuilder menu and then select the target, the library, the object type (Project), and the existing project that you want to modify
- Double-click a project in the System Tree

2 Specify or modify options as needed.

If you opened an existing project or a project created using the wizard, previously selected options display in the workspace. For information about each option, see "Executable application project options" next.

3 When you have finished defining the project object, save the object by selecting File>Save from the menu bar.

PocketBuilder saves the project as an object in the specified library. Like other objects, projects are listed in the System Tree and the Library painter.

Executable application project options

Table 27-3 describes each of the options you can specify in the Project painter for executable applications. You can also specify many of these options in the Application wizard in the Project page of the New dialog box.

Table 27-3: Project options

Option	What you specify or select
Executable File Name	Specify a name for the executable. The name must have the extension *EXE*.
Resource File Name	(Optional) Specify a PocketBuilder resource file (PKR) for your executable if you dynamically reference resources (such as bitmaps and icons) in your scripts and you want the resources included in the executable file instead of having to distribute them separately. You can type the name of a resource file in the box or click the button next to the box to browse your directories for the resource file you want to include. For more information, see "Distributing resources" on page 713.

Option	What you specify or select
Sign the application	(Usually signed; required to be signed by some operators such as AT&T and Orange) Specify that you want to sign the application and select a certificate to use by clicking the Application Certificate button.
Platform to Build	Select the platform for your application in the Select a Build Platform dialog box; you can build and deploy to only one platform for a given project. Desktop, PocketPC Device (ARM), and Smartphone Device (SPARM) are always listed. You deploy projects to Windows Mobile 5 or later emulators as you would to a Pocket PC or Smartphone device. If you have installed Windows Mobile 2003 emulators on your machine, the emulators are also listed as available platforms. If emulators are installed, you can select an emulator and click the Configure button to display the Windows CE Platform Manager Configuration dialog box, allowing you to add or test connections to additional devices and emulators.
Build Directory	Select a directory where you want to build your application. The default build directory is (.), which is the directory that contains the main PocketBuilder application library.
Delete Temporary Files After Build	Select the check box to delete temporary files after the build. The files include the executable and dynamic libraries deployed to the device.
Target Directory	Select a directory on the device or emulator where you want to deploy your application. The default deployment directory for a Pocket PC device or emulator is *\Program Files*. The default deployment directory for a Smartphone device or emulator is *\Storage\Program Files*.
Deploy After Build	Select this to deploy the application after the build is complete. If the check box is not selected, then using the Deploy menu item for a target or using the Deploy PowerBar button results in only a build, not a deploy.
Launch Application	Select this if you want to launch the application immediately upon deployment to a device or emulator.
Add Shortcut to Start Menu	(Not an option for the Desktop platform) Select to add a Start menu item for the deployed application on the device or emulator. For a Pocket PC, the path is Start>Programs. For a Smartphone, the path is Start>Accessories. You can override the default directory by changing the DeviceShortcutPath entry in the PocketBuilder registry.

Option	What you specify or select
Deploy Today Item	(Not an option for the Desktop or older emulator platforms) Select to deploy a custom item to the Today screen of a device. You enter the Today item properties on the Today Item page of the Properties view for the Application object. For more information, see "Application object properties for a custom Today item" on page 66.
Create CAB File for Distribution	(Not an option for the Desktop platform) Select this if you want PocketBuilder to build a CAB file for deployment to a device or emulator. For more information about building and distributing CAB files, see "Delivering your application to end users" on page 736.
Sign the CAB File	(Not an option for the Desktop or Pocket PC platform) Specify that you want to sign the CAB file and select a certificate to use by clicking the CAB Certificate button. For more information about signing CAB files, see "Signing an application and CAB file" on page 735.
Version	Specify your own values for the Company Name, Product Name, Description, Copyright, and Version fields associated with the executable file. These values become part of the Version resource associated with the executable file, and—if you deploy to the desktop—most of these values display on the Version tab page of the Properties dialog box for the file in Windows Explorer. The Company Name and Product Name fields are placed in the CAB file for your own reference.
PKD	Select this check box to define a library as a dynamic library to be distributed with your application.
Resource File Name	Specify a resource file for a dynamic library if library objects use resources (such as bitmaps and icons) and you want to include the resources in the dynamic library instead of having to distribute them separately. The file name cannot be specified in the wizard.

Building and deploying the project

Once you have completed development and defined your project, you can build and deploy the project to create the executable files and all specified dynamic libraries. You can build and deploy your project whenever you make changes to the objects and want to test another version of your application.

This section describes building a single project in the Project painter. You can build all the targets in your workspace at any time using buttons on the PowerBar, pop-up menus in the System Tree, or a command line. For more information, see the section on building workspaces in Chapter 1, "Working with PocketBuilder."

❖ **To build and deploy a project:**

1 Open a project you built in the Project painter.

2 Click the Deploy button in the PainterBar, or select Design>Deploy Project from the menu bar.

If the target's library list has changed
When you click Build, PocketBuilder checks your target's library list. If it has changed since you defined your project, PocketBuilder updates the Project painter workspace with the new library list. Make whatever changes you need in the workspace, then save the project and click Build again.

PocketBuilder builds the executable and all specified dynamic libraries.

The next two sections describe in detail how PocketBuilder builds the project and finds the objects used in the target.

When PocketBuilder has built the target, you can check which objects are included in the target. See "Listing the objects in a project" on page 728.

How PocketBuilder builds the project

When PocketBuilder builds your application project:

1 PocketBuilder regenerates all the objects in the libraries.

2 If you selected Build CAB File for Pocket PC Distribution, PocketBuilder creates a CAB file for the deployment targets you selected.

In the process of building a CAB file, PocketBuilder also creates INF, BAT, DAT, and LOG files. For a detailed description of what gets generated, see the information on packaging an application for distribution in "Delivering your application to end users" on page 736. For a discussion of what to do if errors occur during CAB file generation, see "Troubleshooting errors during CAB file generation" on page 728.

3 To create the executable file you specified, PocketBuilder searches through your target. It copies—into the executable file—the compiled versions of referenced objects from the libraries in the target's library search path that are not specified as dynamic libraries. For more details, see "How PocketBuilder searches for objects" next.

4 PocketBuilder creates a dynamic library for each of the libraries you specified for the target and maintains a list of these library files. PocketBuilder maintains the unqualified file names of the dynamic library files; it does not save the path name.

PocketBuilder does not copy objects that are not referenced in the application to the executable file, nor does it copy objects to the executable file from libraries you declared to be dynamic libraries. These objects are linked to the target at runtime and are not stored in the executable file.

What happens during execution

When an object such as a window is referenced in the application, PocketBuilder first looks in the executable file for the object. If it does not find it there, it looks in the dynamic library files that are defined for the target. For example, if you specified that a dynamic library should be generated from *test.pkl*, PocketBuilder looks for *test.pkd* at runtime.

The dynamic library files must be in the *\Windows* path on the device or in the directory where you deploy the project EXE. If PocketBuilder cannot find the object in any of the dynamic library files, it reports a runtime error.

How PocketBuilder searches for objects

When it searches through a target, PocketBuilder does not find all the objects that are used in your target and copy them to the executable file. This section describes which objects it finds and copies, and which it does not.

Which objects are copied to the executable file

PocketBuilder finds and copies the following objects to the executable file:

Objects that are directly referenced in scripts
Objects that are referenced in painters

Objects that are directly referenced in scripts

PocketBuilder copies objects directly referenced in scripts to the executable file. For example:

- If a window script contains the following statement, w_continue is copied to the executable file:

```
Open(w_continue)
```

- If a menu item script refers to the global function f_calc, f_calc is copied to the executable file:

```
f_calc(EnteredValue)
```

- If a window uses a pop-up menu using the following statements, m_new is copied to the executable file:

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

Objects that are referenced in painters

PocketBuilder copies objects referenced in painters to the executable file. For example:

- If a menu is associated with a window in the Window painter, the menu is copied to the executable file.
- If a DataWindow object is associated with a DataWindow control in the Window painter, the DataWindow object is copied to the executable file.
- If a window contains a custom user object that includes another user object, both user objects are copied.
- If a resource is assigned in a painter, it is copied to the executable file. For example, when you place a Picture control in a window in the Window painter, the image you associate with it is copied.

Which objects are not copied to the executable file

When it creates the executable file, PocketBuilder can identify the associations you made in the painter, because those references are saved with the object's definition in the library. PocketBuilder also identifies direct references in scripts, because the compiler saves this information.

However, PocketBuilder cannot identify objects that are referenced dynamically through string variables. To do so, it would have to read through all the scripts and process all assignment statements to uncover all the referenced objects. The following examples show objects that are not copied to the executable file:

- If the DataWindow object d_emp is associated with a DataWindow control dynamically using the following statement, d_emp is not copied:

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

- The bitmap files assigned dynamically in the following script are not copied:

```
IF Balance < 0 THEN
   p_logo.PictureName = "frown.bmp"
ELSE
   p_logo.PictureName = "smile.bmp"
END IF
```

- The reference to window w_go in a string variable in the following window script is not found by PocketBuilder when building the executable file, so w_go is not copied to the executable file:

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

Which objects are not copied to the dynamic libraries

When it builds a dynamic library, PocketBuilder does not inspect the objects; it simply copies the compiled form of the objects. Therefore, the DataWindow objects and resources (such as GIF files) used by any of the objects in the library—either specified in a painter or assigned dynamically in a script—are not copied into the dynamic library.

For example, suppose *test_dw.pkl* contains DataWindow objects and *test_w.pkl* contains window objects that reference them, either statically or dynamically. If you build a dynamic library from *test_w.pkl*, you must either include the DataWindow objects in a PocketBuilder resource file that is referenced by *test_w.pkl*, or build a dynamic library from *test_dw.pkl*, as described in "How to include the objects that were not found" next.

How to include the objects that were not found

If you use only the types of references that are included in the EXE or PKD files built by PocketBuilder (as described in "Which objects are copied to the executable file" on page 724), you do not need to do anything else to ensure that all objects get distributed: they are all built into the executable file or its dynamic libraries.

If you use the types of references described in the two previous sections ("Which objects are not copied to the executable file" on page 725 and "Which objects are not copied to the dynamic libraries" on page 726), you must include the objects that were not found, using the methods that follow.

Distributing graphic objects

For graphic objects in BMP, ICO, or GIF files, you have two choices:

- Distribute them separately
- Include them in a PKR file, then build an executable file or dynamic PocketBuilder library that uses the resource file

Graphic objects in JPEG files cannot be included in a PKR file and must be distributed separately if they are not built into the executable file. Graphic objects in PNG files cannot be included in a PKR file or an executable file, and can only be used if they are distributed separately.

Distributing DataWindow objects

For DataWindow objects, you have two choices:

- Include them in a PKR, then build an executable file or dynamic PocketBuilder library that uses the resource file
- Build and distribute a dynamic library from the PKL that contains the DataWindow objects

Distributing other objects

All other objects (such as windows referenced only in string variables) must be included directly in a dynamic library.

Table 27-4 summarizes resource distribution possibilities.

Table 27-4: Summary: options for distributing resources

Distribution method	Graphic objects	DataWindow objects	Other objects
As a separate file	Yes	No	No
In an executable or dynamic library that references a PKR	Yes	Yes	No
Directly in a dynamic library	No	Yes	Yes

Listing the objects in a project

After you have built your project, you can display a list of the objects in the project from the Project painter, with three columns that show:

- The source library that contains the object
- The name of the object
- The type of the object

The report lists the objects that PocketBuilder placed in the executable file and the dynamic libraries it created when you built the project.

You can resize and reorder columns in the report just as in grid DataWindow objects. You can also sort the rows and print the report using the Sort and Print buttons.

❖ **To list the objects in a project:**

1 Build your project.

2 Select Design>List Objects from the Project painter menu bar.

Troubleshooting errors during CAB file generation

After you build a project with the Build CAB File for Pocket PC Distribution option, you can check the PocketBuilder Output window or the *Err.log* file for a description of any errors in the build process.

If something goes wrong in the creation of the INF file or the BAT file, or in executing the *cabwiz.exe* to create the final CAB files, you are likely to see the following message in the Output window:

```
CAB Information File Generation Error
```

If no error messages display in the Output window, but a CAB file is still not generated, the error might be due to one of the following reasons:

- The generated INF file is somehow incorrect
- The generated BAT file is somehow incorrect
- The *cabwiz.exe, makecab.exe*, or *cabwizsp.exe* files are corrupted or missing from the PocketBuilder *support\cabwiz* directory

You can check the INF and BAT files in any text editor and you can reinstall the *support\cabwiz* directory from the PocketBuilder setup program.

For deployment to a Smartphone device, PocketBuilder uses the Microsoft Smartphone Cab Wizard (*cabwizsp.exe*) to generate CAB files. You must place this file in the *support\cabwiz* directory before you can generate a CAB file for these platforms.

The wizard executable is available in the Tools directory of the Smartphone 2003 SDK. The SDK download is available from the Microsoft Web site at http://www.microsoft.com/downloads/details.aspx?FamilyID=a6c4f799-ec5c-427c-807c-4c0f96765a81&DisplayLang=en.

If you still cannot generate a CAB file from the PocketBuilder project, you should clear the Create CAB File for Distribution check box in the Project painter.

The Output window displays the following message when CAB file generation is successful:

```
CAB Information File Generation Starting
Created:  D:\DirectoryName\ExeName.inf
Created:  D:\DirectoryName\ExeName_makecab.bat
CAB File(s) Generated in directory: D:\DirectoryName
------- Finished Deploy of ExeName_test
```

In this output message example, *DirectoryName* is the name of the project directory and *ExeName* is the name of the project executable.

Signing applications and CAB files

A certificate is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

The Project painter lets you sign each application file that you deploy with a certificate that you select from your desktop certificate store. You can also sign each CAB file that you deploy to a Smartphone platform from the Project painter. Signing ensures that the application and CAB files are secure.

For the digital signing process, PocketBuilder uses an internal facility. However, for manually signing your files, such as after using a batch file to create a CAB file, you can use the Code Signing Wizard on the Tool tab of the New dialog box in PocketBuilder.

Pocket PC platforms
On the Pocket PC, you can sign only the application file itself. CAB file signing is not currently supported on this platform.

Security concepts

Authentication and authorization

Authentication means that the identity of an entity (a person, client, or server) has been verified to either a server or a client. Authorization means that an entity has permission to use a resource or file. An entity must be authenticated before it can be authorized to use a resource or file.

Public-key cryptography

To maintain secure communications between a client and host, public key cryptography techniques are used for:

- **Authentication** Verifying the identity of both the client and the server. Public-key cryptography techniques use digitally signed certificates that identify network entities.
- **Encryption** Modifying data so that it can be read only by the party for whom it is intended. When used with a user's private key, certificates encrypt and decrypt messages.

Unencrypted messages are known as *plain text*. Encoding the contents of a message is called *encryption*. This encrypted message is the *cipher text*. *Decryption* is the process of retrieving the plain text from the cipher text. A key is usually required to perform encryption and decryption.

Public key encryption uses a pair of keys for encryption and decryption. One key is secret (the private key) and the other key is distributed (the public key). You send your digitally signed public key (certificate) to anyone with whom you wish to communicate using encoded data.

Messages that are sent to you are encrypted with your distributed public key and decrypted by your private key, while messages sent by you are encrypted with your private key and decrypted with your distributed public key. *RSA encryption* is a widely used public-key encryption system.

Public-key certificates

Public key certificates provide a way to identify and authenticate clients and servers on the Internet. Public key certificates are administered and issued by a third party known as a *certification authority* (CA). A subject (individual, system, or other entity on the network) uses a program to generate a key pair and submits the public key to the CA along with identifying information (such as name, organization, e-mail address, and so on). This is known as a *certificate request.* The CA issues a digitally signed certificate. A *digital signature* is a block of data that is created using a private key.

The CA ties the certificate owner to the public key within the certificate. The subject then uses the certificate, along with the private key to establish identity. Once this is done, whomever the subject is communicating with knows that a third party has vouched for his identity.

This process requires three steps:

1 A client submits a request for, and receives, a certificate from the CA.

2 An administrator installs the CA's certificate on the server and marks it trusted. Any client certificate signed by the same CA will now be trusted and accepted by the server.

3 The client supplies its certificate and negotiates a secure connection with the server.

SSL, HTTPS, and IIOPS

SSL provides security for network connections. Specifically, SSL uses public-key encryption to provide:

- Client and server authentication using certificates
- Encryption to prevent third parties from understanding transmitted data
- Integrity checking to detect whether transmitted data has been altered

Packets for other protocols can be embedded inside SSL packets. A connection in which the application protocol is embedded inside SSL is an *SSL-tunnelled* connection.

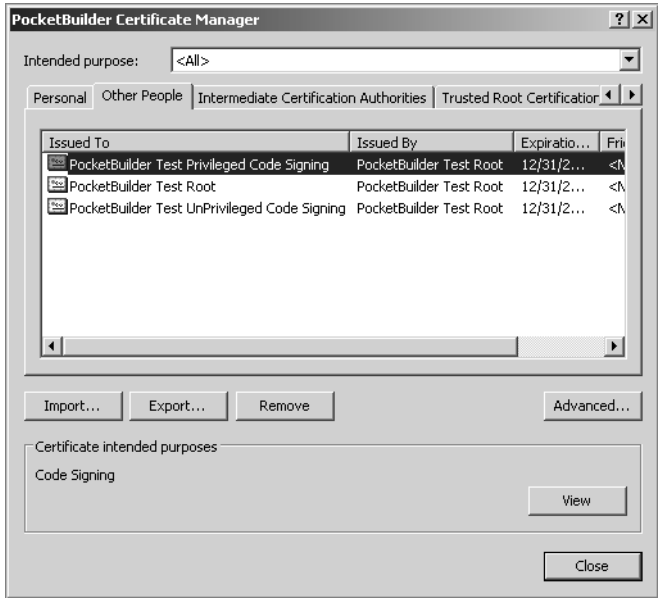
Both IIOP and HTTP can be tunnelled inside SSL, which means that these protocols take advantage of SSL security features. For example, HTTPS connections embed HTTP packets inside SSL packets. Your Web browser creates a secure HTTP connection any time you load a page from a URL that begins with *https*.

Managing certificates on the desktop

The PocketBuilder Certificate Manager lets you manage certificates that you create and certificates created by other people, intermediate certificate authorities, and trusted root certification authorities.

The PocketBuilder setup program installs several test certificates and packages in the *PocketBuilder/Support/TestCertificates* directory. You can import these certificates to your system certificate store. Using the PocketBuilder Certificate Manager, you can also export certificates and delete certificates you no longer need.

Figure 27-2: PocketBuilder Certificate Manager



Managing certificates on a connected device
The PocketBuilder Certificate Manager manages certificates only on the desktop. To manage certificates on a connected device, you can use the Device Certificate Management tool. For information about this tool, see "Managing certificates on a connected device" next.

❖ **To run the PocketBuilder Certificate Manager:**

- Select File>New and in the Tool tab page, select Manage Certificates.

Importing certificates to the certificate store

The Certificate Import Wizard helps you copy certificates, certificate trust lists, and certificate revocation lists from your disk to a certificate store.

❖ **To import the test certificates:**

1. Select File>New and in the Tool tab page, select Manage Certificates.
2. Click the Import button to start the Certificate Import wizard and then click Next.
3. To select the test certificate file you want to import, click Browse and then navigate to the PocketBuilder/Support/TestCertificates directory.
4. In the Files of Type drop-down list, select *Personal Information Exchange (*.pfx, *.p12)*, then select the path to the PKTestCert_Root.PFX file in the Test Certificates directory, and then click Next.
5. Enter `sybase` for the private key password and click Next.
6. For the PKTestCert_Root certificate, specify the certificate store by selecting the *Place all certificates in the following store* option, click the Browse button, select the Trusted Root Certification Authorities entry, and click OK.
7. Click Next and then click Finish.
8. Repeat steps 1-7 for the derived certificate package PKTestCert_Unprivileged.PFX, but in step 6, select the *Automatically select the certificate store based on the type of certificate* option.
9. Repeat steps 1-7 for the derived certificate package PKTestCert_Privileged.PFX, but in step 6, select the *Automatically select the certificate store based on the type of certificate* option.

Exporting certificates

The Certificate Export Wizard helps you export certificates, certificate trust lists, and certificate revocation lists in a new format and from the certificate store to other locations.

❖ **To export certificates:**

1. Select the certificate you want to export and click the Export button.
2. In the Certificate Export Wizard, specify the export file format and the file name and click Finish.

Deleting certificates

You can delete certificates using the PocketBuilder Certificate Manager.

❖ **To delete certificates:**

- Select the certificates you want to delete, click the Remove button, and then confirm your request by clicking the Yes button.

Managing certificates on a connected device

The Device Certificate Management tool lets you deploy certificates and review the certificate status of an attached device using the Windows Mobile 2003 platform. You can start this tool from the Tool tab of the New dialog box. In this tool, you can select one of the following options as the function of an XML file to download to a connected device:

- Install PocketBuilder Default Certificates
- Query for Unprivileged Certificates
- Query for Privileged Certificates
- Custom WAP Command File

PocketBuilder provides an XML file (*PK_AddCerts.xml* in the PocketBuilder *Support>TestCertificates* directory) that it assigns as the default source to install to a connected device when the Install PocketBuilder Default Certificates option is selected. Clicking the Install Certificates button deploys this XML file which contains references to all the default PocketBuilder test certificates.

Certificate expiration

The certificate used to sign the Sybase PocketBuilder DLLs expires every year. If the certificate expiration date has passed and you attempt to run a signed application, PocketBuilder returns an error message telling you that the security signature for the PocketBuilder VM has expired. Typically setup programs will be available on the Sybase Web site for PocketBuilder updates that include a *PK_AddCerts.xml* file referencing currently valid Sybase certificates. Otherwise you can assign your own certificates to the PocketBuilder DLLs prior to deploying them to a handheld device.

Additional XML files installed with PocketBuilder (*QueryStore_priv.xml* and *QueryStore_unpriv.xml*) can be used to query the status of the privileged and unprivileged certificates currently residing on a connected device. One of these XML files is automatically selected as the default source to download when a query option is selected for privileged or unprivileged certificates. The label for the Install Certificates button automatically changes to “Query for Certificates” when one of these options is selected.

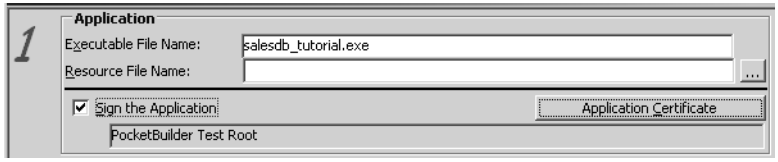
The Device Certificate Management tool also lets you deploy a custom XML file containing information for any certificates that you want to download to a connected device. The label for the Install Certificates button automatically changes to "Execute Custom WAP File on the Device" when the Custom WAP Command File option is selected. The file selection dialog box also opens automatically when the Custom WAP Command file selection is made.

Signing an application and CAB file

Signing an application

You can sign an application using the Application area (1) of the Project painter.

Figure 27-3: Signing an application for deployment



❖ **To sign an application:**

1 In area 1 of the Project painter, select the Sign the Application check box.

2 Click the Application Certificate button and select the certificate you want to use for signing the application.

Signing a CAB file

You can sign a CAB file using the CAB File Packaging area (4) of the Project painter. CAB signing is not supported for CAB files that you deploy to the Pocket PC.

Figure 27-4: Signing a CAB file for deployment

❖ **To sign a CAB file:**

1 In area 4 of the Project painter, select the Sign the CAB check box.

This area displays only a title if you selected Desktop as your deployment platform. It displays only a title and the check box for creating a CAB file if you selected Pocket PC as your deployment platform.

2 Click the CAB Certificate button and select the certificate you want to use for signing the CAB.

Delivering your application to end users

When you deliver the executable version of your application to users, you need to make sure that various files and programs are installed in the right places on the users' devices or emulators. Table 27-5 presents a summary of the types of files you need to distribute. You will distribute most of these components using cabinet (CAB) files.

Table 27-5: Distribution checklist

Checklist item	Details
The application	Application files include: • The executable (EXE) file • Any dynamic libraries (PKD files) • Any files for resources you are delivering separately (BMP, GIF, ICO, JPEG, and PNG files)
Additional files	Additional files might include: • Initialization (INI) files • Text or sound files
External files	If the application references external files, such as DLL files accessed by external function calls, install them in the *\Windows* directory on a Pocket PC or in the *\Storage\Windows* directory on a Smartphone.
PocketBuilder runtime DLLs	The following DLLs must be installed in the *\Windows* directory on every Pocket PC device or in the *\Storage\Windows* directory on every Smartphone device: *pkvm20.dll* *pkbgr20.dll* *pkodb20.dll* *pkodb20.ini*
Database files	If the application needs to access a local database, install the database files on each device. Typically you install the data source (the DSN file) in the device's root directory, and install the database and its log file in the same directory as the application. Make sure that the DSN file references the location where you install the database. If you are connecting to an UltraLite database, you must install the *pkul920.dll* or *pkul1020.dll* to the *\Windows* directory on a Pocket PC or the *\Storage\Windows* directory on a Smartphone device.

Checklist item	Details
Network server	If the application needs to access a consolidated database on a server, the device must be properly connected, and the databases on the device and the server must be correctly configured. For more information about synchronizing databases on the server and device, see the chapter in the *Resource Guide* about using MobiLink synchronization and the *Mobilink Server Administration* book in the SQL Anywhere online book collection.
Windows CE registry	If you rely on the Windows CE registry to manage information needed by the application, update the registry on each device with required values.

The rest of this chapter describes how you create CAB files and deliver them to your users. You can build CAB files with the Project painter or with the Enhanced CAB Generation tool.

Building CAB files with the Project painter

A CAB file is a platform-specific package containing all the files that your application needs, together with information about the application. The Microsoft Windows CE Cab Wizard (*cabwiz.exe*), a tool that builds CAB files, is installed in the *Sybase\PocketBuilder 2.0\Support\Cabwiz* directory when you install PocketBuilder. When you select the Create CAB File for Distribution check box in the Project painter, PocketBuilder invokes the Cab wizard to create the file when you build the project.

Smartphone platforms

PocketBuilder uses the Microsoft Smartphone Cab Wizard (*cabwizsp.exe*) to generate CAB files for the Smartphone. The wizard executable is available in the Tools directory of the Smartphone 2003 SDK. You must place this file in the *PocketBuilder\Support\Cabwiz* directory before you can generate a CAB file for Smartphone platforms.

You can download the Smartphone 2003 SDK from the Microsoft Web site at http://www.microsoft.com/downloads/details.aspx?FamilyID=a6c4f799-ec5c-427c-807c-4c0f96765a81&DisplayLang=en.

For more information about CAB files, see the Microsoft Web site at http://msdn2.microsoft.com/en-us/library/ms924762.aspx

What gets generated

When you choose to create a CAB file from the Project painter, PocketBuilder creates the files listed in Table 27-6 in the same directory as the PKL that contains the project object.

Table 27-6: Files generated by the Project painter

Filename	Description
AppName.ARM.CAB	The Windows CE cabinet files that you distribute to users as described in "Distributing the application" on page 744.
AppName.inf	The generated cabinet information (INF) file. This single ANSI text file has internal options for all selected platforms. The INF file merges your settings in the Project painter with the *cab_template.inf* file installed in the *Support\cabwiz* directory.
AppName_makecab.bat	A batch file that runs the Cab wizard (*CABWiz.exe* or *CABWizSP.exe*, depending on the platform selected for your project) using the generated *AppName.inf* file. This batch file, the INF file, and the platform-specific executable files let you regenerate the CAB files outside of the PocketBuilder environment.
AppNname.exe.ARM	Platform-specific executable files generated by PocketBuilder for use with the batch file.
err.log	A text file containing warnings and other messages from the Cab wizard. In general, warnings can be ignored.
AppNname.ARM.DAT	Temporary files generated by the Cab wizard. These files are used internally by Windows CE.

In Table 27-6, *AppName* is the name you supplied for the executable file in the Project painter. It is used as the base name of all the files generated when you choose to build a CAB file.

The information you supply in the Version group box in the Project painter is stored in the CAB file. After the CAB file is installed on an emulator or Pocket PC device, the company name and product name are visible in the list of programs in storage memory that displays when you select Settings>System>Remove Programs.

Modifying the template INF file

The INF file generated by the Project painter merges your settings with a template file, cab_template.inf, installed in the *Support\cabwiz* directory. If you are familiar with the structure of INF files, you can customize the CAB files generated by PocketBuilder by modifying the template.

Make a backup copy
Before you make any modifications to the cab_template.inf file, make a backup copy so that you can restore the default settings if necessary.

For more information about INF files, see the Microsoft Web site at http://msdn2.microsoft.com/en-us/library/ms924764.aspx

Regenerating CAB files

You can make modifications to the generated INF file, such as adding additional files, registry entries, and shortcuts, by editing the INF file in a text editor.

Regenerating overwrites modifications
If you regenerate the CAB file in the Project painter, your changes to *Appname.inf* are overwritten. After modifying an INF file, make a backup copy of the modified file with a different name so that you can reapply your changes.

When you run the *AppName_makecab.bat* batch file, it uses your edited INF file and the platform-specific executable files generated by the Project painter to build CAB files for all the platforms you selected in the Project painter. You must have a copy of the *AppName.exe* file to rebuild the CAB files. If you clear the Delete Temporary Files After Build check box in the project painter before you build the project, the *AppName.exe* and any required PKD files are saved to the build directory.

Errors when running the *AppName_makecab.bat* batch file are written to the *err.log* file.

Example Suppose you want to install your files in one location, but make the application accessible to the user by placing a shortcut in a company-specific folder that displays when you select Programs from the Start menu on the device.

In this example, the company name is AcmeTools and the product is Bug Zapper. The executable will be installed in the *\Acme Tools* directory, and the shortcut in *\Windows\Start Menu\Programs*.

1 In the generated INF file, locate the CEStrings section. It shows the name of the application and the installation directory you provided in the Version group box in the Project painter:

```
[CEStrings]
AppName = "Bug Zapper"
InstallDir = "\AcmePrograms"
```

2 Change the InstallDir line to the following:

```
InstallDir = %CE11%"\Acme Tools"
```

The executable will be installed in the *Acme Tools* directory as long as the %InstallDir% variable remains specified in the DestinationDirs section of the INF file. If this directory does not exist, it will be created.

3 Add a line that defines the shortcut to the Shortcuts.All section:

```
[Shortcuts.All]
"Bug Zapper",0,"bugzapper.exe",%CE11%
```

`Bug Zapper` is the name of the shortcut, `0` indicates the shortcut is to a file and not to a folder, `bugzapper.exe` is the target of the shortcut, and `%CE11%` is the location where the shortcut will be installed.

`%CE11%` represents the *\Windows\Start Menu\Programs* directory. For a list of other identifiers (macro strings), see Windows CE directory identifiers at http://msdn2.microsoft.com/en-us/library/ms934880.aspx.

4 Save the INF file, run *bugzapper_makecab.bat*, and check the *err.log* file for errors.

After you install the regenerated CAB file on a device or emulator, an *Acme Tools* folder that contains the Bug Zapper shortcut displays when you select Programs.

Building CAB files with the Enhanced CAB Generation tool

You can use the Enhanced CAB Generation tool as an alternative to the Project painter to generate CAB files for application distribution. The tool lets you include additional items in the generated CAB without manual modifications to the generated file.

The tool includes check boxes for the selection of files that you want to include with the CAB file that you generate. These selections include DLLs for the PocketBuilder VM, executables for the SQL Anywhere runtime engine, files for SQL Anywhere database connections, and registry entries to enable SMS reception capability in your PocketBuilder applications.

Building setup files with the Enhanced CAB Generation tool

The Enhanced CAB Generation tool also lets you generate a setup file that can install the CAB on a connected Windows CE device. The setup file generation requires the EZSetup executable that you can download from the Spb Software House Web site at http://www.spbsoftwarehouse.com/products/ezsetup/?en.

When you generate a CAB or setup file with the Enhanced CAB Generation tool, a log file, *err.log*, is also created. The *err.log* file logs warnings and errors that occur during generation of the CAB or setup files.

Table 27-7 lists the fields available on the different tab pages of the Enhanced CAB Generation tool.

Table 27-7: Tab pages of the Enhanced CAB Generation tool

Tab page	Field or control, and Description
Cab Config	• **Application Name** Required field for saving profile settings; can also be used in the application deployment path • **Company Name** Optional field that can be used as a directory name in the application deployment path • **Reset Deployment Path Using Company Name** Button that adds company and application name to deployment path for the CAB file • **Deployment Path** The default deployment path for the application is %CE1% (\Program Files) • **Create CABs for** Radio button options for selecting the device or emulator where you want to deploy the generated CAB file • **Directory for Application Files Included in Deployment** Directory containing the application files you want to deploy • **File Types to Include** Comma-separated list of the types of files to be added to the CAB from the appplication files directory; file extensions should be listed with a period (such as ".pkd, .dll") although files can also be specified with a standard file name • **Executable** The name of the executable file for the application • **Add Shortcut to Start Menu** Check box for deploying a shortcut for the application; this must be selected to enable the Shortcut Link field • **Shortcut Link** Displays where the shortcut is added; the shortcut must use the %CE11% (\Windows\Start Menu\Programs) special folder variable; you can add a subdirectory to this path when the field is enabled

Tab page	Field or control, and Description
Database Options	• **Deploy Application Database** Check box that must be selected to enable other options on the Database Options tab page • **DSN-File** The name and desktop location of a DSN file you want to use for the application database connection • **DB-File** The name and desktop location of a database file • **DB-Log File** The name and desktop location of a database log file • **DB Directory on Device** Directory on the device for the database file when you include one in the deployment CAB • **Include ASA Support DLLs and EXEs** Check box for adding SQL Anywhere DLLs and executables and enabling other database options in the CAB file • **Database** Version of the SQL Anywhere, Adaptive Server Anywhere, or UltraLite database system • **ASA Language** Two-letter code that determines the language you want to use with the database • **ASA Source** The location of the desktop database system files • **ASA Executables Path on Device** The path on the device where you want to deploy database executables
PocketBuilder Options	• **Include PocketBuilder Support DLLs** Check box for including PocketBuilder DLLs in the CAB file that you generate and enabling other options on the PocketBuilder Options tab page • **PocketBuilder Source** Desktop location of PocketBuilder support DLLs for the deployment platform you want to use • **Deploy AppList.exe** Check box for deploying the AppList utility • **AppList Location on Device** Location on the device where you want to install the AppList utility • **Deploy Remote Debugging Server** Check box for deploying the PKDebug remote debugging executable • **Deploy SMS Reception DLL** Check box for deploying the SMS reception DLL; you must select Include PocketBuilder Support DLLs to enable this check box • **Insert Registry Entries for SMS Reception** Includes registry entries for the SMS reception DLL in the CAB; the entries are added to the device registry when the CAB is unzipped • **SMS Reception is Read Only** Includes a registry entry string value that prevents deletion of an incoming SMS message by a PocketBuilder application

Tab page	Field or control, and Description
Preview	• **Generate CAB Info File** Button that displays the INF file contents in the Preview window • **Copy into Clipboard** Button that copies the INF file contents to the desktop clipboard • **Save Settings** Button that saves your settings in a CAB enhancement tool profile; the profile takes the name you selected in the Application Name field on the CAB Config tab page
Build	• **Output Path** Path on the desktop where you want to generate a CAB or setup file, and INF and CMD files • **CAB-INF Name** Name of the INF file that you generate for inclusion in the CAB file • **CMD File (for CAB generation)** The command file for generating the CAB • **EZSetup** Desktop location of the EZSetup executable that is required to create a setup file; you can download this executable at no charge from the Spb Software House Web site at http://www.spbsoftwarehouse.com/products/ezsetup/?en • **INF Name** Name of the INF file to generate with the setup file • **CMD File (for setup executable file creation)** Name of the command file that is automatically created and used to generate the setup file • **EULA File** End-user license agreement file that EZSetup requires for inclusion with the setup file it generates • **ReadMe File** Readme file for inclusion with the setup file; this is a required file for the setup file generation • **Language** Drop-down list of languages you can use in the setup file for the initial CAB installation screens • **Save Settings** Button that saves your settings in a CAB enhancement tool profile; the profile takes the name you selected in the Application Name field on the CAB Config tab page • **Create CAB File** Button that generates the CAB file • **Create Setup Executable** Button that generates the setup file
Afaria	For information about fields on this page, see “Deploying to Afaria with the Enhanced CAB Generation tool” next.

Deploying to Afaria with the Enhanced CAB Generation tool

The Enhanced CAB Generation tool contains a tab page for deploying an application to an Afaria® software channel. Afaria is a Sybase tool that provides management and security capabilities for wireless devices.

Table 27-8 lists the items on the Afaria tab page of the Enhanced CAB Generation tool.

Table 27-8: Fields and controls on the Afaria tab page

Field or control	Description
Afaria UNC path	Universal Naming Convention (UNC) location where the software for the Afaria channel resides. You should refresh the channel contents after deploying any files.
Deploy CAB	Select this check box to deploy a CAB file.
File Name	Select the CAB file that you want to deploy to an Afaria channel.
Deploy Files	Select this check box to deploy application files not packaged in a CAB file.
File Path	Select the path containing the application files you want to deploy.
File Types to Include	List the file types in the selected file path that you want to copy. By default, files with EXE, PKD, JPG, GIF, and INI extensions are copied to the Afaria channel.
Copy button	Click this button to copy the selected source file or files to the designated Afaria channel. The names and locations of the files that you copy display in the untitled list box below the Copy button.
Clear button	Click this button to clear information from the untitled list box.

Distributing the application

When you have created a CAB file, you can distribute it like any other self-extracting executable. The CAB file is a single package that can be downloaded from a Web site or copied using FTP to a user's desktop.

To install the CAB file, the user must copy it to the device or WM 5 emulator using Microsoft ActiveSync, or to older emulators using the Windows CE Remote File Viewer (cefilevw), then select the CAB file to extract its contents.

Creating a setup program

You can make the installation process easier for the user by creating a setup program. Running the setup program on the desktop installs the application to the device or emulator. The Enhanced CAB Generation tool makes it easy to create a setup file. You access the tool from the Tool tab of the New dialog box.

You can create installation programs using a free application such as EZSetup. This is the setup file creation program required by the Enhanced CAB Generation tool. For information about EZSetup, see the EZSetup download page at http://www.spbsoftwarehouse.com/products/ezsetup/index.html. For information about the Enhanced CAB Generation tool, see "Building CAB files with the Enhanced CAB Generation tool" on page 740.

Creating a Windows installer package

You can also build a Windows Installer package (MSI) file that users run on the desktop. The MSI file unpacks CAB files, a *setup.ini* file, and a custom installer component that distributes and installs the correct CAB file to the device.

For more information and to obtain the latest version of Windows Installer, see Windows Installer in the MSDN Library at http://msdn2.microsoft.com/en-us/library/aa372866.aspx.

Distributing the PKVM

Any application you distribute needs the PocketBuilder runtime files for Windows CE (the PKVM) installed on the target device or emulator. There are three ways to include these files (listed in Table 27-5 on page 736) in your distribution:

- Distribute the CAB files that are provided in the *WinCE* directory in your PocketBuilder installation.

  If many of your users already have PocketBuilder applications installed, they need only install the CAB file for your application. Other users need to download and install the CAB file containing the PKVM for their platform in addition to the CAB file for the application.

- Modify the generated INF file to include the PKVM and rebuild the CAB files, either as described in "Regenerating CAB files" on page 739, or by using the Enhanced CAB Generation tool.

  This is the easiest solution for your users, because they need only install one CAB file. However, it forces users who already have the PKVM installed to download and install unnecessary files. You can also add database files and executables to a CAB file using these methods.

- Build an installation program that lets users select the components they want to install.

  You can build an installation program with the Enhanced CAB Generation tool. This is the best solution for all users.

Distributing database files

For any application that uses a remote database, you must install both the database and a DSN file on the Pocket PC device or emulator. The DSN file can be installed in the root directory of the device, the directory from which the database driver is launched, or the *\Windows* directory. The database file must be installed in the location given in the DatabaseFile line in the DSN file.

You can edit the INF file generated by the Project painter or you can use the Enhanced CAB Generation tool to include these database files in your application CAB file.

You also need to deploy the database engine itself, which is provided in a CAB file as part of the SQL Anywhere install. See "Licensing SQL Anywhere" next for licensing information.

For more information about deploying SQL Anywhere databases, see the chapter on SQL Anywhere for Windows CE in the *SQL Anywhere Server Database Administration Guide*. The chapter on creating a remote database in the *MobiLink Client Administration* book provides an overview of the files that need to be distributed with MobiLink servers and clients.

If you need to deploy your application to multiple handheld devices, consider using mobile application deployment software, such as Afaria from iAnywhere Solutions, to manage the distribution process. For more information, see the Afaria Web page at http://www.ianywhere.com/products/afaria.html.

Licensing SQL Anywhere

Included in PocketBuilder are components from SQL Anywhere Studio (the SQL Anywhere and UltraLite database management systems, MobiLink and SQL Remote synchronization technologies, and the Sybase Central administration tool). These components are licensed for use only with PocketBuilder and only for development purposes. Deployment of these components requires the purchase of separate licenses. Contact your Sybase sales representative for more information.

Also included in PocketBuilder is the Adaptive Server Anywhere for Windows CE royalty-free runtime edition (the "ASA Runtime Edition"). The ASA Runtime Edition is a restricted-functionality version of the standalone version of Adaptive Server Anywhere 9. For example, the ASA 9 Runtime Edition does not support stored procedures and triggers, transaction log and synchronization.

The ASA Runtime Edition is intended for use as a low-cost deployment option where the full functionality of Adaptive Server Anywhere is not required. Subject to the PocketBuilder Product Specific License Terms, you may deploy the ASA Runtime Edition with applications developed with PocketBuilder without royalties or additional licensing.

For more information, including the list of Redistributable Components of the ASA Runtime Edition, see the *Support* directory on the PocketBuilder CD.

PART 7

Appendixes

Appendix A describes the extended attribute system tables, Appendix B describes differences between the PowerBuilder and PocketBuilder products, and Appendix C describes the use of OrcaScript for automatic builds and deployment.

APPENDIX A

Extended Attribute System Tables

About this appendix

This appendix describes each column in the extended attribute system tables.

Contents

About the extended attribute system tables

When you provide application-based information such as the text to use for labels and headings for the columns, validation rules, display formats, and edit styles for a database table, PocketBuilder stores this information in system tables in your database. These system tables are called the extended attribute system tables. The tables contain all the information related to the extended attributes for the tables and columns in the database. The extended attributes are used in DataWindow objects.

The system tables

There are five extended attribute system tables.

Table A-1: List of extended attribute system tables

Table	Contains information about
PBCatTbl	Tables in the database
PBCatCol	Columns in the database
PBCatFmt	Display formats
PBCatVld	Validation rules
PBCatEdt	Edit styles

What to do with the tables

You can open and look at these tables just like other tables, in the Database painter. You might want to create a report of the extended attribute information used in your database by building a DataWindow object whose data source is the extended attribute system tables.

Caution
You should not change the values in the extended attribute system tables. PocketBuilder maintains this information automatically whenever you change information for a table or column in the Database painter.

The extended attribute system tables

This section lists and describes all of the columns in each of the extended attribute system tables.

Not all columns apply to PocketBuilder applications
Some of these columns contain information that does not apply to PocketBuilder. The columns were assigned originally for PowerBuilder applications.

Table A-2: The PBCatTbl table

Column	Column name	Description
1	pbt_tnam	Table name
2	pbt_tid	Adaptive Server Enterprise Object ID of table (used for Adaptive Server Enterprise only)
3	pbt_ownr	Table owner
4	pbd_fhgt	Data font height, PowerBuilder units
5	pbd_fwgt	Data font stroke weight (400=Normal, 700=Bold)
6	pbd_fitl	Data font Italic (Y=Yes, N=No)
7	pbd_funl	Data font Underline (Y=Yes, N=No)
8	pbd_fchr	Data font character set (0=ANSI, 2=Symbol, 255=OEM)
9	pbd_fptc	Data font pitch and family (see note)
10	pbd_ffce	Data font typeface
11	pbh_fhgt	Headings font height, PowerBuilder units
12	pbh_fwgt	Headings font stroke weight (400=Normal, 700=Bold)

Column	Column name	Description
13	pbh_fitl	Headings font Italic (Y=Yes, N=No)
14	pbh_funl	Headings font Underline (Y=Yes, N=No)
15	pbh_fchr	Headings font character set (0=ANSI, 2=Symbol, 255=OEM)
16	pbh_fptc	Headings font pitch and family (see note)
17	pbh_ffce	Headings font typeface
18	pbl_fhgt	Labels font height, PowerBuilder units
19	pbl_fwgt	Labels font stroke weight (400=Normal, 700=Bold)
20	pbl_fitl	Labels font Italic (Y=Yes, N=No)
21	pbl_funl	Labels font Underline (Y=Yes, N=No)
22	pbl_fchr	Labels font character set (0=ANSI, 2=Symbol, 255=OEM)
23	pbl_fptc	Labels font pitch and family (see note)
24	pbl_ffce	Labels font typeface
25	pbt_cmnt	Table comments

About font pitch and family

Font pitch and family is a number obtained by adding together two constants:

Pitch: 0=Default, 1=Fixed, 2=Variable
Family: 0=No Preference, 16=Roman, 32=Swiss, 48=Modern, 64=Script, 80=Decorative

Table A-3: The PBCatCol table

Column	Column name	Description
1	pbc_tnam	Table name
2	pbc_tid	Adaptive Server Enterprise Object ID of table (used for Adaptive Server Enterprise only)
3	pbc_ownr	Table owner
4	pbc_cnam	Column name
5	pbc_cid	Adaptive Server Enterprise Column ID (used for Adaptive Server Enterprise only)
6	pbc_labl	Label
7	pbc_lpos	Label position (23=Left, 24=Right)
8	pbc_hdr	Heading
9	pbc_hpos	Heading position (23=Left, 24=Right, 25=Center)
10	pbc_jtfy	Justification (23=Left, 24=Right)
11	pbc_mask	Display format name

Column	Column name	Description
12	pbc_case	Case (26=Actual, 27=UPPER, 28=lower)
13	pbc_hght	Column height, PowerBuilder units
14	pbc_wdth	Column width, PowerBuilder units
15	pbc_ptrn	Validation rule name
16	pbc_bmap	Bitmap/picture (Y=Yes, N=No)
17	pbc_init	Initial value
18	pbc_cmnt	Column comments
19	pbc_edit	Edit style name
20	pbc_tag	(Reserved)

Table A-4: The PBCatFmt table

Column	Column name	Description
1	pbf_name	Display format name
2	pbf_frmt	Display format
3	pbf_type	Data type to which format applies
4	pbf_cntr	Concurrent-usage flag

Table A-5: The PBCatVld table

Column	Column name	Description
1	pbv_name	Validation rule name
2	pbv_vald	Validation rule
3	pbv_type	Data type to which validation rule applies
4	pbv_cntr	Concurrent-usage flag
5	pbv_msg	Validation error message

Table A-6: The PBCatEdt table

Column	Column name	Description
1	pbe_name	Edit style name
2	pbe_edit	Format string (edit style type dependent; see "Edit style types for the PBCatEdt table" next)
3	pbe_type	Edit style type
4	pbe_cntr	Revision counter (increments each time edit style is altered)
5	pbe_seqn	Row sequence number for edit types requiring more than one row in PBCatEdt table
6	pbe_flag	Edit style flag (edit style type dependent)
7	pbe_work	Extra field (edit style type dependent)

Edit style types for the *PBCatEdt* table

Table A-7 shows the edit style types available for the PBCatEdt table.

Table A-7: Edit style types for the PBCatEdt table

Edit style type	pbe_type value (column 3)
CheckBox	85
RadioButton	86
DropDownListBox	87
DropDownDataWindow	88
Edit	89
Edit Mask	90

CheckBox edit style (code 85)

Table A-8 shows a sample row in the PBCatEdt table for a CheckBox edit style. Table A-9 shows the meaning of the values in Table A-8.

Table A-8: Sample row in PBCatEdt for a CheckBox edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Text*	85	1	1	*Flag*	
MyEdit	*OnValue*	85	1	2	0	
MyEdit	*OffValue*	85	1	3	0	
MyEdit	*ThirdValue*	85	1	4	0	

Table A-9: Values used in CheckBox edit style sample

Value	Meaning
Text	CheckBox text
OnValue	Data value for On state
OffValue	Data value for Off state
ThirdValue	Data value for Third state (this row exists only if 3 State is checked for the edit style—bit 30 of *Flag* is 1)
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Left Text Bit 30: 3 State Bit 29: 3D Bit 28: Scale Box Bits 27 – 16 (3 hex digits): Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for CheckBox edit style Bit 3: Always 0 for CheckBox edit style Bit 2: Always 1 for CheckBox edit style Bit 1: Always 0 for CheckBox edit style Bit 0: Always 0 for CheckBox edit style

RadioButton edit style (code 86)

Table A-10 shows a sample row in the PBCatEdt table for a RadioButton edit style. Table A-11 shows the meaning of the values in Table A-10.

Table A-10: Sample row in PBCatEdt for a RadioButton edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Columns*	86	1	1	*Flag*	
MyEdit	*Display1*	86	1	2	0	
MyEdit	*Data1*	86	1	3	0	
MyEdit	*Display2*	86	1	4	0	
MyEdit	*Data2*	86	1	5	0	

Table A-11: Values used in RadioButton edit style sample

Value	Meaning
Columns	Character representation (in decimal) of number of columns (buttons) across.
Display1	Display value for first button.
Data1	Data value for first button.
Display2	Display value for second button.
Data2	Data value for second button. Display and data values are repeated in pairs for each radio button defined in the edit style.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Left Text Bit 30: 3D Bit 29: Scale Circles Bit 38: Not used (set to 0) Bits 27 – 16 (3 hex digits): Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for RadioButton edit style Bit 3: Always 1 for RadioButton edit style Bit 2: Always 0 for RadioButton edit style Bit 1: Always 0 for RadioButton edit style Bit 0: Always 0 for RadioButton edit style

DropDownListBox edit style (code 87)

Table A-12 shows a sample row in the PBCatEdt table for a DropDownListBox edit style. Table A-13 shows the meaning of the values in Table A-12.

Table A-12: Sample row in PBCatEdt for a DropDownListBox edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Limit*	87	1	1	*Flag*	*Key*
MyEdit	*Display1*	87	1	2	0	
MyEdit	*Data1*	87	1	3	0	
MyEdit	*Display2*	87	1	4	0	
MyEdit	*Data2*	87	1	5	0	

Table A-13: Values used in DropDownListBox edit style sample

Value	Meaning
Limit	Character representation (in decimal) of the *Limit* value.
Key	One-character accelerator key.
Display1	Display value for first entry in code table.
Data1	Data value for first entry in code table.
Display2	Display value for second entry in code table.
Data2	Data value for second entry in code table. Display and data values are repeated in pairs for each entry in the code table.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Sorted Bit 30: Allow editing Bit 29: Auto HScroll Bit 28: VScroll bar Bit 27: Always show list Bit 26: Always show arrow Bit 25: Uppercase Bit 24: Lowercase (if bits 25 and 24 are both 0, then case is Any) Bit 23: Empty string is NULL Bit 22: Required field Bit 21: Not used (set to 0) Bit 20: Not used (set to 0) Bits 19 – 16 (1 hex digit): Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for DropDownListBox edit style Bit 3: Always 0 for DropDownListBox edit style Bit 2: Always 0 for DropDownListBox edit style Bit 1: Always 1 for DropDownListBox edit style Bit 0: Always 0 for DropDownListBox edit style

DropDownDataWindow edit style (code 88)

Table A-14 shows a sample row in the PBCatEdt table for a DropDownDataWindow edit style. Table A-15 shows the meaning of the values in Table A-14.

Table A-14: Sample row in PBCatEdt for a DropDownDataWindow edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*DataWin*	88	1	1	*Flag*	*Limit*
MyEdit	*DataCol*	88	1	2	0	*Key*
MyEdit	*DisplayCol*	88	1	3	0	*Width%*

Table A-15: Values used in DropDownDataWindow edit style sample

Value	Meaning
DataWin	Name of DataWindow object to use.
DataCol	Data column from DataWindow object.
DisplayCol	Display column from DataWindow object.
Limit	Character representation (in decimal) of *Limit* value.
Key	One-character accelerator key.
Width%	Width of the dropdown part of the DropDownDataWindow in %.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Allow editing Bit 30: Auto HScroll Bit 29: VScroll bar Bit 28: Always show list Bit 27: Uppercase Bit 26: Lowercase (if bits 27 and 26 are both 0, then case is Any) Bit 25: HScroll bar Bit 24: Split horizontal scroll bar Bit 23: Empty string is NULL Bit 22: Required field Bit 21: Always show arrow Bit 20: Not used (set to 0) Bits 19 – 16 (1 hex digit): Not used (set to 0) Bits 15 – 8 (2 hex digits): Always 0 for DropDownDataWindow edit style Bit 7: Always 0 for DropDownDataWindow edit style Bit 6: Always 0 for DropDownDataWindow edit style Bit 5: Always 0 for DropDownDataWindow edit style Bit 4: Always 1 for DropDownDataWindow edit style Bit 3 – 0 (1 hex digit): Always 0 for DropDownDataWindow edit style

Edit edit style (code 89)

Table A-16 shows a sample row in the PBCatEdt table for an Edit edit style. Table A-17 shows the meaning of the values in Table A-16.

About the example

This example shows an Edit edit style using a code table of display and data values. There is a pair of rows in PBCatEdt for each entry in the code table *only if* bit 23 of *Flag* is 1.

For information about code tables in edit styles, see Chapter 21, "Displaying and Validating Data."

Table A-16: Sample row in PBCatEdt for an Edit edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Limit*	89	1	1	*Flag*	*Key*
MyEdit	*Format*	89	1	2	0	*Focus*
MyEdit	*Display1*	89	1	3	0	
MyEdit	*Data1*	89	1	4	0	
MyEdit	*Display2*	89	1	5	0	
MyEdit	*Data2*	89	1	6	0	

Table A-17: Values used in Edit edit style sample

Value	Meaning
Limit	Character representation (in decimal) of *Limit* value.
Key	One-character accelerator key.
Format	Display format mask.
Focus	Character "1" if Show Focus Rectangle is checked. NULL otherwise.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Uppercase Bit 30: Lowercase (if Bits 31 and 30 are both 0, then case is Any) Bit 29: Auto selection Bit 28: Password Bit 27: Auto HScroll Bit 26: Auto VScroll Bit 25: HScroll bar Bit 24: VScroll bar Bit 23: Use code table Bit 22: Validate using code table Bit 21: Display only Bit 20: Empty string is NULL Bit 19: Required field Bit 18: Not used (set to 0) Bit 17: Not used (set to 0) Bit 16: Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for Edit edit style Bit 3: Always 0 for Edit edit style Bit 2: Always 0 for Edit edit style Bit 1: Always 0 for Edit edit style Bit 0: Always 1 for Edit edit style

Edit Mask edit style (code 90)

Table A-18 shows a sample row in the PBCatEdt table for an EditMask edit style. Table A-19 shows the meaning of the values in Table A-18.

About the example

This example shows an Edit Mask edit style using a code table of display and data values as part of a spin control. Rows 2 and beyond exist in PBCatEdt only if the edit mask is defined as a spin control (bit 29 of *Flag* is 1). Rows 3 and beyond exist only if the optional code table is populated.

For information about using an edit mask as a spin control, see Chapter 21, "Displaying and Validating Data."

Table A-18: Sample row in PBCatEdt for an EditMask edit style

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	*Format*	90	1	1	*Flag*	*DtFcKy*
MyEdit	*Range*	90	1	2	0	*SpinInc*
MyEdit	*Display1*	90	1	3	0	
MyEdit	*Data1*	90	1	4	0	
MyEdit	*Display2*	90	1	5	0	
MyEdit	*Data2*	90	1	6	0	

Table A-19: Values used in EditMask edit style sample

Value	Meaning
Format	Display format mask.
DtFcKy	Concatenated string with 1-character data-type code, 1-character focus-rectangle code (0 or 1), and 1-character accelerator key. Data type codes: Format String = "0" Format Number = "1" Format Date = "2" Format Time = "3" Format DataTime= "4" Examples: "10x" means format is Number type, focus rectangle option is unchecked, accelerator key is "x" "31z" means format is Time type, focus rectangle option is checked, accelerator key is "z"
Range	Character representation (in decimal) of spin control range. The min value and max value are tab-delimited. Example: "1[tab]13" means min = 1, max = 13
SpinInc	Character representation (in decimal) of spin increment.

Value	Meaning
Display1	Display value for first entry in code table.
Data1	Data value for first entry in code table.
Display2	Display value for second entry in code table.
Data2	Data value for second entry in code table. Display and data values are repeated in pairs for each entry in the code table.
Flag	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked. Bit 31: Required Bit 30: Autoskip Bit 29: Spin control Bit 28: Read only (code table option) Bit 27: Use code table Bit 26: Not used (set to 0) Bit 25: Not used (set to 0) Bit 24: Not used (set to 0) Bit 23 – 16 (2 hex digits): Not used (set to 0) Bit 15 – 8 (2 hex digits): Always 0 for Edit Mask edit style Bit 7: Always 0 for Edit Mask edit style Bit 6: Always 0 for Edit Mask edit style Bit 5: Always 1 for Edit Mask edit style Bit 4: Always 0 for Edit Mask edit style Bits 3 – 0 (1 hex digit): Always 0 for Edit Mask edit style

APPENDIX B

PowerBuilder and PocketBuilder Product Differences

About this appendix

This appendix describes many of the differences between PowerBuilder and PocketBuilder. It also explains how to convert existing PowerBuilder applications to PocketBuilder.

Contents

PocketBuilder features

PocketBuilder is a 4GL application development tool for the Windows CE platform. It enables you to build rich-client handheld applications on the desktop and deploy them to a Windows CE environment.

The PocketBuilder IDE is similar to that of PowerBuilder. If you are a PowerBuilder user, you can leverage your existing PowerBuilder skill set, reusing knowledge and expertise to build and deploy mobile applications.

Key features

Key features of PocketBuilder include:

- Support for the Sybase DataWindow, providing powerful data access and sophisticated programming capabilities on Windows CE devices
- Tight integration with SQL Anywhere and UltraLite databases
- Support for MobiLink, enabling bidirectional data synchronization with heterogeneous enterprise systems

- Integrated tools for code signing, managing security certificates, and CAB file generation

PocketBuilder supports many PowerBuilder features without modification. You can reuse code from PowerBuilder applications, although PowerBuilder projects that you import to PocketBuilder must be redesigned for deployment to handheld devices.

File name changes

Although PowerBuilder and PocketBuilder use the same file extensions for exported objects, file extensions for libraries, targets, workspaces, and resource files differ to prevent confusion for developers who work with both products. Table B-1 lists the differences between these extensions.

Table B-1: File extensions that differ

File type	PowerBuilder extension	PocketBuilder extension
Workspace	PBW	PKW
Target	PBT	PKT
Library	PBL	PKL
Dynamic library	PBD	PKD
Resource file	PBR	PKR
Trace file	PBP	PKP

The file name for the PocketBuilder initialization file is *PK.INI*; the PowerBuilder initialization file is *PB.INI*. Both PocketBuilder and PowerBuilder 10 primarily support Unicode, although they allow you to import and export files with ANSI character sets. PowerBuilder 9 primarily supports ANSI character sets.

Environment variables

In both products, the enumerated types for the Environment object returned by a GetEnvironment call include the OSType and CPUType. PocketBuilder has additional values for the GetEnvironment OSType and CPUType. For a Pocket PC or Smartphone device, or a Windows Mobile 5 or later emulator, the value returned for the GetEnvironment OSType is “WindowsCE!” and the value returned for the CPUType is “ARM!”. For other Pocket PC or Smartphone emulators, the value returned for the CPUType is “Pentium!”.

System functions and window events for the Soft Input Panel

PocketBuilder has system functions to control the display of the Soft Input Panel (SIP) on a Pocket PC device or emulator, and window events that respond to changes in the SIP display. Table B-2 lists the system functions you can use to control the SIP display. The SipUp and SipDown window events occur when the SIP is opened and closed, respectively.

Table B-2: System functions for displaying and selecting the SIP

Function	Description
GetDeskRect	Gets the rectangular coordinates, in pixels, of the current window without including the area of the SIP when the latter is visible
GetSIPRect	Gets the rectangular coordinates of the SIP, whether it is visible or not
GetSIPType	Returns the type of the current SIP window, whether it is visible or not
IsSIPVisible	Indicates whether the SIP is currently visible to the user
SetRuntimeProperty	Allows you to set the following SIP properties at runtime: AllowAutomaticSIPHandling, SIPOnFocusUp_AlternateTechnique, and SIPOnFocusDown_AlternateTechnique.
SetSIPPreferredState	Displays or hides the SIP used on the Pocket PC or emulator
SetSIPType	Specifies the SIP panel type (keyboard or character recognizer) used by the application on a Pocket PC or emulator

Syntax and usage notes for the SIP system functions and window events are described in the online Help. These functions and events are not available in PowerBuilder applications.

System function for obtaining localized folder names

Operating systems using different languages can have different names for system folders. The GetSpecialFolder system function provides a means of obtaining the name of a system folder on a specific desktop machine or deployment device. This enables you to develop applications that access system folders and can still be deployed to devices using different language operating systems.

Objects and controls for Windows CE platforms

PocketBuilder includes objects and controls that are available to Windows CE platforms. Table B-3 lists these objects and controls. For more information, see Chapter 15, “Working with Native Objects and Controls for Windows CE Devices,”or *Objects and Controls* and the *PowerScript Reference*.

Table B-3: Native objects and controls for Windows CE platforms

PocketBuilder object or control	Description
BarcodeScanner object	Base class for capturing bar codes
BiometricScanner object	Base class for capturing fingerprints
CallLog and CallLogEntry objects	Intefaces with the call log on a Smartphone or a Pocket PC Phone Edition device
DialingDirectory object	Interfaces with the phone books on a Smartphone, SIM card, or Pocket PC device

PocketBuilder object or control	Description
NotificationBubble object	Displays a message to the application user
PhoneCall object	Makes a phone call from a Smartphone or Pocket PC Phone Edition device
POOM object (including POOMAppointment, POOMContact, POOMTask, POOMRecurrence, and POOMRecipient objects)	Stores appointments, contacts, and tasks, and integrates an application with the Pocket PC calendar and contact manager
Signature control	Captures user signatures, but can also capture hand drawings
SMSSession, SMSAddress, SMSMessage, SMSProtocol, and SMSProviderSpecificData objects	Interfaces with the SMS messaging system on a Smartphone or Pocket PC Phone Edition
Today item	Adds a custom item to the Pocket PC Today screen (Start page)
Toolbar control (including ToolbarItem objects)	Useful as a visual extension to a Pocket PC application menu

Differences required by target platform

The target platform for PocketBuilder applications is Windows CE. The Windows CE API is a subset of the API for traditional Windows platforms. The most obvious difference between Windows CE and Windows 2000 or Windows XP is the screen size (real estate) available to deployed applications. There are also stylistic differences for applications deployed to Windows CE platforms.

MOP views and menu styles

PocketBuilder lets you create multiple orientation views for windows, user objects, and DataWindow objects. You can use these orientations for creating layouts that change automatically when users switch screen orientation settings on their hand-held devices. For more information about MOP views in windows, see Chapter 10, “Working with Windows.”

Menus for handheld devices display at the bottom of windows, rather than at the top, as they do for desktop applications. Menus for Smartphones and Windows Mobile 5 devices consist of two top-level buttons that facilitate one-handed selection. For more information about menu styles, see Chapter 13, “Working with Menus.”

Window properties

The default window object size is smaller in PocketBuilder than in PowerBuilder, since it is tailored to the size of a Pocket PC screen. Main windows in PocketBuilder applications are also automatically resized to fit the device where they are deployed, and are automatically reoriented if the window layout settings on the device are changed. Main and response windows are currently the only window types available for PocketBuilder applications.

In addition to differences between the default values for window properties, some properties exist in one product and not in the other.

The following PowerBuilder window properties do not apply to windows in PocketBuilder applications, and are not selectable from the General tab page of the Properties view for windows, as they are in PowerBuilder:

ContextHelp	MinBox	Resizeable
ControlMenu	MDIClientColor	RightToLeft
MaxBox	PaletteWindow	WindowState

Unlike PowerBuilder, PocketBuilder does not have a Toolbar tab page in the Properties view for windows and menus. The Toolbar page applies only to MDI windows, which are not supported on the Windows CE platform. The PowerBuilder Pointer property on the Other tab page of the window Properties view also does not exist in PocketBuilder.

Table B-4 shows properties for windows in PocketBuilder that do not apply to PowerBuilder windows. (These properties apply to PocketBuilder applications deployed to a Pocket PC device or emulator, not to a Smartphone device or emulator.) You set these properties in the Properties view of the Window painter at design time. You cannot set these properties in a script at runtime.

Table B-4: Window properties only in PocketBuilder for the Pocket PC

Property	Description
Close	Adds an OK icon to the title bar of a main or response window that you deploy to a Windows CE platform. By default, when users click OK, user input is confirmed, the window object is destroyed, and the PocketBuilder application is closed.
SmartMinimize	Adds an X icon to the title bar of a main window that you deploy to a Windows CE platform. By default, when users click the X, the application is removed from the current navigational stack, but remains in memory for quicker availability and enhanced performance. This property and the Close property are mutually exclusive. Selecting one deselects the other.

Property	Description
MenuBar	Selecting this property makes room at the bottom of the current window for insertion of a menu. When you set a value for the MenuName property of a window, the MenuBar property is automatically selected.
IDE Window Size	Selects the window size at design time. At runtime windows are automatically configured to match a deployment device's full screen settings. For more information about the design time size setting, see "Choosing the window's size and position" on page 208.
ShowSIPButton	Ensures that the SIP button (that is used to open the soft input panel) displays at the bottom of the window when you run the window on a Windows CE platform. This is selected by default.
Tap_And_Hold_Indicator	Determines whether the red and green system visual cue displays when a user taps and holds down the stylus on a blank area of the window.

Platform-specific properties for multiple controls

In addition to window properties that are specific for the Pocket PC platform, platform-specific properties have also been added to different categories of controls. Table B-5 describes these properties.

Table B-5: General control properties only in PocketBuilder

Property	Description
InputEditMode	Integer property for edit controls and DataWindow columns that specifies an input method edit mode at runtime. You can use this property to set the SIP type on Pocket PC devices or the keypad entry mode on Smartphone devices.
SIPOnFocus	Boolean property for all edit controls that lets you display the SIP automatically when a user changes focus to one of these controls. This property is not supported on Smartphone devices or emulators.
Tap_And_Hold_Indicator	Boolean property for all draggable controls on a window (and for the window itself) that determines whether the red and green system visual cue displays when a user taps and holds the selected control (or window). This property is not supported on Smartphone devices or emulators.

Unsupported PowerBuilder features

PocketBuilder 2.0 supports PowerScript targets only. Because Web targets are not supported, the System Tree in PocketBuilder has a single Workspace tab. PocketBuilder does not currently support distributed applications or deployment to component transaction servers.

Using PocketBuilder with PocketSOAP
You can use PocketBuilder in conjunction with PocketSOAP to access online Web services. DLL files that define a PocketSOAP interface for PocketBuilder are available on the Sybase CodeXchange Web site at http://pocketbuilder.codexchange.sybase.com/. (Click the Pocket SOAP hyperlink and select the *soapif.zip* file for download.) A readme file is included in the Pocket SOAP folder and in the zip file on this Web site.

PowerBuilder installs a *Sybase\Shared* folder that contains subdirectories with PowerBuilder DLL files, Web target files, and Java support. There is no shared folder in a PocketBuilder installation.

Wizard differences

The differences between PowerBuilder and PocketBuilder functionality are reflected in the wizards provided with the two products. PocketBuilder has wizards that support the conversion of PowerBuilder and PocketBuilder targets, as well as separate wizards to generate template applications for the Pocket PC and the Smartphone.

Table B-6 describes differences between several PowerBuilder and PocketBuilder wizards. Wizards that differ only in the extension of the files they create are not included in this table.

Table B-6: Modified wizards in PocketBuilder

PocketBuilder wizard	Differences from PowerBuilder wizard
Application Creation wizards (Target tab)	In PocketBuilder there are separate wizards for Pocket PC and Smartphone targets. These wizards do not support selection of MDI or PFC application types.
Connection Object (PB Object tab)	Does not support EAServer connection; allows for entry of connection information that is not in a database profile.
Application wizard (Project tab)	Does not support creation of machine code EXE or DLL files or incremental builds; allows selection of deployment target options; version information is more limited than in PowerBuilder and is valid only for files deployed to the desktop. In PocketBuilder, you can select security certificates in the Project painter to sign application and CAB files.

Object types

Table B-7 shows standard class PowerBuilder object types that are not supported in PocketBuilder.

Table B-7: Unsupported standard class PowerBuilder object types

ADOResultSet	ErrorLogging	SSLCallback
Connection	JaguarORB	SSLServiceProvider
CorbaUnion	Mail object types	Timing
DynamicDescriptionArea	OLE object types	TransactionServer
DynamicStagingArea	Pipeline	

In addition, PocketBuilder does not support external visual user objects. The standard RichTextEdit and OLE visual controls are also not supported in PocketBuilder. ClassDefinition and ScriptDefinition objects, and other objects that descend from the ClassDefinitionObject object, are supported on the desktop but not on Windows CE devices or emulators.

PowerScript language

Datatypes The longlong datatype added for PowerBuilder 9 is not supported in PocketBuilder.

Functions The following classes of functions are not supported in PocketBuilder:

- Connection and TransactionServer object functions (such as CreateInstance and Lookup)
- CORBA object functions (such as BeginTransaction and Init)
- DBCS functions (such as LenW and PosW)
- DDE functions (such as CloseChannel and GetDataDDE)
- OLE object functions (such as Length and MemberExists)
- RichTextEdit control functions (such as CopyRTF and GetTextColor)
- SSLCallback and SSLServiceProvider object functions (such as GetPin and SetGlobalProperty)

Table B-8 lists PowerScript functions that are not currently implemented for the Windows CE platform. Although you are not prevented from coding these functions, if you call any of them at runtime, the calls will either be ignored, return partially valid data, or throw a system exception.

Table B-8: Unsupported PowerScript functions

Activate	GetLibraryList	LibraryExport
AddToLibraryList	GetRecordSet	LibraryImport
Arrangesheets	GetToolbar	LongLong
CanUndo	InsertDocument	Mail functions other than mailLogon, mailLogoff, and mailSend
ClassList	InsertFile	ObjectAtPointer
Connection object functions	IsAllArabic	OLE object functions
CORBA object functions	IsAllHebrew	PageCreated
CreatePage	IsAnyArabic	Profiling object functions
DBCS functions	IsAnyHebrew	RichTextEdit functions
DDE functions	IsArabic	SetLibraryList
GetArgElement	IsArabicAndNumbers	ShowHelp
GetActiveSheet	IsHebrew	ShowPopupHelp
GetFirstSheet	IsHebrewAndNumbers	SSLCallback functions
GetFolder	LibraryDirectory	SSLServiceProvider functions
GetLastReturn	LibraryDirectoryEx	Trace functions

Method limitations on Windows CE platforms

- The SetPointer function works only with the Hourglass! and Arrow! values in applications deployed to Windows CE platforms. The pointer is an arrow by default. If you set the pointer to an hourglass in a desktop application, the pointer reverts to an arrow after the script is run. On a Windows CE device you must explicitly call SetPointer a second time to reset the pointer.
- On Windows CE platforms, SetRedraw (FALSE) works only for the ListBox, DropDownListBox, and TreeView controls, however SetRedraw (TRUE) forces a repaint of all control types. This can lead to unexpected performance penalties in applications that you deploy to these platforms.
- You must install the FieldSoftware PrinterCE SDK before you can use print methods in PocketBuilder applications deployed to Pocket PC platforms. An evaluation version of this software is available from the FieldSoftware Web site at http://www.fieldsoftware.com.
- To send mail from a PocketBuilder application you must configure an ActiveSync to synchronize mail files with a desktop mail client. Microsoft Outlook and Outlook Express can be configured to work with ActiveSync.

Events Table B-9 lists events that are not supported on the Windows CE platform. Although you are not prevented from coding these events, if you add script for any of them, it will be ignored at runtime.

Table B-9: Unsupported PowerScript events

CloseQuery	RemoteExec	RemoteRequest
Help	RemoteHotLinkStart	RemoteSend
HotLinkAlarm	RemoteHotLinkStop	ToolbarMoved

DataWindow objects and database support

The DataWindow types supported by PocketBuilder are: Freeform, Graph, Grid, Group, and Tabular. PocketBuilder does not support the following DataWindow types: Composite, Crosstab, Label, N-Up, OLE 2.0, and RichText.

The “Rows to Disk” retrieve option for DataWindow objects is not available in PocketBuilder.

The ODBC driver for SQL Anywhere is the database driver installed with PocketBuilder, as well as a native driver for UltraLite. Database drivers for OLE/DB and JDBC, as well as DBMS native drivers, are not supported. PocketBuilder also does not support data pipeline objects. If you need to access an enterprise database from a PocketBuilder application, you can use MobiLink synchronization technology or convert the enterprise database to a SQL Anywhere database

For information describing MobiLink support in PocketBuilder, see "Using the MobiLink Synchronization for ASA wizard" on page 403 and the chapter on MobiLink synchronization in the *Resource Guide*.

Colors, presentation style, and figures

Some defaults for background colors have been changed from Btn_Face (Gray) in PowerBuilder to Window (White) in PocketBuilder. Some of the named colors on Windows machines are not supported on Windows CE devices. Unsupported colors are rendered in black on these devices.

Controls in PocketBuilder default to a 2D presentation style. If a 3D look is selected for a control, it might not have the desired appearance when deployed to a Windows CE device. Different versions of the Windows CE platform can vary in their support of 3D controls.

PocketBuilder supports GIFs, BMPs, JPGs, and stock icons for picture controls. Other picture files (WMFs, RLEs, and cursor files) are not currently supported.

Deployment and runtime differences

Debug and deployment options

For a PocketBuilder project, you specify deployment options that are not available in PowerBuilder. You must select one of the following deployment options for each project: Desktop, Pocket PC Device (ARM), Smartphone Device (ARM), or any of the emulators that you have installed.

An application that you deploy to the desktop will look slightly different from the same application deployed to a PDA device or emulator. The desktop application has its own title bar with a maximize, minimize, and close button. Even if you select Close or SmartMinimize icons for a window, these do not display in the window when it is run or debugged on the desktop. Desktop deployment is for testing and demonstration purposes only.

When you debug an application, you do not have access to the Tip Watch or Quick Watch views that were added to PowerBuilder in version 9. You must debug an application from the PocketBuilder IDE; you cannot run the PocketBuilder debugger with a deployed application.

Running applications on an emulator or PDA device

PocketBuilder has a toolbar icon and Tools menu item that opens the Select and Launch an Emulator dialog box. You can use this dialog box to launch an emulator from a saved state or a cold boot.

You can download Pocket PC and Smartphone SDKs from the Microsoft Web site. These SDKs include emulators that you can use as target platforms for your PocketBuilder applications.

Using the Windows CE Start Menu

By default, PocketBuilder applications are deployed to the *\Program Files* directory of a Pocket PC device or emulator and to the *\Storage\Program Files* directory on a Smartphone device or emulator, but you can change the deployment directory in the Project painter. On a Pocket PC device, users can run the PocketBuilder applications you deploy by tapping on an application executable file in the directory where it is deployed. (You can also select a Project painter option to launch the application immediately upon deployment.)

Users can take advantage of the built-in PocketBuilder application list utility as a selection vehicle for deployed applications. However, you can also select a Project painter option to deploy an application shortcut to the *\Windows\Start Menu\Programs* directory on the Pocket PC or to the *\Storage\Windows\Start Menu\Accessories* directory on the Smartphone. That way users will be able to find the deployed applications quickly using the Start menu.

Using the built-in application list

Users can display the built-in application list by selecting `PocketBuilder 2.1` from the Start menu. The list is empty until you begin deploying applications. By default, the application list displays executable files that you deploy to the *\Program Files* directory on the Pocket PC device or the *\Storage\Program Files* directory on the Smartphone. Users can launch any PocketBuilder application that you deploy to the device by selecting the application from this application list.

On the Pocket PC, the application list includes a directory browser that lets users select a different directory containing deployed applications. On the Smartphone, users can use the Menu soft key to change directories when browsing for deployed applications. However, on the Smartphone, it is not possible to browse above the parent directory of the application list utility, which is typically run from the *\Storage\Program Files* directory.

The application list is present on the device only if the complete PocketBuilder runtime package is installed. If you install only the PocketBuilder runtime DLLs to the device, the application list is not available.

Connecting to a SQL Anywhere database

By default, PocketBuilder runtime applications on the Windows CE device use the *PKODB20.DLL* dynamic link library to connect to SQL Anywhere data sources through the Adaptive Server Anywhere version 9 ODBC driver (*dbodbc9.dll*). You can specify a different Adaptive Server Anywhere ODBC driver by including the `driver=dbodbcX.dll` parameter in your DBPARM ConnectString assignment or in the file data source, where *X* is the version number of the ODBC driver for Adaptive Server Anywhere or SQL Anywhere.

For example, to use a the SQL Anywhere 10 ODBC driver on the Windows CE device, you can use the following DBPARM:

```
SQLCA.DBPARM="ConnectString='DSN=myDSN;driver=dbodbc10.dll;UID=dba;PWD=sql'"
CONNECT using SQLCA;
```

SQLCA is the default connection object in all PocketBuilder environments.

In all runtime environments, the connect string for the DBPARM parameter must not include spaces other than those in the data source name (DSN). The DSN you assign must exist in the root directory on the Windows CE device or emulator.

When you install the complete PocketBuilder runtime package to the device or emulator (rather than just the runtime DLLs), the *ASA 9.0 Sample.DSN* file is added to the root directory. Instead of using the ConnectString DBPARM to set the driver name, you can modify *ASA 9.0 Sample.DSN*, or any other DSN file that you use, so that it includes an assignment for the database driver:

```
[ODBC]
enginename=asademo
databasename=asademo
databasefile=\Program Files\Sybase\ASA\asademo.db
start=\Program Files\Sybase\ASA\dbsrv9.exe
Driver=dbodbc9.dll
```

If you are using the ASA Sample database that is migrated to SQL Anywhere 10, you can deploy the *DemoDB_SA10.DSN* file with your application. The PocketBuilder setup program installs this DSN file in the PocketBuilder *Code Examples\SA10DemoData* directory. The DSN file sets the the databse file to *\Program Files\SQLAny10\ASADemo_10.db*, and the driver name to *dbodbc.dll*.

SQL Anywhere requires that DSN files be saved in ANSI format rather than Unicode format.

If you are running your PocketBuilder applications from the desktop, it is recommended that you use the actual name of the driver (for example, "Adaptive Server Anywhere 9.0") in a ConnectString DBPARM rather than the name of the DLL. Otherwise, the driver you select in the connection string might be ignored.

Preventing the SQL Anywhere log screen from displaying

You can add a -qw switch to the start line in the DSN file to prevent the SQL Anywhere log screen from displaying when you connect to the database:

```
start=\Program Files\SQLANY10\dbsrv10.exe -qw
```

Because the connection might take a few moments, it is a good idea to call the SetPointer function to display the Windows CE version of the hourglass icon when using the `-qw` switch. For information about the use of SetPointer on the Windows CE platform, see the note on "Method limitations on Windows CE platforms" in this appendix.

CAB file creation and distribution

In PocketBuilder you can generate a CAB file with all the objects from a project and the project executable file. You can use the CAB file to distribute the project to multiple devices.

For more information about generating and distributing CAB files for your PocketBuilder projects, see Chapter 27, "Packaging and Distributing an Application."

Resizing and moving controls

PocketBuilder painters do not allow you to set properties for resizing and moving controls at runtime. However, you can still give users the ability to move and resize controls by modifying these control properties in code.

Converting a PowerBuilder application

PocketBuilder has wizards for importing PowerBuilder targets as PocketBuilder targets and exporting PocketBuilder targets as PowerBuilder targets.

If you plan to convert an existing PowerBuilder application to PocketBuilder, however, you should make as many modifications as possible prior to migration. The target conversion is more likely to succeed if you make initial changes to targets in PowerBuilder. This will help you avoid spending time after the migration fixing multiple compile errors due to third-party controls or MDI dependencies.

The recommended conversion strategy is therefore:

1 In PowerBuilder, duplicate the entire target and cut out the features that you know will not work or that are inappropriate for the Windows CE environment.

   These include features such as the use of MDI windows, large-sized windows, OLE and RichText controls, and CORBA and other distributed processing.

2 When you are satisfied that your application conforms reasonably well to requirements of the Windows CE environment, use the Import Desktop to Pocket wizard on the Target page of the New dialog box in PocketBuilder. (This wizard is not available in PowerBuilder.)

   The wizard prompts you for the PowerBuilder target you want to convert.

You can use PowerScript targets from PowerBuilder versions 7 through 11. The Import Desktop to Pocket wizard copies and converts all the PBL files in the target that you select on the first page of the wizard into PKL files, and it references the PKL files in a new target with a *PKT* extension. The original target and libraries are left in their original directories, which is also where the new target and migrated libraries are saved.

The conversion wizard gives you the option of adding the new target to the current PocketBuilder workspace.

Files converted by the wizard keep the same names as the original files, except for their extensions. The PowerBuilder VM for your source files (target and PBL files) must be in your machine's PATH variable. The wizard uses the *PBVMxx.DLL* and the compiler *PBCMPxx.DLL* to convert the source files, where *xx* is the version of the build.

Generating an application that is not in the current workspace can cause compile errors during the conversion of PBL to PKL files. However, once the application has been added to a workspace, a full rebuild typically fixes everything except for true compile errors.

For information on the Export Pocket to Desktop wizard for converting a PocketBuilder application to a PowerBuilder application, see “Converting a PocketBuilder target to PowerBuilder” on page 26.

APPENDIX C

The OrcaScript Language

About this appendix

This appendix describes the OrcaScript scripting language. OrcaScript allows you to perform source control operations and build PocketBuilder workspaces and executables without operator intervention.

Contents

About OrcaScript

OrcaScript allows you to write batch scripts to process PocketBuilder applications and files without using the PocketBuilder development environment. You can use OrcaScript to get the latest version of a PowerScript target from source control, build the target PKLs, and compile and deploy PocketBuilder executable files—all without operator intervention.

OrcaScript uses the Sybase ORCA software to access PocketBuilder library functions without a visual user interface. ORCA library source and header files, the ORCA User's Guide in PDF format, and a customer license agreement are installed in the PocketBuilder\Support\ORCA directory. None of these files is required to use OrcaScript.

Using OrcaScript with source control

The targets you obtain from source control using OrcaScript could be placed on a network build machine that is shared by PocketBuilder developers. This is especially advantageous for large shops with fixed working hours: the builds could be done nightly by running an OrcaScript batch file, and an up-to-date version of the targets and libraries would be available at the start of the next work day.

Developers could then use OrcaScript or operating system commands to copy the shared files directly to their local machines. Although developers would still connect directly to source control from their local workspaces, refreshing the targets in the workspaces would be much faster since compilation times for complex PowerScript targets would be greatly minimized.

Batch file order

If you include OrcaScript commands in a batch file, the file is read line by line. Each OrcaScript batch file must begin with a start session command and end with an end session command. You can save the batch file with any extension. You run the batch file by calling the OrcaScript executable on a command line and passing the batch file name as an argument:

```
pkorca20 myOrcaBat.dat
```

If you use relative directories in the OrcaScript batch file, the directories are relative to the working directories of the DOS shell. Typically, it is a good idea to put the OrcaScript batch file in the directory that contains the PKT file for the target you want to refresh, and to make this the current working directory before launching OrcaScript using the batch file.

Table C-1lists parameters you can use when you call the OrcaScript executable from a command line.

Table C-1: Command line parameters you can use with OrcaScript

Parameter	Description	Example
/D	Sets variables that are valid in the batch file	`pkorca20 /D myVar1=value1 /D myVar2=value2 myOrca.dat`
/H or /?	Prints syntax help to screen	`pkorca20 /H`

Caution

You should not run an OrcaScript batch file if PocketBuilder is currently running on the same machine. If the PocketBuilder development environment is not shut down while OrcaScript is running, your PocketBuilder libraries can become corrupted. For this reason, casual use of OrcaScript is not recommended.

Error handling

Each line of an OrcaScript batch file either succeeds or fails. If a command fails, subsequent commands are not processed and the OrcaScript session is ended. An error message is printed to the command window.

OrcaScript Commands

OrcaScript commands are not case sensitive. The generic command parameters can include only strings delimited by quotation marks, or predefined variables and constants without quotation marks. White space is used to separate multiple parameters for a single command. Any place a string is expected, a name that has been previously defined (set) in an OrcaScript command can be used.

In the OrcaScript command prototype syntax that follows, square brackets indicate a parameter is optional, and angle brackets indicate a parameter is required. A pipe character (|) inside brackets indicates that a selection can be made from one of the values inside the brackets. As elsewhere in the PocketBuilder documentation, text in italic type indicates a variable.

For commands where a string variable is required by the command syntax but is not essential to the command function (such as *pkrName* for the build library command), you can use an empty string inside quotation marks for the string value. Most of the OrcaScript commands and parameters are self-explanatory. For usage notes and an example of an OrcaScript batch file for obtaining a PowerScript target from source control, see "Usage notes for OrcaScript commands and parameters" next.

OrcaScript commands

OrcaScript supports the following commands:

start session
end session
set liblist *pkl_list* [*pkl_list* ...]
set application *pklName applicationName*
set *name* = *value*
set *name* += *value*
echo *value* [*value* ...]
file copy *fromFile toFile* [Clobber | NoClobber | Clobber Always]
file delete *file_name* [Clobber | NoClobber | Clobber Always]
regenerate *pklName entryName entryType*
copy entry *pklName entryName entryType topklName*
build library *pklName pkrName* <RTLib>
build executable *exeName iconName pkrName pkdflags*
    [newvstylecontrols] [Desktop | ARM | SPARM]
build application <full | migrate | incremental >
create library *pklName pklComments*
scc get connect properties *workspaceName*
scc set connect property deletetempfiles <true|false>
scc set connect property provider *sccProvider*
scc set connect property userid *userID*

scc set connect property password *password*
scc set connect property logfile *logFileName*
scc set connect property project *projectPath*
scc set connect property localprojpath *localProjectPath*
scc set connect property auxproject *auxProjectPath*
scc set connect property logappend < true | false >
scc set connect
scc set target *targetName* [*refreshType*][*refreshOption*][*refreshOption*]
scc get latest version *file_list* [*file_list* ...]
scc exclude liblist *pklName* [*pklName* ...]
scc refresh target *targetName* <full | migrate | incremental >
scc close

Argument description

Table C-2 lists and describes arguments you can use in OrcaScript commands.

Table C-2: Arguments for OrcaScript commands

Argument	Description
pkl_list	String containing the list of PKLs for the session application. PKL names can be separated by semicolons in a single string, or separated by a blank space in multiple strings.
pklName	Name of a PKL for an OrcaScript action or for the OrcaScript session application.
applicationName	Name of the application for an OrcaScript action.
name	Program variable that you define for an OrcaScript session.
value	Value of a program variable that you set for the OrcaScript session. The value is typically a string that you must place in quotation marks.
fromFile	File that you want to copy during an OrcaScript session.
toFile	Destination file name for a file that you copy during an OrcaScript session.
Clobber | NoClobber | Clobber Always (in file copy command)	Clobber replaces a destination file if it is marked read/write, but returns an error if a file does not exist or the destination file is read-only. NoClobber fails when the destination file already exists. Clobber Always replaces read-only destination files and does not return an error if the file does not exist. Clobber is the default behavior.
file_name	File name for a file that you want to delete.
Clobber | NoClobber | Clobber Always (in file delete command)	Clobber deletes read-only files and returns an error if a file does not exist. NoClobber deletes files marked read/write and returns an error if the file is read-only or if it does not exist. Clobber Always deletes read-only files but does not return an error if the file does not exist. Clobber is the default behavior.
entryName	The name of the referenced object.

Argument	Description
entryType	Value specifying the type of the referenced object. Values can be: application, datawindow, function, menu, query, structure, userobject, or window. Certain abbreviations (app, dw, fn, struct, uo, and win) are allowed as substitute values.
topklName	Name of the PKL to which you copy an entry.
pkrName	Name of a resource file you want to include in a build.
RTLib	Enter RTLib to generate PocketBuilder dynamic libraries.
exeName	Name of the executable you want to build.
iconName	Name of an icon to use for an executable you build with OrcaScript.
pkdFlags	String composed of a series of Y and N values for each library in the library list. A value of "nnyy" indicates that there are four libraries in the library list, the last two being PKDs. Objects from PKLs are copied into the executable; objects from PKDs are not copied.
newvstylecontrols	Use Microsoft XP visual style controls.
Desktop | ARM | SPARM	Use to select the deployment platform where you want the executable to be built. If an executable from a previous build exists, it will be replaced by the new executable on the target platform.
full | migrate | incremental	Build strategy for the session application.
pklComments	Comments for a PKL you create in an OrcaScript session.

Arguments for source control commands

In addition to some of the arguments listed in the Table C-2, Table C-3 lists and describes arguments you can use with OrcaScript source control commands.

Table C-3: Source control arguments for OrcaScript commands

Argument	Description
workspaceName	Name of the workspace to connect to source control.
sccProvider	Name of the source control provider.
userID	Name of the user registered to source control.
password	Password for the user ID.
logFileName	Name of a log file used to record SCC transactions.
projectPath	Path to the source control project.
localProjectPath	Local root directory for the project.
auxProjectPath	Contains any string that the SCC provider wants to associate with the project. It has a different meaning for every SCC vendor.
targetName	Name of the target for source control operations.

Argument	Description
true | false	Boolean value for appending to the source control log file. If this command is not used but a log file is specified, the session value defaults to true.
offline	Keyword indicating that an actual SCC connection will not be required for this session. It is appropriate only when the ImportOnly refresh option is used on a subsequent `scc set target` command. When refreshing a target using ImportOnly, no communication with the SCC provider is required at runtime, so the job may be run offline.
refreshType	Value can be “refresh_all” or “outofdate”. For a description of when to use these values, see "Setting refreshType and refreshOption values" next.
refreshOption	Value can be “importonly” or “exclude_checkout”. For a description of when to use these values, see "Setting refreshType and refreshOption values" next.
file_list	String containing one or more resource file names (such as GIFs, HLPs, or PKRs) using relative or absolute path specification. The string should not include file names for objects contained in application PKLs. File names can be separated by semicolons in a single string, or separated by a blank space in multiple strings. The list of files must be on a single line even when file names are contained in multiple strings.

Setting refreshType and refreshOption values

When you set up a target with a source control connection, you can use *refreshType* and *refreshOption* options in various combinations. You can even combine these values in the same string if the values are separated by a blank space. For example:

```
scc set target ".\dbauto\dbauto.pkt" "refresh_all
  importonly”
```

Refresh_all option Refresh_all performs no comparisons and imports all applicable objects. Refresh_all behavior varies depending on whether ImportOnly is set. If ImportOnly is off, Refresh All issues an SccGetLatestVersion call to obtain the tip revision of the PKT file specified in the scc set target command. From the PKT file, it obtains the library list for the target. It then calls SccGetLatestVersion to obtain the tip revision of the PKG file associated with each PKL.

Each PKG file contains a list of objects registered under source control that reside in the associated PKL. Refresh All then issues SccGetLatestVersion to obtain the tip revision of each object and imports these objects into the PKL.

In offline processing, ImportOnly must be set to on. If you also set the Refresh_all option, the PKT file that already exists on the local project path is used to obtain the library list for the target. The PKG file that also exists on the local project path is then read to obtain a list of objects associated with each PKL. Refresh_all then processes the PKG lists, importing source entries residing on the local project path into the appropriate PKL.

ImportOnly option When ImportOnly is on, the expectation is that the user has already populated the local project path with the desired revision of each object. ImportOnly is used to build a target from a previous milestone, such as a version label or a promotion model that does not represent the tip revision of each object. Therefore, no SccGetLatestVersion calls are issued. The desired revisions of the PKT, PKG, and object source files must already exist on the local project path and they are used to import objects into the PKLs. You must use this option if you are building a source controlled target while you are offline.

OutOfDate option OutOfDate processing behaves differently depending on whether ImportOnly is set. When ImportOnly is off, OutOfDate issues an SccGetLatestVersion call to obtain the tip revision of the PKT and PKG files. It then compares each object in the target PKLs with the tip revision in the SCC repository and imports the SCC source files into the PKLs for the objects that are out of sync.

With ImportOnly turned on, OrcaScript never performs GetLatestVersion since the desired revision of all objects already exists on the local project path. In this case, OutOfDate processing compares source code in the PKL against object source on the local project path to decide which objects, if any, need to be reimported. Using ImportOnly with OutOfDate processing works the same whether you are online or offline.

Advantage of using OutOfDate with ImportOnly option
Combining the OutOfDate option with the ImportOnly option is particularly useful if you perform nightly builds of a project that has several promotion models defined. If the volume of changes is low, it may be more efficient to use OutOfDate processing rather than Refresh All. In one PocketBuilder workspace, you build the "development" view of the project that includes all development work in progress. In another workspace, you build the "maintenance" view of the project, which includes bug fixes waiting for QA verification. Elsewhere, you build a "production" view of the project containing only verified bug fixes.

Each workspace connects to the same SCC project, but uses a different local project path.You use your vendor-specific SCC administration tool to synchronize the local project path with the desired revision of each object belonging to each promotion model. Then you launch OrcaScript to refresh the PKLs in each workspace. This results in a nightly rebuild of all three promotion models, which development team members can download each morning from a shared network drive.

Exclude_checkout option The Exclude_checkout option excludes from the import list all objects that are currently checked out by the current user, no matter what other refresh options are used. When connected to SCC, this option requires an additional call to SccQueryInfo for each object in the target. Therefore it is not recommended on a nightly build machine. However, it is highly recommended when a developer uses OrcaScript on his or her own workstation.

If you use Exclude_checkout processing while offline, the workspace PKC file is used to determine current status, so you must specify the `set get connect properties` *workspaceName* command. Objects marked as checked out to the current user in the PKC file will not be imported into the PKLs during target processing.

How the current user is determined for Exclude_checkout processing
For online SCC connections, Exclude_checkout calls `scc connect property userid` *userID* or the `scc get connect properties` *workspaceName* to determine the current user. The runtime processing makes actual SccQueryInfo calls to the SCC provider to determine check out status, so the information in the PKC file (from the prior SCC connection) is ignored. Objects checked out to the current user are not imported and replaced in the target library list.

For `scc connect offline`, the `scc connect property userid` command is completely ignored. OrcaScript must rely on information from the prior SCC connection. Each PKC file entry contains a bit flag that indicates "checked out to current user". This flag determines whether the object is imported or excluded. The current user at the time the PKC file was created is the user who last connected to this workspace through the PocketBuilder IDE on this workstation.

Usage notes for OrcaScript commands and parameters

Before calling any other ORCA functions, you need to open a session:

```
start session
```

You can start and end multiple OrcaScript sessions in the same batch file.

Copying files, objects, and properties

If you want to use OrcaScript simply to move objects among libraries, you do not need to set a library list or application. You can use the copy commands to copy files, objects, and properties. This example copies the d_labels DataWindow from the *source.pkl* library to the *destin.pkl* library:

```
copy entry "c:\\app\\source.pkl""d_labels" dw "c:\\app\\destin.pkl"
```

Setting a library list and an application

If you want to use OrcaScript to build targets or deploy components, you must first set the library list and the current application. You can set the library list and current application only once in an OrcaScript session. To use another library list and application, end the OrcaScript session and start a new session. The following OrcaScript commands build target libraries and compile an executable file:

```
start session
set liblist   ".\qadbtest\qadbtest.pkl;.\shared_obj\shared_obj.pkl;
   .\datatypes\datatype.pkl;.\chgreqs\chgreqs.pkl"
set application ".\qadbtest\qadbtest.pkl" "qadbtest"
build library ".\shared_obj\shared_obj.pkl" "" RTLib
build library ".\datatypes\datatype.pkl" "" RTLib
build library ".\chgreqs\chgreqs.pkl" "" RTLib
build executable ".\qadbtest\qadbtest.exe" ".\emp.ico" ".\qadbtest.pkr"
  "nyyy" ARM
file copy ".\qadbtest\qadbtest.exe" ".\bin\qadbtest.exe" Clobber Always
file copy ".\chgreqs\chgreqs.pkd" ".\bin\chgreqs.pkd" Clobber Always
file copy ".\datatypes\datatype.pkd" ".\bin\datatype.pkd" Clobber Always
file copy ".\shared_obj\shared_obj.pkd" ".\bin\shared_obj.pkd"
  Clobber Always
end session
```

You can use relative paths when you generate PKDs with the RTLib option, but the PKD always gets generated in the same directory as the PKL. To actually run the executable, you might have to move the PKDs to a "BIN" directory. The above example calls several file copy commands to accomplish this.

Source control example

You can use OrcaScript source control commands instead of the commands to set the library list and application. The following is an example of an OrcaScript session that builds the same libraries as the previous example, but uses the target properties to set a library list and application:

```
start session
scc get connect properties "c:\testbld\testbld.pkw"
scc connect
scc set target "c:\testbld\qadbtest\qadbtest.pkt" "outofdate
  exclude_checkout"
scc refresh target "incremental"
build library ".\shared_obj\shared_obj.pkl" "" pkd
build library ".\datatypes\datatype.pkl" "" pkd
build library ".\chgreqs\chgreqs.pkl" "" pkd
build executable ".\qadbtest\qadbtest.exe" ".\emp.ico" ".\qadbtest.pkr"
  "nyyy"

scc close
end session
```

You can call the scc connect command only after getting connection properties, and you must call it before you set or refresh the source-controlled targets. You must call the scc close command before you end your OrcaScript session.

Shared library example

If you have another target that shares libraries with a target that you already refreshed, you can use the OrcaScript exclude command to quickly reconstitute your target. The following example excludes the shared libraries *shared_obj.pkl*, *datatype.pkl*, and *chgreqs.pkl* that were refreshed in the previous example. It also demonstrates the use of variables for refresh options and build type. Set statements define variables that can be used throughout an OrcaScript session wherever the parser expects a string token.

```
start session
set refresh_flags = "outofdate"
set refresh_flags += "exclude_checkout"
set build_type = "incremental"
scc get connect properties "c:\testbld\testbld.pkw"
scc connect
scc set target ".\dbauto\dbauto.pkt" refresh_flags
scc exclude liblist ".\shared_obj\shared_obj.pkl"
   ".\datatypes\datatype.pkl" ".\chgreqs\chgreqs.pkl"
scc refresh target build_type
build executable ".\dbauto\dbauto.exe" ".\emp.ico" "" "nyyy" ARM
scc close
end session
```

Defining variables from the command line

Instead of defining variables in the OrcaScript session, you can define them from the command line when you call your script. If you saved the OrcaScript example in the previous script in a file named *MyExample.dat*, you could set the same variables by typing the following at a command line prompt:

```
pkorca15 /D refresh_flags="outofdate exclude_checkout"
/D build_type="incrementral" MyExample.dat
```

SCC connection properties

The SCC get connect properties command is an easy way to populate the Orca SCC connection structure with the source control properties of a local workspace. However, to create OrcaScript batch files that are portable from one workstation to another, the recommended technique is to set each property explicitly. Many of these properties are vendor specific. The best way to obtain correct values is to copy them directly from the SCC log file for your PocketBuilder workspace.

After you have obtained the values you need from the SCC log file, you can create portable batch files by setting the required connection properties and using relative directories and URLs for path information. The following example shows portable OrcaScript batch file commands for a workspace that connects to PBNative:

```
start session
scc set connect property provider "PB Native"
scc set connect property userid "Jane"
scc set connect property localprojpath ".\"
scc set connect property project "\\network_machine\PBNative_Archive\qadb"
scc set connect property logfile ".\MyPortableExample.log"
scc set connect property logappend "FALSE"
scc set connect property deletetempfiles "FALSE"
scc connect
; Perform refresh and build operations
scc close
end session
```

Sharing a batch file

If you share an OrcaScript batch file with colleagues, make sure that the *userid* value for the scc set connect property userid command is set for each user.

Using OrcaScript with source control targets offline

You can call `scc connect offline` to build source control targets offline. When you use this command, you must specify ImportOnly as a refresh option. If you also specify the Refresh_all option or the OutOfDate or Exclude_checkout refresh types, no connection is made to source control.

For the OutOfDate refresh type, the object source residing in the PKL is compared with the object source on the local project path. If these object sources are different, the object source on the local project path is imported and compiled. For the Exclude_checkout refresh type, the workspace PKC file is used to determine current status. In order for the offline exclude_checkout processing to locate the PKC file, you must use the `scc get connect properties` *workspaceName* command at the beginning of the script. Objects marked as checked out to the current user in the PKC file will not be imported into the PKLs during target processing.

For more information on PKC files, see Chapter 5, “Using Source Control.”

Applicable scc connect properties for offline processing
When `scc connect offline` is used, only the following connect properties apply:

```
scc set connect property localprojpath localProjectPath
scc set connect property logfile logFileName
scc set connect property logappend <true | false>
```

Build command failures

OrcaScript build commands for an executable or a library fail if the executable or library already exists in the build directory. To prevent an OrcaScript batch file containing these commands from failing, move or delete existing executables and libraries from the build directory before running the batch script.

Escape characters for string variables

OrcaScript, like PowerScript, uses the tilde (~) as an escape character. If you need to include a special character, such as a quotation mark, inside a string, you must place a tilde in front of it. A character in an OrcaScript batch file with a tilde in front of it is processed as a literal character.

Building executables for different platforms

The following example builds executables for three different PocketBuilder deployment platforms:

```
start session
set liblist "c:\pkapps\dbpaint\dbpaint.pkl;c:\pkapps\dbpaint\mlshared.pkl"
set application "c:\pkapps\dbpaint\dbpaint.pkl" dbpaint
;
build library "c:\pkapps\dbpaint\mlshared.pkl" "" rtlib
file copy "c:\pkapps\dbpaint\mlshared.pkd"
   "c:\pkapps\dbpaint\desktop\mlshared.pkd" CLOBBER ALWAYS
file copy "c:\pkapps\dbpaint\mlshared.pkd"
   "c:\pkapps\dbpaint\arm\mlshared.pkd" CLOBBER ALWAYS
file copy "c:\pkapps\dbpaint\mlshared.pkd"
   "c:\pkapps\dbpaint\sparm\mlshared.pkd" CLOBBER ALWAYS
;
build executable "c:\pkapps\dbpaint\desktop\dbpaint.exe"
   "c:\pkapps\emp.ico" "" "ny" DESKTOP
build executable "c:\pkapps\dbpaint\arm\dbpaint.exe" "c:\pkapps\emp.ico" ""
   "ny" ARM
build executable "c:\pkapps\dbpaint\sparm\dbpaint.exe"
   "c:\pkapps\emp.ico" "" "ny" SPARM
;
end session
```

Ending an OrcaScript session

You must close an OrcaScript session after you finish calling other OrcaScript commands. You close an OrcaScript session by calling:

```
end session
```

Property values are deleted during end-session processing. If an OrcaScript program starts numerous sessions, each individual session must contain statements to specify property values, such as those assigned in set exeinfo or scc set connect commands. However, variables that you set on a batch script command line using the /D parameter, or inside a batch file using the set *variable_name*="value" syntax, remain valid for the entire multisession program.

APPENDIX D

Designing Applications for Windows CE Platforms

About this chapter

This chapter describes some of the particularities of Pocket PC and Smartphone devices that you must account for when you deploy applications to these platforms.

Contents

General considerations

Because of the smaller size and typically intermittent energy supply for Windows CE platforms in comparison to Windows desktop platforms, fast and efficient code is essential to reduce power utilization.

When you open connections to peripheral utilities such as Bluetooth, SDIO, or WiFi, you must consider closing all peripheral handles as soon as such utilities are no longer needed. Applications deployed to Windows CE devices should save up requests for bulk usage and avoid constant polling. Polling loops prevent the Windows CE device processor from going into low-power idle mode, and thus reduce battery life.

Windows CE applications should also avoid unnecessary processing, such as that required for animations, and should lose focus and deactivate when overlayed by another application process. Where possible, you should add timers to applications to stop unnecessary processing.

Comparative performance

Table D-1 lists comparative performance characteristics of Smartphone and Pocket PC devices. Pocket PC Phone Edition devices have performance advantages of the Pocket PC with the functionality of the Smartphone.

Table D-1: Device performance comparison

Pocket PC	Smartphone
Faster processor	Slower processor
Persistent RAM backed up by storage battery	Permanent flash storage, but less RAM
On standby when not in use	Idle when not in use

Because of the generally weaker performance characteristics of Smartphones, design Smartphone applications to be tolerant of slow storage and to avoid saving too much data, especially persistent data.

Designing for the Pocket PC

Efficient design

In applications for the Pocket PC, you should reduce redundancy and promote a single way of doing things. This means avoiding the use of options by presenting only the most efficient selection or procedure to use. The main view of an application should include the most important information, and other information should be just a single step away. Presentation and procedures should be as consistent as possible, allowing application users to "learn once, do everywhere."

Tab pages, notifications, and information flow

On the Pocket PC, you can use a tab page approach to application design as a substitute for MDI windows, which are not supported on Windows CE platforms.

For persistent notifications, you can use the NotificationBubble object. Unlike a standard modal message box, the notification bubble does not stop processing.

On a Pocket PC, information should flow from the top down, and on English-language Pocket PCs, from left to right. You should place frequently needed controls close to the top of an application window. Audio should be used as a UI cue rather than as a novelty item. The default font for Pocket PCs is typically 8-point Tahoma.

Application titles, SIP display, and Pocket PC menus

You should display application titles in the navigation bar of a Pocket PC. On Pocket PC 2003, you should use the system support for automatic sizing of windows to fill the screen. (This is automatic with PocketBuilder applications.) Do not override these settings by entering custom size values. The Soft Input Panel (SIP) should display continuously for user text entry, and it should be hidden only where no user input is possible.

Menus on the Pocket PC are typically oriented from left to right, whereas buttons are typically aligned closer to the rightmost edge of the application screen. Typical Pocket PC applications can do without a File menu. A typical application might use the following menu bar items and display them in the order shown: Edit, View, Insert, Format, and Tools. Common command buttons display in the order: New, Open, Save, and Print.

Accelerator key combinations in menus and dialog boxes are not supported in Windows CE environments, and menu shortcuts should be avoided.

Menu styles for WM 5 devices

Microsoft Windows Mobile 5 is designed to facilitate one-handed operation of devices using this platform. To support this design constraint, Microsoft recommends using a maximum of two top-level menu items in all applications you deploy to devices or emulators running on the WM 5 or later. Although this design constraint is not yet a requirement for Pocket PC devices, it has been and continues to be a requirement for Smartphone devices.

At runtime, the PocketBuilder VM determines whether the operating system platform is WM 5 or later. If it is, and the application that you deploy opens a window with a menu containing one or two top-level menu items, PocketBuilder changes the menu design to the style used by menus on Smartphone devices—that is, the top-level menu displays as two adjoining buttons, with labels corresponding to the menu item names.

For information on designing for Smartphone applications, see “Designing for the Smartphone” next.

If, however, you open a window on a Pocket PC with the WM 5 or later operating system, and the window uses a menu with more than two top-level menu items, the menu style remains the same as the menu style on Pocket PC devices that use an older operating system. After a menu with a legacy style displays, all menus in the same application display with the legacy style, even if they have only one or two top-level menu items.

External files and icons

Pocket PC applications that use external file input or output must support long file names and the common system dialog boxes for opening and saving files. You can use 16x16 or 32x32 icons to represent a PocketBuilder application in the Pocket PC Switcher Bar. You should avoid user-exposed methods for closing applications. Applications should shut down without displaying dialog boxes or other controls for user input.

Using system functionality

Applications must not duplicate system functionality such as the Pocket Outlook Object Manager (POOM) object model, the notification system, or the MAPI, TAPI, and phone API functionality.

Providing online help

Online help for PocketBuilder applications should be integrated with the system TOC. You write help for an application as an HTML file and place the HTML file, or a shortcut to it, in the \Windows\Help directory. You can then call the PowerScript Run function to start the Pocket PC Help application and include the name of the Help file you want to open. The following example in a Help menu item script opens the Help file called myHelpTest:

```
Run ("peghelp.exe file:myHelpTest.htm#Main_Contents")
```

Wait cursors and size adjustments

Applications must respect user settings and display a standard wait cursor for unresponsive events.

Applications should adjust window and control sizes in SIPUp and SIPDown PowerScript events. The SIP must not overlay partial screen dialog boxes. Docked SIP panels have a maximum height of 80 pixels.

Saving application state

One instance only of an application should be running at a time. The application state is saved and restored on reactivation.

For a Pocket PC or Smartphone emulator, you must explicitly save the emulator state before shutting down the emulator, or the application is removed.

Designing for the Smartphone

Display screen and text entry modes

The screen for the Smartphone is typically smaller than that of the Pocket PC. It is for display only and does not trigger clicked events in response to user actions such as tapping an item with a stylus. It primarily serves as a telephone, not as a Personal Digital Assistant (PDA). A user interacts with a Smartphone through its keypad. Keypad entry is typically a one-handed operation.

Smartphones support several text entry modes from the Smartphone keypad. The current text entry mode is indicated in the title bar of a window when focus is in an editable field. For numeric data entry, the title bar has an icon displaying "123." For alphanumeric or multitap mode, the icon displays "abc", or "ABC" for entry of capital letters. Users can switch among modes by pressing or holding the asterisk key on the Smartphone keypad. (In numeric mode, just pressing the asterisk key enters an asterisk in the edit field.)

In PocketBuilder applications, you can assign a text entry mode to an edit field by setting its IMEMode property.

Application requirements

Typically applications must be signed with trusted certificates available on the Smartphone device. If you distribute an application in a CAB file, the CAB file must also be signed. PocketBuilder lets you select certificates for signing applications and CAB files in the Project painter. Although PocketBuilder supplies certificates for testing, you must purchase or generate your own certificates.

Window layout and control conversion

In the window layout for a Smartphone application, you should use one control (and control label) only per line.

In PocketBuilder applications, list boxes and drop-down lists are automatically converted to spinner controls. A spinner control is like a typical spin control on the desktop, except that arrows point to the right and left rather than up and down.

A user can spin through the items in a list by selecting the right or left arrow on the Smartphone's 4-way navigation pad. Otherwise, the user can press the Action button when a spinner control has focus. This opens a dialog box with the complete list of items. The user can select an item in this dialog box with the up and down arrows. After selecting an item, pressing the Action button again validates the selection and returns the user to the window with the spinner control.

Figure D-1 shows a window with a ListBox and a DropDownListBox control that have been converted to spinner controls. Spinner controls provide no indication of their original design-time control type. The only differences in the controls seen in the figure are due to their data content, background color property, and the current focus indicator. The focus indicator is a rectangular box drawn around the control.

Figure D-1: Window with two spinner controls

Menu design

Smartphones have two soft menu keys that link to the two main menus required for Smartphone applications. The left menu key typically opens a menu for quick, commonly used actions. In many Smartphone applications, when it functions to create new items or documents, this menu is labeled New. It is labeled Done or OK to save parameter settings and return to a previous screen. A Cancel menu item or button should be available for any windows that an application user can edit.

The right menu key typically opens a menu that has submenu items. If a right menu is not needed, it can be left blank, and the menu item can be left unscripted, but it still must be included in any menu object that you deploy with your application. Do not use ellipsis marks (...) in a menu label.

Focus indication

Indication of current focus is often problematic in Smartphone applications, particularly for noneditable controls.

If you add more than one command button to an application, it is best to select the Default property for one of the buttons. The Default property adds thicker black lines around the button, which serve, initially, as a focus indicator. Unfortunately, unless you set the Default property for the first button to false (for example, in a LoseFocus event script), the thick black lines remain. However, the other buttons on the same window now display the thick black lines when they have focus and display thinner black lines when they do not have focus.

When an editable control gains focus, it is typically surrounded, like the Default command button, by thick black lines in the form of a rectangle. The rectangle also typically encloses the control label.

Tabbing and scrolling using menu items

The Smartphone platform is not particularly well suited for tabbing among edit fields or scrolling in a window. The arrows on the 4-way navigation pad can be used to change the current focus, but with some controls (such as list views, multiline edit boxes, and DataWindows), the change in focus remains internal to the control or to a column in the control.

To change focus to a particular control, you can use the PowerScript SetFocus function in a ChangeFocus menu item. To provide a user with the ability to tab among columns in a DataWindow, it is useful to create a Tab menu item that mimics a Tab key press action. The following menu item script sends a Windows WM_KEYDOWN message for a Tab key press to a DataWindow control, enabling the user to tab among editable columns in the DataWindow object:

```
send(Handle(w_myMainWindow.dw_1),256,9,0)
```

When you use the above methods, you cannot change focus to a control that has a tab order of 0 or to a DataWindow column that is read-only. You can, however, scroll the window or DataWindow to display the control or read-only column. The following code in a ScrollToFirstColumn menu item script causes the first column to display on a screen even when the first column is read-only:

```
string modstring
modstring="DataWindow.HorizontalScrollPosition=" &
  + String(0)
w_myMainWindow.dw_1.Modify(modstring)
```

Closing applications

If an application does not save data in the flash memory storage file system or on a storage card, the data is lost when the user turns off the device.

The Microsoft online Help for the Smartphone SDK recommends that you simplify Smartphone memory management by not providing a Close button on an application toolbar or enabling Exit on a File menu. Memory use for running applications is optimized by the Smartphone shell. Smartphone devices send a WM_CLOSE message to close idle applications when demand for memory is high. To preserve an application's state from one session to another, you can save persistent-state variables to a temporary file after the application receives a WM_CLOSE message, but before the operating system is notified. The next time the application starts, it restores the application state using this temporary file.

Message processing on different devices

Prompting the user for information

Table D-2 lists the different ways you should use to process a question or display information to the user, depending on whether you deploy your application to a Pocket PC or a Smartphone device. This information is provided in more detail in the Smartphone User Interface Guidelines topic of the Microsoft Smartphone 2003 SDK online Help.

Table D-2: Displaying questions or information to prompt the user

Information type	On the Pocket PC	On the Smartphone
Message boxes	Use message balloons or partial-screen dialog boxes to prompt the user for information. Use full-screen message boxes primarily for system messages when the user must select an item from a list or enter data that is not selectable.	Use full-screen message boxes to prompt the user for information or answers to application-specific questions. Message box options should typically be limited to two.
Alert message boxes	Include code to wake up a Pocket PC device and display a message balloon or play a sound. Time-sensitive information should also be displayed in a balloon. Information that is not time sensitive can be displayed on the Today screen or in the notification area.	Use a full-screen alert message box only if the alert concerns time-sensitive information for a current activity or an activity running in the background. Information that is not time sensitive can be displayed on the Home screen or in the notification area. You cannot wake up a Smartphone device. It must be turned on by the user.
Options	Minimize the the number of controls in an Options dialog box. Do not use menus with the Options dialog box or the Settings control panel. Any tab content should not scroll, and all editable controls should be displayed above the expanded SIP area.	Use one column only and one control per row. Each row should be a minimum of 22 pixels high. Avoid contact with the title bar and soft keys by adding a two-pixel buffer at the top and bottom of the Options dialog box. Use the right soft key as the Menu soft key and the left soft key as the Done soft key.

Porting an application from the Pocket PC to the Smartphone

You must make certain manual changes to a Pocket PC application before deploying it to a Smartphone. Table D-3 lists required changes.

Table D-3: Modifications for porting an application to the Smartphone

Item to change or add	Required change for a Smartphone device
Left menu	Use a maximum of two menu bar items. Use the first menu bar item (left menu soft key) to display the most likely user task. When closing the window is a needed task, label this menu item "Done". Typically there are no submenu items on the left menu.
Right menu	Use the second menu item (right menu soft key) to display the second most likely user task. This menu item is required, although it can be left unscripted. If the menu item is not needed, the menu label can be an empty string. Use Cancel for the menu bar item or as a menu item listing any time an application state can be saved.
Radio buttons	Change radio buttons to spin controls.
Tab controls	Change tabs to list views with each tab pane as an item in the list.
Security certificate	Applications must have security certificates before they can be run on certain Smartphone platforms. You can create a security certificate with the Manage Certificates utility on the Tools page of the New dialog box. You can attach the certificate to a PocketBuilder application in the Project painter before you deploy the application to a Smartphone.

Screen rotation

PocketBuilder supports deployment to Windows Mobile 2003 Second Edition and Windows Mobile 5 devices and emulators. These platforms allow for screen rotation between portrait and landscape modes. They also support square screens and changes to screen resolution.

PocketBuilder lets you create multiple orientation views for windows, user objects, and DataWindow objects. You can use these orientations for creating layouts that change automatically when users switch screen orientation settings on their hand-held devices. For more information about MOP views in windows, see Chapter 10, "Working with Windows."

Effects of screen rotation

PocketBuilder processes messages from WM 2003 SE or later operating systems when the screen orientation or resolution has changed. It automatically makes adjustments to all Main! type application windows to conform to the current Windows CE environment. The title bar on a main window also resizes automatically. Any open menus or notification bubbles are closed, and if the SIP is raised immediately before screen rotation, it is automatically lowered.

Response windows and message boxes are not changed when the screen orientation or resolution changes. When an application is running and the display orientation is changed, any response windows and message boxes are recentered on the display. There is special logic to make sure the response window's title bar is still visible, since that is often the only way to close the window.

The Windows CE operating system sends a setting change message, PBM_Setting_Change, with a wParam value of 12290 when the user presses a button on the device (WM 2003 SE or later) to rotate the screen. The PocketBuilder pbm_settingchange event handler for this message resizes main windows to the new screen dimensions with required adjustments for any title and menu bars. If a MOP view exists for the new orientation, that MOP view is selected.

You can use MOP views or add code to the Resize event (or a pbm_settingchange user event) for a main window if you need to adjust the sizes or positions of controls on the window.

Obtaining the current screen size

You must instantiate a new Environment object to obtain the current screen height and width. Environment objects that are instantiated prior to screen rotation or resolution changes will have incorrect or stale information.

Deployment using CAB files

If you build CAB files for deployment, the VersionMin value (in the CEDevice section of the application INF file) is set to 3.0. This defines the application for compatibility with Pocket PC 2002 devices and emulators, but also permits deployment to Windows Mobile devices and emulators.

PocketBuilder adds the BuildMax value of 0xE0000000 to the CEDevice section of the application INF file. This value prevents application users from seeing a warning message when installing the CAB file on older Pocket PCs, but allows developers to take advantage of features introduced for WM 2003 SE and WM 5 platforms.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z