

 **Powersoft.**

Open Tools from Sybase, Inc.

PowerBuilder

Developer's Guide

Version 6

Power Builder®

AA0517

October 1997

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

All rights reserved.

Printed in Ireland.

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

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

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

Sybase, Inc. and its subsidiaries claim copyright in this program and documentation as an unpublished work, revisions of which were first licensed on the date indicated in the foregoing notice.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

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

AccuFonts is a trademark of AccuWare Business Solutions Ltd.

All other trademarks are the property of their respective owners.

Contents

About This Book xxi

PART 1 THE POWERBUILDER ENVIRONMENT

1	Working with PowerBuilder	3
	What is PowerBuilder?	4
	About the painters	4
	About events and scripts	5
	About functions.....	5
	About libraries	6
	Creating an executable	6
	The PowerBuilder environment.....	7
	About the PowerBar	7
	About PowerTips	8
	About the PowerPanel.....	8
	Painter summary	10
	Opening a painter or tool.....	11
	Using online Help	12
	Linking to the online books.....	12
	Using popup menus	14
	Using property sheets	16
	Using toolbars	18
	Toolbar basics	18
	Dropdown toolbars	18
	Controlling the display of toolbars	19
	Moving toolbars using the mouse.....	19
	Customizing toolbars.....	20
	Creating new toolbars	25
	Using the file editor	27
	Setting file editing properties	27
	Editing activities.....	28
	Executing AppleScript scripts on the Macintosh	29
	Changing fonts	31

	Defining colors	32
	Managing the initialization file	33
	About the initialization file	33
	Using the .WindU initialization file on UNIX	36
	Building an application	37
	Starting PowerBuilder from the command line	39
2	Working with Applications	43
	Overview of Application objects	44
	Creating a new Application object	45
	Working with other Application objects	48
	Using the Quick Application feature	49
	Looking at an application's structure	50
	Working in the workspace	50
	Which objects are displayed	51
	Specifying application properties	54
	Using the application properties tab	54
	Writing application-level scripts	62
	Setting application properties	63
3	Managing Libraries	65
	Overview of libraries	66
	Using libraries	66
	Organizing libraries	67
	Working with libraries	70
	Viewing the tree	70
	Using the popup menu	71
	Limiting the display of library entries	71
	Selecting library entries	73
	Using comments	73
	Creating and deleting libraries	75
	Copying, moving, and deleting entries	77
	Searching library entries	78
	Jumping to a painter	80
	Browsing the class hierarchy	81
	Using check-out and check-in	83
	How check-out works	83
	How check-in works	84
	Connecting to a version control system	84
	Checking out entries	85
	Viewing the checked-out entries	87
	Checking entries back in	88
	Clearing the check-out status of entries	89

Working with the Application object.....	90
Optimizing libraries.....	92
Regenerating library entries	93
Rebuilding libraries.....	94
Migrating libraries	95
Exporting and importing entries	96
Creating runtime libraries	100
Including additional resources.....	101
Creating reports on library contents	102
Creating library entry reports.....	102
Creating the library directory report.....	103

PART 2

CODING FUNDAMENTALS

4	Writing Scripts	107
	Opening the PowerScript painter	108
	Changing the current event.....	109
	Seeing which events have scripts	109
	Working in the PowerScript painter.....	110
	Modifying Painter properties.....	110
	Using the PainterBar and menu bar	113
	Getting context-sensitive Help.....	115
	Printing your script.....	115
	Pasting information.....	115
	Compiling the script.....	123
	Leaving the PowerScript painter	127
	Leaving without saving your work	127
5	Working with User-Defined Functions	129
	What are user-defined functions?	130
	Deciding which kind you want	131
	Defining user-defined functions.....	132
	Opening the Function painter	132
	Naming the function	134
	Defining a return type	134
	Defining the access level.....	136
	Defining arguments	136
	Coding the function	138
	Compiling and saving the function	140
	Correcting compiler errors.....	141
	Modifying user-defined functions	143
	Using your functions.....	146

6	Working with Structures	147
	What are structures?	148
	Deciding which kind you want	148
	Defining structures	149
	Modifying structures	152
	Using structures	153
	Referencing structures	153
	Copying structures	155
	Using structures with functions.....	155
	Displaying and pasting structure information	156

PART 3

WORKING WITH WINDOWS

7	Defining Windows.....	159
	Overview of windows	160
	Designing windows.....	160
	Building windows.....	161
	Types of windows.....	162
	Main windows.....	162
	Popup windows	163
	Child windows	164
	Response windows	165
	MDI frames.....	166
	Window types on the Macintosh.....	166
	Building a new window.....	169
	Opening the Window painter	169
	About the painter.....	170
	Working in the Window painter.....	171
	Defining the window's properties.....	171
	Viewing your work	183
	Previewing a window.....	183
	Printing a window's definition	184
	Writing scripts in windows	185
	About events for windows and controls.....	185
	About functions for windows and controls.....	186
	About properties of windows and controls.....	186
	Declaring instance variables	187
	Examples of statements	188
	Running a window.....	189
	Using inheritance to build a window.....	190

Working with Controls	195
Overview of controls.....	196
About controls with events	196
About the drawing objects	196
Placing controls in a window	197
Selecting controls	198
Defining a control's properties.....	199
Naming controls	201
About the default prefixes.....	201
Changing the name	202
Changing text.....	205
How text size is stored	205
Moving and resizing controls.....	207
Using the grid	207
Aligning controls	208
Equalizing the space between controls	209
Equalizing the size of controls	209
Copying controls	210
Defining the tab order.....	211
Establishing the default tab order.....	211
Changing the window's tab order	212
Defining accelerator keys	214
Specifying accessibility of controls	217
Using the Visible property	217
Using the Enabled property.....	217
Choosing colors	218
Using the 3D look.....	220
Using the individual controls	221
Using CommandButtons	222
Using PictureBoxes.....	223
Using RadioButtons.....	224
Using three states	226
Using StaticText	226
Using SingleLineEdits and MultiLineEdits	227
Using EditMasks.....	227
Using ListBoxes.....	230
Using PictureBoxes	232
Using DropDownListBoxes.....	233
Using DropDownPictureListBoxes	235
Using pictures.....	235
Using drawing objects	236
Using HScrollBar and VScrollBar	237
Using Tab controls	237

	Using TreeView controls	241
	Using ListView controls	244
9	Understanding Inheritance	247
	Overview of inheritance.....	248
	The inheritance hierarchy.....	249
	Viewing the hierarchy	250
	Working with inherited objects	251
	Using inherited scripts.....	253
	Viewing inherited scripts.....	253
	Overriding a script	254
	Extending a script.....	255
	Calling an ancestor script.....	256
	Calling an ancestor function.....	256
10	Working with Menus.....	259
	About menus and Menu objects.....	260
	Building a new menu.....	262
	Opening the Menu painter.....	262
	About the Menu painter	263
	Working in the Menu painter	264
	Adding Menu objects.....	264
	How Menu objects are named.....	266
	Inserting Menu objects	268
	Moving Menu objects	268
	Deleting Menu objects.....	269
	Defining the appearance of Menu objects.....	269
	Setting General properties	270
	Setting Style properties	271
	Assigning accelerator and shortcut keys.....	272
	Creating separation lines in menus.....	274
	Setting toolbar and picture properties	274
	Saving menus.....	276
	Viewing your work	278
	Previewing a menu.....	278
	Printing a menu's definition	279
	Writing scripts for Menu objects	280
	Menu object events	280
	Using functions and variables	281
	Referring to objects in your application	282
	Using inheritance to build a menu	285
	Using the inherited information.....	286
	Inserting Menu objects in a descendent menu	287

	Using menus	291
	Adding a menu bar to a window	291
	Displaying popup menus	292
11	Working with User Objects	293
	Overview of user objects	294
	Visual user objects	294
	Class user objects	296
	Building user objects	297
	Building a new user object	298
	Opening the User Object painter	298
	Building a standard visual user object	299
	Building a custom visual user object	300
	Building an external visual user object	301
	Building a standard class user object	303
	Building a custom class user object	304
	Events in user objects	304
	Saving a user object	305
	Using inheritance to build user objects	307
	Using the inherited information	307
	Using user objects	309
	Using visual user objects	309
	Using class user objects	310
	Communicating between a window and a user object	314
	Two methods	314
	Two examples: user object controls affecting a window	317
12	Working with User Events	321
	Overview of user events	322
	User events and event IDs	322
	Defining user events	325
	Using a user event	328
	Examples of user event scripts	328
PART 4	WORKING WITH DATABASES	
13	Managing the Database	335
	About databases	336
	About tables and columns	336
	About keys	336
	About indexes	337
	About views	337

About extended attributes	338
About managing databases	339
How you work with the database.....	339
Using the Database painter.....	342
Changing the database connection.....	342
About the painter	343
Modifying database preferences	347
Logging your work.....	348
Creating and deleting a SQL Anywhere database	351
Working with tables in the Database painter.....	353
Opening a table	353
Modifying properties in the Database painter	355
Closing a table.....	358
Dropping a table	358
Working with tables in the Table painter	360
About the Table painter	360
Creating a new table from scratch in the Table painter.....	360
Creating a new table from an existing table in the Table painter.....	362
Specifying column definitions and extended attributes	363
Modifying table properties in the Table painter	366
Altering a table in the Table painter.....	367
Cutting, copying, and pasting columns in the Table painter.....	369
Specifying fonts for the table in the Table painter	370
Working in SQL Syntax view	371
Logging applied SQL syntax changes.....	372
Printing the table definition	373
Working with keys	374
Why you should use keys.....	374
What you can do in the painters	375
Viewing keys	375
Opening related tables	376
Defining primary keys.....	376
Defining foreign keys.....	378
Dropping a key	379
Working with indexes	381
Creating an index	381
Modifying an index	382
Dropping an index	382
Working with views.....	384
Opening a view.....	384
Creating a view.....	385
Displaying a view's SQL statement	386

Joining tables	387
Dropping a view.....	389
Exporting table or view syntax.....	389
Manipulating data	390
Opening the Data Manipulation painter.....	390
Retrieving data	391
Modifying data	391
Sorting and filtering data	392
Viewing row information	395
Importing data	395
Printing data	396
Saving data	396
Returning to the Database painter workspace	397
Administering the database.....	398
About administering databases	398
Opening the Database Administration painter.....	398
Using the editor	399
Building and executing SQL statements	399

14

Defining DataWindow Objects.....	405
Introducing DataWindow objects.....	406
DataWindow object examples	406
How to use DataWindow objects.....	407
Introducing reports	409
Report examples	409
Reports versus DataWindow objects	415
Building a DataWindow object	417
Connecting to a database	417
Modifying an existing DataWindow object.....	417
Creating a new DataWindow object	417
Choosing a presentation style	420
Using the Tabular style.....	421
Using the Freeform style	421
Using the Grid style	422
Using the Label style	422
Using the N-Up style	424
Using the Group presentation style	426
Using the Composite presentation style	426
Using the Graph and Crosstab presentation styles.....	426
Using the OLE 2.0 presentation style.....	427
Using the RichText presentation style.....	427
Choosing DataWindow object-wide options.....	428
Defining the data source	430
How to choose the data source.....	430

- Using Quick Select 431
- Using SQL Select 441
- Using Query 455
- Using External 456
- Using Stored Procedure 457
- Generating and saving a DataWindow object 460
 - About the Powersoft repository and DataWindow objects 460
 - Saving the DataWindow object 461
- Defining queries 462
 - Previewing the query 462
 - Saving the query 463
 - Naming the query 463
 - Modifying a query 464
- What's next..... 465

15

Enhancing DataWindow Objects..... 467

- Working in the DataWindow painter workspace 469
 - Understanding the DataWindow painter workspace 469
 - Using the DataWindow painter toolbars 472
 - Using property sheets in the DataWindow painter 473
 - Selecting objects in the DataWindow painter 474
 - Using keyboard shortcuts in the DataWindow painter..... 476
 - Resizing bands in the DataWindow painter workspace 478
 - Using zoom in the DataWindow painter 478
 - Undoing changes in the DataWindow painter 478
- Previewing a DataWindow object..... 479
 - Retrieving data 480
 - Modifying data 483
 - Sorting and filtering data 484
 - Viewing row information 485
 - Importing data into a DataWindow object..... 485
 - Using print preview 486
 - Printing data 488
 - Saving data in an external file 488
 - Saving the data in HTML Table format..... 489
 - Working with PSR files 491
 - Mailing reports..... 493
 - Working in a grid DataWindow object 494
- Modifying general DataWindow object properties 496
 - Changing the DataWindow object style..... 496
 - Setting colors in a DataWindow object..... 497
 - Specifying properties of a grid DataWindow object 498
 - Specifying pointers for a DataWindow object..... 499

Defining print specifications for a DataWindow object	499
Modifying text in a DataWindow object	503
Defining the tab order in a DataWindow object	504
Naming objects in a DataWindow object	506
Using borders in a DataWindow object	506
Specifying variable-height detail bands in a DataWindow object	507
Modifying the data source of a DataWindow object	509
Reorganizing objects in a DataWindow object	512
Displaying boundaries for objects in a DataWindow object	512
Using the grid and the ruler in a DataWindow object	513
Deleting objects in a DataWindow object	513
Moving objects in a DataWindow object	514
Copying objects in a DataWindow object	514
Resizing objects in a DataWindow object	515
Aligning objects in a DataWindow object	516
Equalizing the space between objects in a DataWindow object	516
Equalizing the size of objects in a DataWindow object	517
Sliding objects to remove blank space in a DataWindow object	517
Prompting for retrieval criteria in a DataWindow object	519
Adding objects to a DataWindow object	521
Adding columns to a DataWindow object	521
Adding text to a DataWindow object	522
Adding drawing objects to a DataWindow object	523
Adding a groupbox to a DataWindow object	523
Adding pictures to a DataWindow object	524
Adding computed fields to a DataWindow object	525
Adding buttons to a DataWindow object	532
Adding graphs to a DataWindow object	536
Adding OLE objects to a DataWindow object	536
Adding reports to a DataWindow object	537
Positioning objects in a DataWindow object	538
Storing data in a DataWindow object	539
What happens during execution	540
Retrieving rows as needed	541
Saving retrieved rows to disk	542
Controlling updates	543
What you can do	543
Specifying the table to update	544
Specifying the unique key columns	545
Specifying an identity column	545

	Specifying updatable columns.....	546
	Specifying the WHERE clause for update/delete	546
	Specifying update when key is modified	549
16	Displaying and Validating Data	551
	About displaying and validating data.....	552
	Presenting the data	552
	Validating data.....	553
	Working with display formats	554
	Using display formats.....	555
	Defining display formats.....	559
	Working with edit styles.....	566
	Using edit styles.....	567
	Defining edit styles	570
	Defining a code table.....	578
	Working with validation rules.....	582
	Understanding validation rules	582
	Using validation rules	583
	Defining validation rule properties in the Database painter.....	584
	Defining a validation rule in the DataWindow painter.....	588
	Maintaining the entities	591
17	Filtering, Sorting, and Grouping Rows	593
	Filtering rows.....	594
	Sorting rows	597
	Suppressing repeating values	598
	Grouping rows.....	600
	Using the Group presentation style	602
	Defining groups in an existing DataWindow object	606
18	Highlighting Information in DataWindow Objects	617
	Overview of highlighting information	618
	Conditionally modifying properties at execution time	622
	Example 1: creating a gray bar effect	624
	Example 2: rotating objects.....	627
	Example 3: highlighting rows of data	629
	Example 4: changing the size and location of objects	632
	Supplying property values.....	635
	Background.Color.....	636
	Border.....	637
	Brush.Color	638

	Brush.Hatch.....	638
	Color.....	641
	Font.Escapement (for rotating objects)	641
	Font.Height.....	644
	Font.Italic.....	645
	Font.Strikethrough.....	646
	Font.Underline.....	647
	Font.Weight.....	647
	Format.....	648
	Height.....	649
	Pen.Color.....	649
	Pen.Style.....	650
	Pen.Width.....	651
	Pointer.....	652
	Visible.....	653
	Width.....	654
	X.....	654
	X1, X2.....	654
	Y.....	655
	Y1, Y2.....	655
	Specifying colors.....	657
19	Using Nested Reports	659
	About nested reports.....	660
	Creating a report using the Composite presentation style	664
	Placing a nested report in another report.....	667
	Placing a related nested report in another report.....	667
	Placing an unrelated nested report in another report.....	671
	Working with nested reports.....	672
	Adjusting nested report width.....	673
	Changing a nested report from one report to another.....	673
	Modifying the contents of a nested report.....	673
	Adding another nested report to a composite report.....	674
	Supplying retrieval arguments to relate a nested report to its base report.....	675
	Specifying criteria to relate a nested report to its base report.....	676
	Using options for nested reports.....	678
20	Working with Graphs.....	681
	Overview of graphs.....	682
	Parts of a graph.....	682
	Types of graphs.....	684

Using graphs in applications	688
Using graphs in DataWindow objects	689
Placing a graph in a DataWindow object.....	689
Using the graph's property sheet.....	691
Changing a graph's position and size.....	692
Associating data with a graph	694
Using overlays.....	705
Using the Graph presentation style.....	707
Defining a graph's properties	708
Using the Graph property page of the graph's property sheet.....	708
Naming a graph.....	709
Defining a graph's title.....	709
Specifying the type of graph.....	710
Using legends.....	710
Sorting data	710
Specifying text properties for titles, labels, axes, and legends	711
Specifying overlap and spacing.....	716
Specifying axis properties	716
Specifying a border	721
Specifying a pointer.....	721
Specifying point of view in 3D graphs	722
Using graphs in windows	723
Placing a graph in a window.....	723
Using the graph's property sheet.....	724

21

Working with Crosstabs.....	725
About crosstabs	726
Two types of crosstabs.....	728
Creating crosstabs	730
Associating data with a crosstab.....	731
Specifying the information.....	731
What happens	733
Specifying more than one row or column	735
Previewing crosstabs	737
Enhancing crosstabs.....	738
Specifying basic properties	738
Modifying the data associated with the crosstab	739
Changing the names used for the columns and rows	740
Defining summary statistics.....	741
Cross-tabulating ranges of values.....	744
Creating static crosstabs	747
Using property conditional expressions.....	748

22	<p>Working with Rich Text 751</p> <ul style="list-style-type: none"> What is rich text? 752 Using the RichText presentation style 753 <ul style="list-style-type: none"> Creating the DataWindow object 754 Formatting for RichText objects within the DataWindow object 758 Previewing and printing 763 Using the RichTextEdit control 765 <ul style="list-style-type: none"> Creating a RichTextEdit control 765 Specifying the properties of a RichTextEdit control 765 Making a RichTextEdit control read-only 765 Controlling the appearance of a RichTextEdit control 766 Enabling the popup menu 767 Formatting keys and toolbars 768
23	<p>Using OLE in a DataWindow Object 771</p> <ul style="list-style-type: none"> OLE support in DataWindow objects 772 <ul style="list-style-type: none"> About activation 773 OLE objects and OLE presentation style 774 <ul style="list-style-type: none"> Adding an OLE object to the DataWindow object 775 Using the OLE presentation style 776 Defining the OLE object 776 Previewing the DataWindow object 779 Specifying data for the OLE object 781 Using OLE columns in a DataWindow object 785 <ul style="list-style-type: none"> Creating an OLE column 786
24	<p>Working with Data Pipelines 793</p> <ul style="list-style-type: none"> About data pipelines 794 <ul style="list-style-type: none"> Defining a data pipeline 795 Piping extended attributes 796 Creating a data pipeline 798 Modifying the data pipeline definition 801 <ul style="list-style-type: none"> Choosing a pipeline operation 803 Dependency of modifications on pipeline operation 804 When execution stops 805 Piping blob data 808 Changing the destination and source databases 809 Correcting pipeline errors 811 Saving a pipeline 813 Using an existing pipeline 814 Pipeline examples 815

PART 5

RUNNING YOUR APPLICATION

25	Debugging and Running Applications	819
	Overview of debugging and running applications	820
	Debugging an application.....	821
	Using the Debug window.....	821
	Setting breakpoints.....	827
	Running in debug mode	831
	Examining an application at a breakpoint.....	833
	Stepping through an application.....	839
	Debugging windows opened as local variables.....	841
	Just-in-time debugging.....	842
	Running an application.....	844
	Running the application.....	844
	Handling errors during execution	844
	Tracing and profiling an application	851
	The Application Profiler	852
	Collecting trace information.....	853
	Analyzing trace information	860
	Generating a trace file without timing information	870
 26	 Creating an Executable	 873
	Overview of creating an executable	874
	Defining a project	875
	About the Project painter.....	875
	Defining a project object.....	876
	Project painter options.....	878
	Using dynamic libraries	883
	Specifying the dynamic libraries in your project	884
	Including additional resources for a dynamic library	884
	Building a project.....	885
	How PowerBuilder builds the project.....	886
	How PowerBuilder searches for objects.....	886
	Listing the objects in a project.....	889
	Distributing resources	891
	Distributing resources separately	891
	Using PowerBuilder resource files	892
	Creating the PowerBuilder resource file.....	893
	Macintosh resources	895
	What happens at execution time	900
	Tracing execution	901

APPENDIX	The Powersoft Repository	905
	About the Powersoft repository	906
	The PBCatTbl table	907
	The PBCatCol table	909
	The PBCatFmt table	910
	The PBCatVld table	911
	The PBCatEdt table	912
	Available edit style types	912

About This Book

Subject

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

Audience

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

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

PART 1

The PowerBuilder Environment

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

Working with PowerBuilder

About this chapter

This chapter describes the basics of working with PowerBuilder and its painters.

Contents

Topic	Page
What is PowerBuilder?	4
The PowerBuilder environment	7
Painter summary	10
Using online Help	12
Using popup menus	14
Using property sheets	16
Using toolbars	18
Using the file editor	27
Changing fonts	31
Defining colors	32
Managing the initialization file	33
Building an application	37
Starting PowerBuilder from the command line	39

Before you begin

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

What is PowerBuilder?

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

What's in a PowerBuilder application?

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

Cross-platform development

PowerBuilder supports cross-platform development and deployment. For example, you can develop an application using PowerBuilder under Windows (either Windows 95 or Windows NT) and deploy the very same application—without changes—on a Windows 3.1, Macintosh, or UNIX machine. You can even have a cross-platform team of developers, some using Windows and some using the Macintosh, developing the same application at the same time. They can freely share PowerBuilder objects used in the application, because the objects are the same across the different computing platforms that PowerBuilder supports.

Most of the figures in this book show PowerBuilder running on Windows 95, but the PowerBuilder graphical user interface looks and works much the same on all platforms.

FOR INFO For more information about cross-platform development using PowerBuilder, see *Application Techniques*.

Internet development

PowerBuilder includes a number of tools that let you build web-based applications and extend your existing applications to the Internet.

FOR INFO For more information about Internet development using PowerBuilder, see *Building Internet Applications with PowerBuilder*.

About the painters

You build the components of your application using **painters**, which provide an assortment of tools for building objects. PowerBuilder provides a painter for each type of object you build.

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

About events and scripts

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

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

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

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

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

About functions

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

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

About libraries

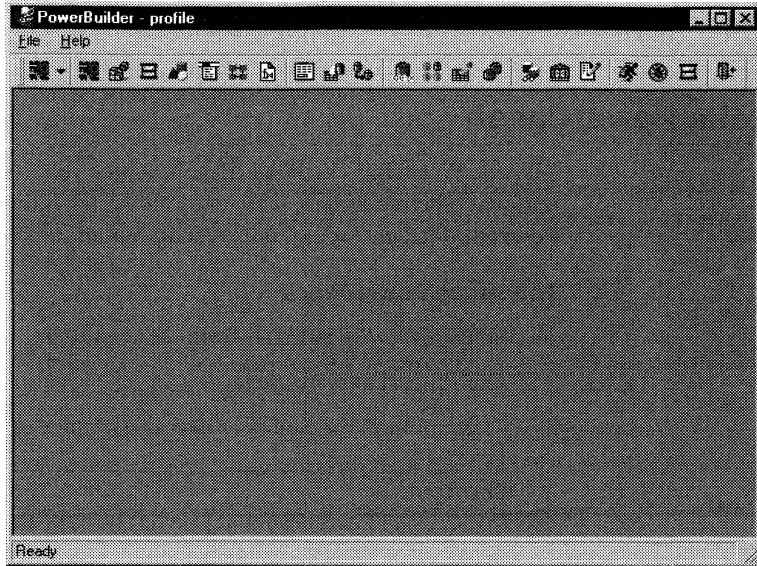
You save your objects, such as windows and menus, in PowerBuilder libraries (PBL files). When you run your application, PowerBuilder retrieves the objects from the library. PowerBuilder provides a Library painter for you to manage your libraries.

Creating an executable

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

The PowerBuilder environment

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



You use PowerBuilder painters to create the windows, menus, database tables, and other objects you need in your application. You can open painters and perform other tasks by clicking buttons in the PowerBar.

About the PowerBar

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

Customizing the PowerBar

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

FOR INFO For more information, see "Using toolbars" on page 18.

About PowerTips

By default, PowerBuilder displays a brief description of the button, called a **PowerTip**, when you leave the mouse pointer over a button for a second or two.

About the PowerPanel

Like the PowerBar, the PowerPanel enables you to open painters and tools and perform other activities. It contains all the tools that are available throughout PowerBuilder, including tools that are not in the default PowerBar.

❖ **To use the PowerPanel:**

- 1 Select File>PowerPanel from the menu bar.
The PowerPanel displays.
- 2 Click an item on the list to access a painter or tool and click OK.

Shortcut

To access a PowerPanel item quickly, type the first letter or two of the item name. The PowerPanel jumps to that item immediately. For example, to jump to the System Options item, type S.

About the
PowerPanel
dropdown toolbar

The PowerPanel dropdown toolbar offers a quick way to access most PowerPanel items. When PowerBuilder first opens, the two buttons at the left of the PowerBar both open the Application painter. The button on the left has a down arrow next to it. Click the arrow to display the PowerPanel dropdown toolbar:



Whenever you click a button in either the PowerBar or the PowerPanel dropdown toolbar, that button replaces the leftmost button in the PowerBar.

Painter summary

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

Painter or tool	What you do
Application painter	Specify information about your application, such as its name and the PowerBuilder libraries in which the application's objects will be saved
Project painter	Create your executable by specifying the components that go into the application
Window painter	Build the windows that will be used in the application
User Object painter	Build custom objects that you can save and use repeatedly in windows
Menu painter	Build menus that the windows will use
Structure painter	Define structures (groups of variables) for use in your application
Function painter	Build functions to perform processing specific to your application
DataWindow painter	Build intelligent objects called DataWindow objects that present information from the database
Report painter	Build and preview reports (DataWindow objects without update capability)
Run Report	Preview reports
Query painter	Graphically define and save SQL SELECT statements for reuse with DataWindow objects and reports
Data Pipeline painter	Transfer data from one data source to another
Configure ODBC	Define a data source that uses ODBC
Database Profiles	Define and use named sets of parameters to connect to a particular database
Table painter	Create database tables, alter existing tables, and define keys, indexes, and relationships between tables
Database painter	Maintain databases, control user access to databases, and manipulate data in databases

Painter or tool	What you do
Database Administration painter	Perform database administration tasks, such as maintaining users and security
Browser	View information about system objects and objects in your application, such as properties, events, functions, and global variables, and copy, export, or print it
Library painter	Create and maintain libraries of PowerBuilder objects
File Editor	Edit text files such as source, resource, and initialization files
Run	Run your current PowerBuilder application just as your users would run it
Debug	Set breakpoints and watch expressions, step through your application, examine and change variables during execution, and view the call stack and objects in memory
Run window	Run a single window in your application
System options	Set PowerBuilder preferences such as initialization path, fonts, and profiling preferences
Help	Invoke the PowerBuilder online Help system to give you quick answers to questions

Opening a painter or tool

There are several ways to open a painter or tool:

From here	You can
PowerBar	Click the button for the painter or tool
PowerPanel	Select the painter or tool
PowerPanel dropdown toolbar	Click the button for the painter or tool
Library painter or Browser	Double-click an object to open it and start the corresponding painter
File menu	Select one of the last four objects you've worked on—they are listed at the bottom of the File menu
Anywhere	Use shortcut keys to open a painter or tool directly—shortcut keys are listed in the PowerPanel

Using online Help

PowerBuilder has online Help that provides both reference and task-oriented information.

How to access Help

You can get Help in any of these ways:

Approach	What it does
Use the Help menu on the menu bar	Displays the Help contents, the Welcome to PowerBuilder window, or Help for the current painter
Click the Help button on the PowerPanel (you can add the Help button to the PowerBar if you want to)	Displays the Help contents
Press F1	Displays the Help contents
Press SHIFT+F1 in the PowerScript painter or Function painter	Displays context-sensitive Help about the object under the cursor
Select Help from the popup 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

Learning to use online Help

To get information on using Help, press F1 anywhere within online Help.

Using the popup menu

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

To display the popup menu in online Help, click the right mouse button on Windows and UNIX, and press the **OPTION** key and click the mouse button on Macintosh.

Linking to the online books

Some Help topics provide links you can click to go to the Powersoft Online Books. These links are represented by a picture of a CD.

FOR INFO For information about installing the Powersoft Online Books and reading them from a CD or from a server, see the *Installation Guide*.

FOR INFO For information about searching and using the annotation features in the Powersoft Online Books, see the *Using the Powersoft Online Books* collection in the Powersoft Online Books.

Online books on the Macintosh

Powersoft Online Books on the Macintosh are available through an external folder only.

❖ **To view online books on the Macintosh:**

- 1 Open the folder where you installed the Powersoft Online Books.
- 2 Open the Online Book folder.
- 3 Double-click the PowerBuilder Online Book alias.

Online books on AIX and HP-UX

Powersoft Online Books on AIX and HP-UX are not available from the online Help.

❖ **To view online books on AIX or HP-UX:**

- ◆ Double-click the PBBooks icon in the folder where you installed the Powersoft Online Books.

or

Enter the following command in a shell:

```
pbbooks &
```

Using popup menus

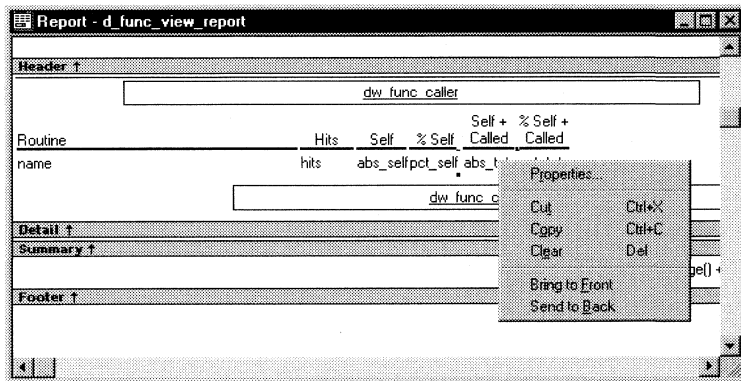
PowerBuilder provides a context-sensitive popup menu that lists:

- ◆ Actions appropriate to the currently selected object or the current position of the pointer
- ◆ Where appropriate, a Properties menu item for accessing the property sheet associated with the current object or the current position of the pointer

The popup menu is available almost everywhere in PowerBuilder.

Example

For example, the following screen shows the popup menu for a column in a report:



❖ To display a popup menu:

- 1 Select an object, or position the pointer on an object or in an open area of the workspace.
- 2 Click the right mouse button.

On Macintosh

On the Macintosh, press the CONTROL key while you click the mouse.

The appropriate popup menu displays.

- 3 Click the menu option you want.

Working with multiple objects

If you want to perform an action on multiple objects, you can select them (you can use lasso selection to do this) and display the popup menu.

When multiple objects are selected, you *cannot* use Properties from the popup menu to change their characteristics. The Properties menu item only works on single objects. You *can* use a painter's StyleBar to change properties for multiple objects.

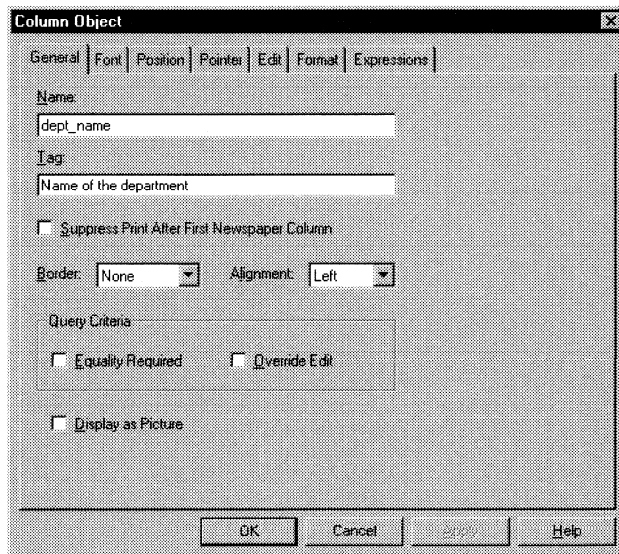
FOR INFO For more information about working with multiple objects, see "Selecting objects in the DataWindow painter" on page 474.

Using property sheets

A **property sheet** is a tab dialog box you use to set properties associated with an object, painter, or tool by making changes in one or more tabs in the dialog box.

Example

For example, for a column in a report, you can set several different kinds of properties (general, font, position, pointer, edit style, display format, and conditional expressions) by clicking appropriate tabs in the Column Object property sheet:



Property sheet buttons

Each property sheet has OK, Cancel, Apply, and Help buttons. The Apply button is enabled when you make a change on one tab:

Use this button	To do this
OK	Apply the properties you've set on all tabs and close the property sheet
Cancel	Close the window and apply no new changes
Apply	Apply the properties you've set on all tabs immediately without closing the property sheet
Help	Get Help on setting properties for the tab that displays

Displaying property sheets

You can display property sheets in four ways (the first two are usually faster):

- ◆ By selecting Properties from the popup menu of an object, painter, or tool
- ◆ By double-clicking an object
- ◆ By selecting Object>Properties or Edit>Properties from the menu bar (depending on the painter you are working in)
- ◆ By clicking the Properties button in the PainterBar

Using toolbars

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

Toolbar basics

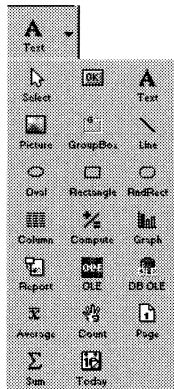
PowerBuilder uses three toolbars: the PowerBar, PainterBar, and StyleBar:

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

Dropdown toolbars

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

For example, the down arrow next to the Text button in the DataWindow painter displays the Objects dropdown toolbar, which has a button for each object you can place on a DataWindow object:



Default button replaced

The button you select from a dropdown toolbar replaces the default button on the main toolbar. For example, if you select the Picture button from the Objects dropdown toolbar, it replaces the Text button in the PainterBar.

Controlling the display of toolbars

You can control:

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

Choosing to display text and PowerTips affects all toolbars.

❖ To control a toolbar using the popup menu:

- 1 Position the pointer on the toolbar and display the popup menu.
- 2 Click the items you want.

A checkmark means the item is currently selected.

❖ To control a toolbar using the Toolbars dialog box:

- 1 Select Window>Toolbars from the menu bar. (If no painter is open, select File>Toolbars from the menu bar.)

The Toolbars dialog box displays.

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

PowerBuilder saves your toolbar preferences in the PowerBuilder initialization file.

Moving toolbars using the mouse

You can use the mouse to move a toolbar.

❖ To move a toolbar with the mouse:

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

- 2 Press and hold the left mouse button.
- 3 Drag the toolbar and drop it where you want it.

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

Docking toolbars

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

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

Example

These two toolbars are on separate docks:



You could customize the toolbars to reduce their size, then dock them side by side:



Customizing toolbars

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

Adding, moving, and deleting buttons

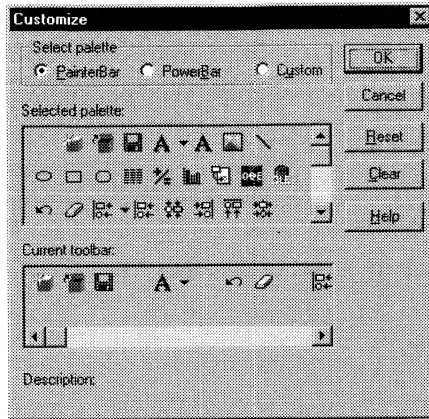
You can add, move, and delete buttons in any toolbar.

❖ To add a button to a toolbar:

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

- 2 Select Customize.

The Customize dialog box displays.



- 3 Click the palette of buttons you want to use in the Select palette group.
- 4 Choose a button from the Selected palette 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 INFO For more information, see "Adding a custom button" on page 23.

Seeing what's available in the PowerBar

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

❖ To move a button on a toolbar:

- 1 Position the pointer on the toolbar, display the popup 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 popup 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 popup 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 don't add new buttons to the empty toolbar, the toolbar is deleted. You can delete both built-in toolbars and toolbars you've created.

To recreate a toolbar

If you delete one of PowerBuilder's built-in toolbars, you can recreate it easily. For example, to recreate the PowerBar, select PowerBar1 in the New Toolbar dialog box.

FOR INFO For information about creating new toolbars and about the meaning of PowerBar1, see "Creating new toolbars" on page 25.

❖ **To clear or delete a toolbar:**

- 1 Position the pointer on the toolbar, display the popup 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 Select new buttons for the current toolbar and click OK.
or
Click OK to delete the toolbar.

Adding a custom button

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

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

❖ **To add a custom button:**

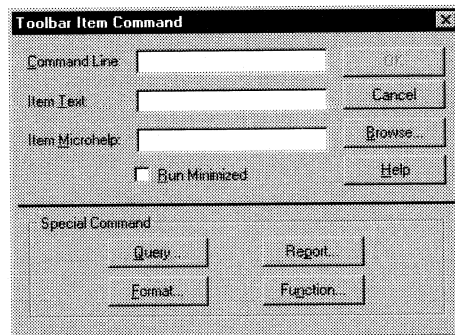
1 Position the pointer on the toolbar, display the popup menu, and select Customize.

2 Select Custom in the Select Palette group.

The custom buttons display in the Selected Palette box.

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

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



4 Fill in the dialog box as follows:

To have the button	Do this
Invoke a PowerBuilder menu item	Type @MenuBarItem.MenuItem in the Command Line box. For example, to have the button mimic the Open item on the File menu, type @File.Open You can also use a number to refer to a menu item. The first item in a dropdown/cascading menu is 1, the second item is 2, and so on. Separator lines in the menu count as items. Example: @Edit.Align Controls.5
Run an executable outside PowerBuilder	Type the name of the executable in the Command Line box. Specify the full path name if the executable is not in the current search path To search for the filename, click the Browse button
Run a query	Click the Query button and select the query from the displayed list
Run a report (same as previewing a DataWindow object)	Click the Report button and select a report (DataWindow) from the displayed list. You can then specify command-line arguments in the Command Line box, as described below
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 or report	(DataWindow and Report painters 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 INFO For more about specifying display formats, see Chapter 16, "Displaying and Validating Data"
Create a computed field in a DataWindow object or report	(DataWindow and Report painters only) Click the Function button to display the Function for Toolbar dialog box. Select the function from the list

- 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.

Supplying arguments with reports

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

Argument	Meaning
<i>/l LibraryName</i>	Specifies the library containing the report
<i>/o ReportName</i>	Specifies the report
<i>/r</i>	Runs the report
<i>/ro</i>	Runs the report but does not provide design mode for modifying the report
<i>/a "Arguments"</i>	Specifies arguments to pass to the report

The default command line is:

```
Report /o ReportName /ro
```

Modifying a custom button

❖ To modify a custom button:

- 1 Position the pointer on the toolbar, display the popup 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 23.

Creating new toolbars

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

PowerBars and PainterBars

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

For this toolbar type	The default is named	And you can have up to
PowerBar	PowerBar1	4 PowerBars
PainterBar	PainterBar1	8 PainterBars in each painter

Where you create them

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

❖ **To create a new toolbar:**

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

The New Toolbar dialog box displays.

About the StyleBar

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

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

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

- 3 One at a time, drag the toolbar buttons you want from the Selected palette box to the Current toolbar box.

Using the file editor

PowerBuilder provides a text editor that is always available. Using the editor, you can view and modify text files (such as initialization files and tab-separated files with data) without leaving PowerBuilder.

❖ **To open the file editor:**

- 1 Press SHIFT+F6 anywhere in PowerBuilder.
or
Click the Edit button in the PowerBar.

Adding an Edit button

If there is no Edit button on the PowerBar, you can add one. The button is available from the PowerBar palette.

FOR INFO For more information, see "Customizing toolbars" on page 20.

The File Open dialog box displays.

- 2 Open the file you want to edit.
or
Click Cancel to display an empty editing workspace.

Setting file editing properties

The file editor has Font and Indentation properties that you can change to make files easier to read. Select Design>Options from the menu bar to open the properties sheet. If you don't change any properties, files have black text on a white background and a tab stop setting of 3 for indentation.

❖ **To specify File Editor properties:**

- 1 Select Design>Options to display the property sheet.
- 2 Choose the tab appropriate to the property you want to specify.

Editor properties apply elsewhere

When you set properties for the file editor, the settings also apply to the Function, Script, and Database Administration painters and the Debug window.

Editing activities

The file editor provides a full set of basic editing facilities including:

- ◆ Opening, saving, and printing files
- ◆ Cutting, copying, pasting, and clearing selected text
- ◆ Finding and replacing text
- ◆ Undoing changes
- ◆ Commenting and uncommenting lines
- ◆ Importing and exporting text files
- ◆ Dragging and dropping text

Using the file editor's PainterBar and menu bar

The file editor has a PainterBar that provides a shortcut for performing frequently used activities. There is also a corresponding menu item (and often a shortcut key) for each activity:

Menu item	Shortcut key	Activity
Edit>Undo	CTRL+Z	Undoes the most recent edit
Edit>Cut	CTRL+X	Cuts selected text to the clipboard
Edit>Copy	CTRL+C	Copies selected text to the clipboard
Edit>Paste	CTRL+V	Pastes the contents of the clipboard at the current cursor location; replaces any selected text
Edit>Clear	DELETE	Deletes selected text; does not place the text in the clipboard
Edit>Select All	CTRL+A	Selects all text in the workspace
Edit>Comment Selection	—	Comments out the current line or all lines containing selected text by inserting two slashes before the first character in each line
Edit>Uncomment Selection	—	Uncomments the current line or all lines containing selected text by removing the two slashes before the first character in each line

Menu item	Shortcut key	Activity
Search>Find	CTRL+F	Specifies a string for which you want to search
Search>Find Next	CTRL+G	Finds the next occurrence of the specified search string
Search>Replace	CTRL+H	Replaces the specified search string
Search>Go to Line	CTRL+SHIFT+G	Goes to a specific line number

On Macintosh On the Macintosh, use the COMMAND key instead of the CTRL key.

On UNIX On UNIX systems, you can also use the middle mouse button and dedicated editing keys on your keyboard for copying and pasting text if your window manager supports them.

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.

On Macintosh

On the Macintosh, use the COMMAND key instead of the CTRL key.

Executing AppleScript scripts on the Macintosh

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

❖ To open an AppleScript file for execution:

- 1 Open the file editor.
- 2 Do one of the following:
 - ◆ Select File>Open from the menu bar and select a source file containing AppleScript.

- ◆ Type the name of a file in the empty file editor (do not enclose the filename in quotation marks). The file can be either a source file containing AppleScript or a compiled AppleScript script.
- ◆ Create a new script by typing AppleScript commands in the file editor. You can save the new script using File>Save from the menu bar.

❖ **To execute an AppleScript script:**

- ◆ Select Edit>Execute AppleScript from the menu bar.

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

PowerBuilder displays a dialog box with the results.

Executing AppleScript scripts during execution

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

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

Changing fonts

The following table summarizes the various ways you can change the fonts used in PowerBuilder:

For this object or painter	Do this
A table's data, headings, and labels	In the Database or Table painter, display the table's property sheet, and change the font properties on the Data, Heading, and Label Font tabs
Objects in the Report, Window, and DataWindow painters	Select objects and then modify settings in the StyleBar <i>or</i> Select an object, display its property sheet, and change the font properties on the Font tab
Application, Menu, and Library painters, Browser, and MicroHelp	Click the System Options button on the PowerPanel and change the font properties on the Font tab
Function, Script, and Database Administration painters and the file editor and Debug window (changes made for one of these apply to all)	Select Design>Options from the menu bar to display the editor's property sheet and change the font properties on the Font tab. In the Debug window, select Debug>Options

Changes you make in the System Options dialog box and from the Design>Options menu selection are reflected in the PowerBuilder initialization file and are used the next time you open PowerBuilder.

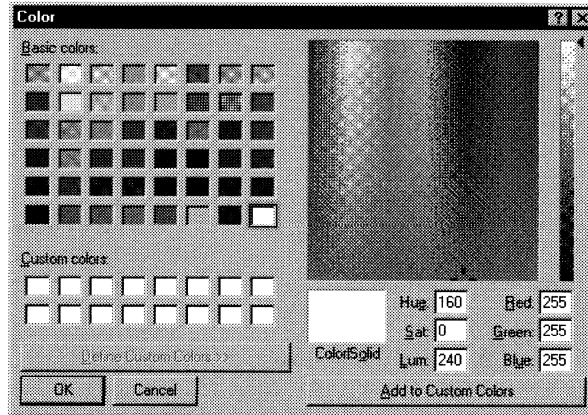
Defining colors

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

❖ **To define custom colors:**

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

The Color dialog box displays:



- 2 Define your custom colors:

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

Managing the initialization file

When you start PowerBuilder, it looks for the PowerBuilder initialization file to set up your environment.

About the initialization file

The initialization file is a text file that contains variables that specify your PowerBuilder preferences. These preferences include things such as the last database you connected to, the PBL you are using, and your toolbar settings. When you perform an action in PowerBuilder, PowerBuilder writes your preferences to the initialization file automatically.

Specifying preferences

Normally, you don't need to edit the initialization file. 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. But sometimes a variable doesn't appear by default in the options sheet for the painter. So if you don't see a variable whose value you want to change, use a text editor such as the PowerBuilder file editor to add the variable to the appropriate section of the initialization file.

Editing the initialization file

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

Format of INI files

The PowerBuilder initialization file uses the Windows INI file format on all platforms. It has three types of elements:

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

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

Default sections

The sections in the initialization file and the variables in each section can be in any order, but the variables that belong to a particular section must be in that section.

Here are some of the sections and what they contain:

Section	What it contains
Application	The name and location of the current application and PowerBuilder library, and a history of previous applications
PB	Basic toolbar, window size, and code generation preferences as well as the names of the most recently opened objects
Database	The current database profile, the list of available DBMSs, and other Database painter preferences
DBMS_Profiles	The name of the current database profile and a list of previous profiles
Profile <i>name</i>	The database profile called <i>name</i> . An additional Profile section is created for each database profile you define
Debug	The current Debug window layout, any saved layouts, and the current breakpoints and watch expressions

Some sections are always present by default, but others are created only when you specify different preferences. The default initialization file may contain a section for specific painters: for example, the Data Window section contains preferences for the DataWindow painter and Report painter, and the SQL Painter section contains preferences for the SQL Select painter.

If you specify preferences for another painter or tool, PowerBuilder creates a new section for it at the end of the file. Customized toolbar layouts are also saved in separate Toolbar sections at the end of the file.

Where the initialization file is kept

The default PowerBuilder initialization file has different names and is stored in different locations on each platform:

Platform	Name	Default location of the initialization file
Windows	PB.INI	The directory where PowerBuilder is installed
UNIX	.pb.ini	Your home directory. You also need a copy of the .WindU initialization file in your home directory
Macintosh	PowerBuilder Preferences	System Folder:Preferences:Powersoft 6.0 Preferences

Telling PowerBuilder where your initialization file is

On Windows and UNIX, you can keep your initialization file in another location and tell PowerBuilder where it can find it by specifying the location in the System Options tab dialog. You may want to do this if you use more than one version of PowerBuilder or if you are running PowerBuilder over a network.

On Macintosh

On the Macintosh, changing the initialization path in the System Options dialog box has no effect.

❖ **To record your initialization path:**

- 1 Open the PowerPanel and click the System Options button.

The Systems Options dialog box displays.

- 2 Enter the path of your initialization file in the Initialization Path textbox.

On Windows, PowerBuilder records the path in the Windows registry.

On UNIX, it records it in a file called reg.dat in your home directory.

How PowerBuilder finds the initialization file

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

On UNIX, PowerBuilder looks in the reg.dat file in your home directory for a path to the file. If it doesn't find one, it looks for the file in your home directory and then in the directory where PowerBuilder is installed.

On the Macintosh, PowerBuilder looks in System Folder:Preferences: Powersoft 6.0 Preferences.

If the initialization file is missing

If PowerBuilder doesn't find the initialization file in the default location 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 your initialization file. The recreated file has the default preferences.

Using the .WindU initialization file on UNIX

The .WindU file is an initialization file that is unique to PowerBuilder for UNIX. This file contains settings for **Wind/U**, a product from Bristol Technology used to implement PowerBuilder in the UNIX Motif environment. Like .pb.ini, .WindU resides in your home directory.

The organization and content of the .WindU file correspond closely to the WIN.INI file in Microsoft Windows. You'll find many of the same sections, such as [devices] and [ports]. Each section has one or more entries with appropriate settings assigned to them. The .WindU file also contains some unique sections of its own.

Changing .WindU settings

For most of the sections in the .WindU file, the default settings provided should meet your needs. If you do want to change any of them, use any text editor.

Making changes in .WindU can give you control over such things as:

- ◆ Printer configuration

Alternative

If you don't want to edit the .WindU file bookly, you can use the File>Printer Setup menu command in PowerBuilder to specify your printer configuration settings. This displays dialog boxes you can fill in and updates .WindU automatically.

- ◆ Font mapping and performance
- ◆ Common dialog style

FOR INFO For details on what you should change in the .WindU file to meet your particular requirements, look in the online Help file named **windu.hlp** (by opening it from HyperHelp).

Building an application

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

Using other books

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

FOR INFO For an overview of the kinds of applications you can build in PowerBuilder and the building blocks you can use, see the *Feature Guide*.

FOR INFO For complete information about the process of building applications in PowerBuilder, see *Application Techniques*.

❖ To build an application:

1 Create the Application object

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

FOR INFO See Chapter 2, "Working with Applications".

2 Create windows

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

FOR INFO See Part 3, "Working with Windows".

3 Create DataWindow objects

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

FOR INFO See Part 4, "Working with Databases".

4 Create menus

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

FOR INFO See Chapter 10, "Working with Menus".

5 Create user objects

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

FOR INFO See Chapter 11, "Working with User Objects".

6 Create functions and structures

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

FOR INFO See Chapter 5, "Working with User-Defined Functions", and Chapter 6, "Working with Structures".

7 Test and debug your application

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

FOR INFO See Chapter 25, "Debugging and Running Applications".

8 Prepare an executable

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

FOR INFO See Chapter 26, "Creating an Executable".

Starting PowerBuilder from the command line

On Windows and UNIX systems, you can start PowerBuilder from a command line (or the Windows 95 Run dialog box) and optionally open one of the following painters or tools:

Application painter	Library painter
Database painter	Menu painter
Data Pipeline painter	Query painter
DataWindow painter	Report painter
Debug window	Structure painter
File Editor	
Function painter	Window painter

On Windows

To start PowerBuilder and open a painter on Windows, use the following syntax:

```
{ win } directory\pb60.exe /P paintername
```

The use of **win** is required to start PowerBuilder from the MS-DOS prompt.

On UNIX

To start PowerBuilder and open a painter on UNIX, use the following syntax:

```
{ directory } pb60 /P paintername
```

The location of the pb60 executable should be in your PATH environment variable, so that the directory is usually not required.

Opening an object

You can also add one or more of the following optional switches to the command line to open a specific object or create a new one:

```
{/L libraryname} {/O objectname} {/N} {/R} {/RO} {/A arguments}
```

All of these switches must follow */P paintername* on the command line, as shown in the examples after the tables.

Switch	Description
/P	Opens the specified painter
/L	Identifies the library that contains the object you want to open
/O	Identifies the object, such as a report or window, you want to open
/N	Creates a new report

Switch	Description
/R	Runs the DataWindow object or report specified with /O and allows designing
/RO	Runs the DataWindow object or report specified with /O but does not allow designing
/A	Provides arguments for the specified DataWindow object or report

Parameter	Description
<i>directory</i>	The fully qualified name of the directory containing PowerBuilder
<i>paintername</i>	<p>The name of the painter you want to open. The default is the window that displays when you begin a new PowerBuilder session</p> <p>The painter name must uniquely identify the painter. You do not have to enter the entire name. For example, you can enter q to open the Query painter and dat to open the Database painter. If you enter the full name, omit any spaces in the name (enter UserObject and DataPipeline, for example)</p> <p>The painter name is not case sensitive. To open the file editor, you could set <i>paintername</i> to FI or fileeditor</p>
<i>libraryname</i>	The name of the library that contains the object you want to open. The default is the library specified in the DefLib variable in the [PB] section of the PowerBuilder initialization file
<i>objectname</i>	The name of the object you want to open

Examples

The following examples use pb6 to represent the directory where PowerBuilder is installed.

Enter this command at the Windows MS-DOS prompt to start PowerBuilder and open the Database painter:

```
win pb6\pb60.exe /P datab
```

Enter this command in the Windows 95 Run dialog box to start PowerBuilder and open the DataWindow object called d_emp_report in the library master.pbl:

```
pb6\pb60.exe /P dataw /L master.pbl /O d_emp_report
```

Enter this command in the Windows 95 Run dialog box to start PowerBuilder, open the report called sum_report in the Report painter, and run it:

```
pb6\pb60.exe /P report /O sum_report /R
```

Enter this command in a UNIX terminal window to start PowerBuilder and open the menu `m_frame` in the Menu painter:

```
pb60 /P m /O m_frame
```


Working with Applications

About this chapter

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

Contents

Topic	Page
Overview of Application objects	44
Creating a new Application object	45
Working with other Application objects	48
Using the Quick Application feature	49
Looking at an application's structure	50
Specifying application properties	54
Writing application-level scripts	62

Overview of Application objects

An **application** is a collection of PowerBuilder windows that perform related activities. It is what you deliver to your users.

The **Application object** is the entry point into the windows that perform these activities. It is a discrete object that is saved in a PowerBuilder library, just like a window, menu, function, or DataWindow object. When a user runs the application, the scripts you write for events are triggered in the Application object.

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

Events in the Application object

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

When a user ends the 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.

FOR INFO For a more detailed explanation of PowerBuilder Application objects, see *PowerBuilder Getting Started*.

Creating a new Application object

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

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

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

Using an existing Application object

If you want to work with an existing application, see "Working with other Application objects" on page 48.

❖ To create an Application object:

1 Click the Application painter button in the PowerBar or PowerPanel.
The Application object opens and displays in the workspace.

2 Select File>New from the menu bar.

The Select New Application Library dialog box displays.

3 Specify the PowerBuilder library in which you want to store the Application object and click Save.

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

The Save Application dialog box displays.

Putting more than one Application object in a library

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

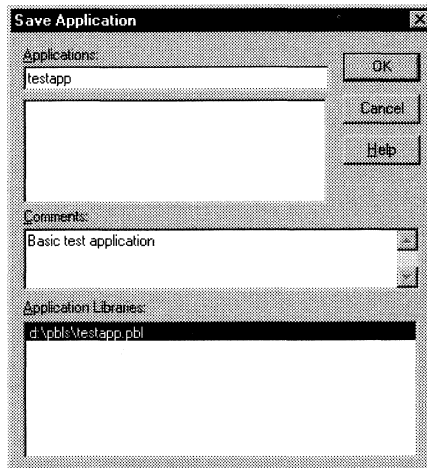
- 4 Name the Application object.

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

Using an existing name

If you create a new application with the same name as an application that already exists in the PBL, PowerBuilder overwrites the existing application. It does not remove any other objects from the PBL, because they may be referenced by another application.

- 5 (Optional) In the Comments box, enter comments to document the application you are building and to help other developers understand the application.



The comments display in the Library painter.

6 Click OK.

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

FOR INFO For more information, see "Using the Quick Application feature" on page 49.

7 Click Yes to generate a template for your application.

or

Click No to start your application from scratch.

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

You have created the Application object and placed it in a PowerBuilder library. You now need to define the Application object's properties.

FOR INFO For how, see "Specifying application properties" on page 54.

Working with other Application objects

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

❖ **To work with another Application object:**

- 1 Open the Application painter.

The last Application object you worked with opens.

- 2 Click the Open button.

or

Select File>Open from the menu bar.

The Select Application Library dialog box displays.

- 3 Select the library containing the Application object and click Open.

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

- 4 Choose the Application object from the list.

You can choose an Application object from a different PowerBuilder library by clicking the Other button.

The chosen Application object opens and displays in the workspace.

Using the Quick Application feature

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

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

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

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

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

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

For more information

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

For more information about MDI frames on the Macintosh, see "Window types on the Macintosh" on page 166.

Looking at an application's structure

Once you have selected an Application object, it displays as an icon in the Application painter workspace. If you are working with an application that references one or more objects in an application-level script, you can look at the application's structure in the Application painter.

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

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

The Application object displays as an icon in the workspace.



- 2 Double-click the icon.

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

FOR INFO For complete information about exactly which objects are shown in the workspace, see "Which objects are displayed" on page 51.



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

Working in the workspace

The Application painter workspace displays referenced objects in an outline format. A popup menu offers shortcuts to working with each displayed object.

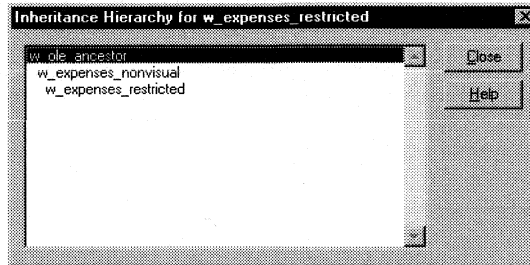
Working with
inherited objects

An asterisk following an object's name in the workspace indicates that the object is a descendant of another object.

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

- ◆ Select Inheritance Hierarchy from the popup menu.

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



Which objects are displayed

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

Which references are displayed

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

Objects that are referenced in painters For example:

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

Objects that are directly referenced in scripts For example:

- ◆ If a window script contains the following statement:

```
Open(w_continue)
```

Then `w_continue` displays when the window is expanded.

Which referenced windows display in the workspace

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

- ◆ If a menu item script refers to the global function `f_calc`:

```
f_calc(EnteredValue)
```

Then `f_calc` displays when the menu is expanded.

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

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

Then `m_new` displays when the window is expanded.

Which references are not displayed

The workspace does not display the following types of references.

Objects that are referenced only through instance variables or properties For example:

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

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

Then `w_emp` does not display as a reference for `w_go`.

Objects that are referenced dynamically through string variables For example:

- ◆ If a window script has the following statements:

```
window mywin
string winname = "w_go"
```



```
Open(mywin,winname)
```

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

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

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

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

Specifying application properties

The Application object specifies the following properties of the application:

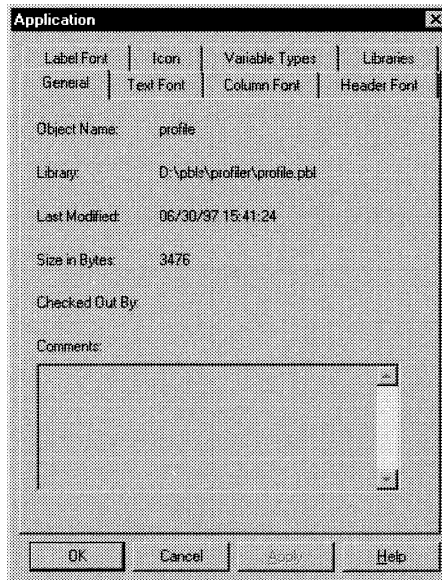
- ◆ "Specifying default text properties" on page 55
- ◆ "Specifying the library search path" on page 57
- ◆ "Specifying an icon" on page 59
- ◆ "Specifying default global objects" on page 60

You specify all these properties in the Application painter property sheets.

Using the application properties tab

❖ To specify application properties:

- 1 Select Properties from the application's popup menu.



- 2 Choose the tab of the property you want to specify:

To specify	Choose this tab
The default font for static text as it appears in windows, user objects, and DataWindow objects	Text Font

To specify	Choose this tab
The default font for data retrieved in a DataWindow object	Column Font
The default font for column headers in tabular and grid DataWindow objects	Header Font
The default font for column labels in freeform DataWindow objects	Label Font
The application icon	Icon
Global objects for your application	Variable Types
An application search path	Libraries

All of these properties are discussed below.

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.

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

Properties set in the Database painter override application properties

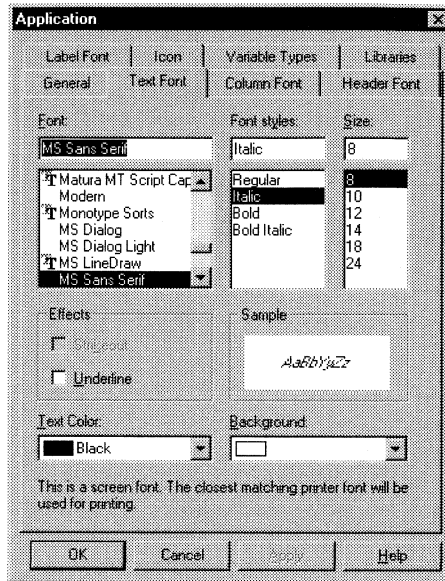
If extended attributes have been set for a database column in the Database painter or Table painter, those font specifications override the fonts specified in the Application painter.

❖ To change the text defaults for an application:

- 1 Select Properties from the application's popup menu 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 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 listbox.
 - ◆ To change the size, select a size from the list in the Size listbox or type a valid size in the listbox.
 - ◆ To change the style, select a style (Regular, Italic, Bold, or Bold Italic) from the Font styles listbox.
 - ◆ 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 listbox. (You don't 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 listbox.

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 the library search path

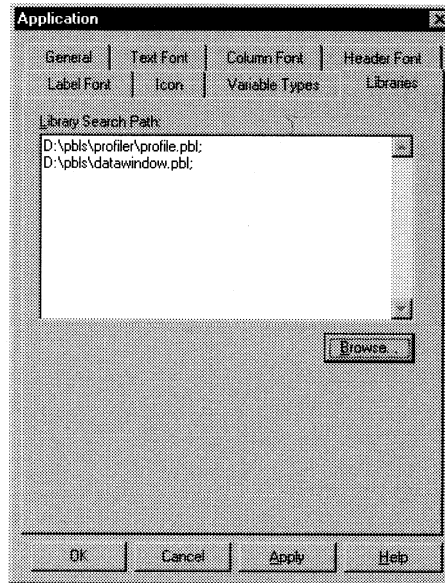
The objects you create in painters are stored in PowerBuilder libraries. You can use objects from one library or multiple libraries in an application. You define each library the application uses in the library search path.

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

❖ **To define a library search path:**

- 1 Select Properties from the application's popup menu and select the Libraries tab.

The Libraries property page displays the current library search path and lists the PowerBuilder libraries in the current directory.



- 2 Enter the name of each library you want to include in the Library Search Path listbox, separating them with semicolons.

or

Use the Browse button to include other libraries in your search path.

Make sure the order is correct

When you select multiple libraries from the Select Library dialog box using SHIFT+click or CONTROL+click, the first library you select appears last in the Library Search Path listbox and will be the last library searched.

To delete a library from the search path, select the library in the listbox and press DELETE.

- 3 Click OK.

PowerBuilder updates the search path for the application.

Where PowerBuilder maintains the library search path

PowerBuilder stores your application's library search path only in your initialization file. *It does not store that information in the Application object.*

There is a line in your initialization file's Application section for each application you have defined; that line lists all libraries in the application's search path. So if you give the PBLs for an application you are building to another PowerBuilder developer, make sure you copy the appropriate line from your initialization file to the other developer's initialization file or that the other developer opens the Application object and specifies the library search path as described above. That way the application's library search path gets defined correctly on the other developer's machine.

For more information

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

The *Application Techniques* book describes these strategies in detail.

Specifying an icon

Users may minimize your application during execution. If you specify an icon in the application painter, the icon will display when the application is minimized.

On Macintosh and UNIX

The icon that you specify in the Application painter does not display when you run your application on the Macintosh or UNIX platforms. If you are creating a cross-platform application on these platforms, you can select an ICO file that will be used when you build your application on Windows.

FOR INFO For information about specifying Macintosh Finder icons for your application, see "Macintosh resources" on page 895.

❖ To associate an icon with an application:

- 1 Select Properties from the application's popup menu and select the Icon tab.
- 2 Specify a file containing an icon (an ICO file).
The button displays at the right of the Icon Name box.
- 3 Click OK to associate the button with the application.

Specifying default global objects

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

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

You can create your own versions of these objects by going to the User Object painter and defining a standard class user object that inherits from one of the built-in global objects. You can add instance variables and functions to enhance the behavior of the global objects.

FOR INFO For more information, see Chapter 11, "Working with User Objects".

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

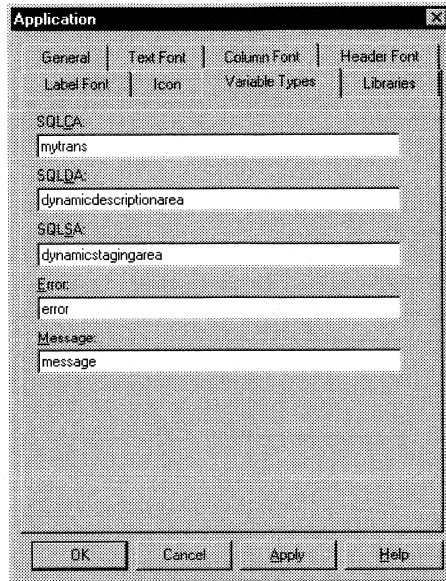
❖ **To specify the default global objects:**

- 1 Select Properties from the application's popup menu and select the Variable Types tab.

The Variable Types property page displays.

- 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`.



- Click OK.

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

Writing application-level scripts

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

Batch applications

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

This table lists all events that can occur in the Application object. The only event that requires a script is Open.

Event	Occurs when
Open	The user starts the application
Close	The user closes the application. Typically, you write a script for this event that shuts everything down (such as closing the database connection and writing out a preferences file)
SystemError	A serious error occurs during execution (such as trying to open a nonexistent window). If there is no script for this event, PowerBuilder displays a message box with the PowerBuilder error number and message text. If there is a script, PowerBuilder executes the script FOR INFO For more about error handling, see "Handling errors during execution" on page 844
Idle	The Idle PowerScript function has been called and the specified number of seconds have elapsed with no mouse or keyboard activity
ConnectionBegin	In a distributed computing environment, a client establishes a connection to a server by calling the ConnectToServer function.
ConnectionEnd	In a distributed computing environment, a client disconnects from a server by calling the DisconnectServer function.

Setting application properties

The Application object has several properties, which specify application-level properties. For example, the property `ToolBarText` specifies whether text displays on toolbars in an MDI application.

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

AppName.property

For example, to specify that text displays on toolbars in the Test application, code this in a script:

```
Test.ToolbarText = TRUE
```

(If the script is in the Application object itself, you don't 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 INFO For a complete list of the properties of the Application object, see *Objects and Controls*.

Managing Libraries

About this chapter

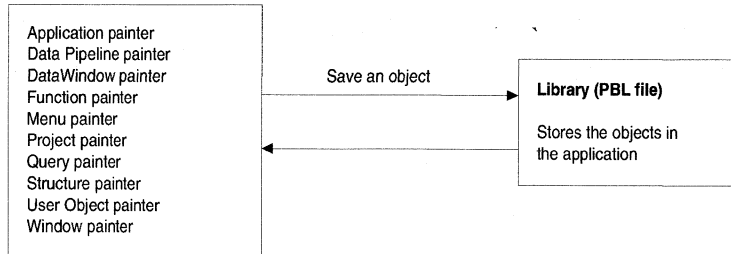
PowerBuilder stores all the objects you create in libraries. When you work with an application, you specify which libraries it will use. This chapter describes how to work with your libraries.

Contents

Topic	Page
Overview of libraries	66
Working with libraries	70
Creating and deleting libraries	75
Copying, moving, and deleting entries	77
Searching library entries	78
Jumping to a painter	80
Browsing the class hierarchy	81
Using check-out and check-in	83
Optimizing libraries	92
Regenerating library entries	93
Exporting and importing entries	96
Creating runtime libraries	100
Creating reports on library contents	102

Overview of libraries

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



Assigning libraries

Applications can use as many libraries as you want. Libraries can be on your own PC or workstation or on a server. When you create an application, you specify which libraries it uses. You can also change the library search path for an application at any time during development.

FOR INFO For more on assigning libraries to an application, see "Specifying the library search path" on page 57.

How the information is saved

Every object is saved in two parts in a library:

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

Using libraries

It is hard to predict the needs of a particular application, so the organization of an application's libraries will probably evolve over the development cycle. PowerBuilder lets you reorganize your libraries easily at any time.

For small applications, you might use only one library. But for larger applications, you will want to split the application into different libraries.

About library size

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

- ◆ **Size** Try to keep your libraries smaller than about 800K. If your libraries are larger, performance can suffer because PowerBuilder has to search more in order to save or open an object.
- ◆ **Number of objects** It is a good idea not to have more than 50 or 60 objects saved in a library. This is strictly for your convenience; the number of objects doesn't affect performance. But if you have many objects in a library, you will find that listboxes that list library objects become unmanageable and that the Library painter becomes more difficult to use.
- ◆ **Balance** You don't want to have to manage a large number of libraries with only a few objects. That makes the library search path too long and can slow performance by forcing PowerBuilder to look through many libraries to find an object. So you should try to maintain a balance between the size and number of libraries.

Organizing libraries

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

A recommended organization

If you are working with other developers on a large application, here is a setup that works well.

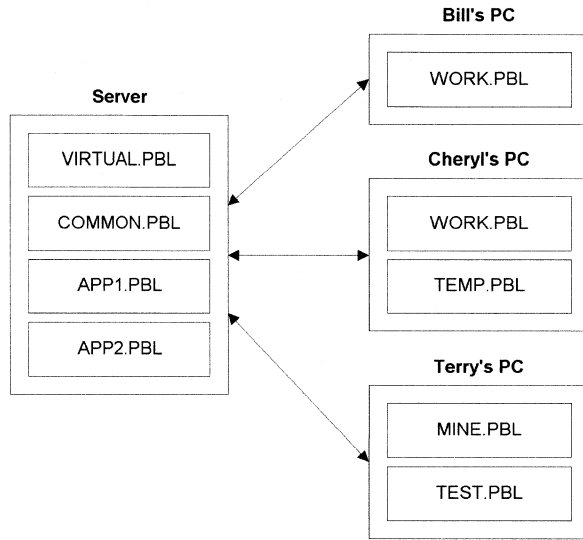
Put libraries containing objects that are shared by developers on a server machine on the network. That way all developers have direct access to them. Put objects that only *you* are working on in a library on your PC or workstation.

Here is an example. The following libraries would be publicly available on a server:

Library	Contents
VIRTUAL.PBL	This library contains all ancestor objects used in the application. For example, if all the windows in your application inherit from w_master, place w_master in this library
COMMON.PBL	This library contains all objects that are used across applications, such as user objects and functions

Library	Contents
APP1.PBL	This library contains objects specific to application 1
APP2.PBL	This library contains objects specific to application 2

Also, each developer would have one or more private libraries where they would keep the objects they are working on.



Sharing objects with others

PowerBuilder provides check-out/check-in facilities that let you check an object out of one library, such as APP1.PBL, and store a working copy in another library, such as your private library. While you have the object checked out, no one else can modify it. When you finish updating an object, you can check the object back in to the public library.

Ordering the application's library search path

If you use the scenario described above with public and private libraries, you should place the private libraries first in the library search path. Then when you check an object out of a public library and place it in your private library, PowerBuilder will find the private one first when executing the application. When you check the object back in to the public library, it is removed from the private library and PowerBuilder finds the updated public version when executing the application.

FOR INFO For more about check-out/check-in, see "Using check-out and check-in" on page 83.

Specifying the library search path

You specify an application's library search path in the Application painter.

FOR INFO For more information, see "Specifying the library search path" on page 57.

Working with libraries

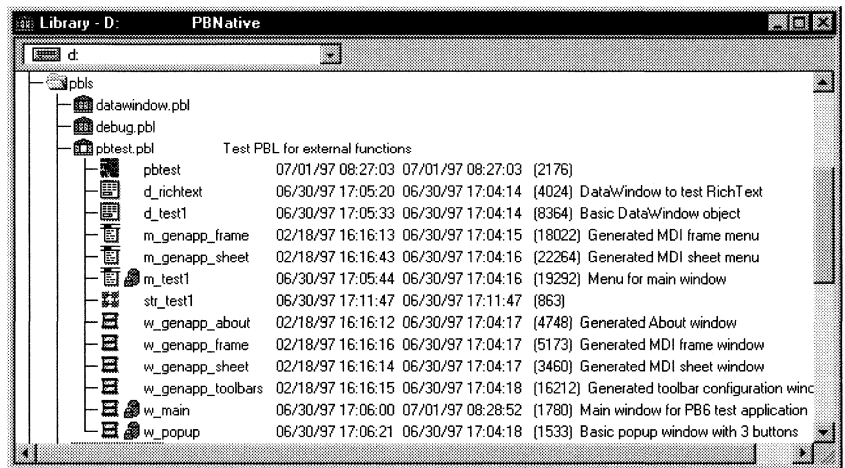
You work with libraries in the Library painter.

❖ To open the Library painter:

- ◆ Click the Library painter button in the PowerBar or PowerPanel.

The Library painter displays.

When you open the Library painter, it lists all directories on the current drive. It expands the current directory and expands the current library (the most recently used library) to show its entries.



Viewing the tree

You can expand (display the contents of) libraries and directories. You can also collapse the display of libraries and directories.

❖ To expand a library or directory:

- ◆ If the library or directory is currently collapsed, double-click the library or directory.

PowerBuilder displays all objects stored in the selected library or the files and subdirectories in the selected directory.

Each entry in a library has an icon that identifies the painter in which the entry was created. The icons are the same as those used in the PowerBar.

- ❖ **To collapse a library or directory:**
 - ◆ If the library or directory is currently expanded, double-click the library or directory.
PowerBuilder hides all objects stored in the selected library or the files and subdirectories in the selected directory.

Using the popup menu

Like the other painters, the Library painter has a popup menu that provides items that apply to the selected object in the workspace.

- ❖ **To use the popup menu:**
 - 1 Position the mouse pointer on an object listed in the workspace.
 - 2 Click the right mouse button.

On Macintosh

On the Macintosh, hold down the `COMMAND` key and press the mouse button.

The popup menu displays. Which items display on the menu depend on the type of object you have selected.

- 3 Select the item you want from the menu.

Limiting the display of library entries

You can change what is shown in expanded libraries. You can specify:

- ◆ Which objects are displayed
- ◆ What information is shown for the displayed objects

Settings are remembered

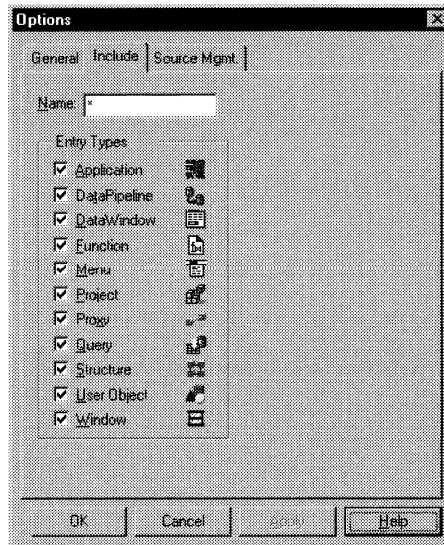
PowerBuilder records your preferences in the Library section of the PowerBuilder initialization file so that the next time you open the Library painter, the same objects and information are displayed.

Specifying which objects are shown

Initially, the Library painter displays all objects in expanded libraries. You can have the painter display only specific kinds of objects and/or objects whose names match a specific pattern. For example, you can limit the display to only DataWindow objects, or limit the display to windows that begin with w_emp.

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

- 1 Select 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: ? represents one character, * represents any string of characters. The default is all entries of the selected types.
 - ◆ To limit the display to specific entry types, clear the checkboxes for the entry types that you do not want to display. The default is all entries.
- 3 Click OK.

The Options property sheet closes.
- 4 Expand libraries to display the entries that meet the criteria.

Specifying which information is shown for the displayed objects

Initially, PowerBuilder displays the name, modification date, compilation date, size (of the compiled object), check-out status, and comments for displayed entries.

❖ **To specify which information is displayed:**

- ◆ Select Comments, Modification Date, Compilation Date, Check Out Status, or Size from the View menu to toggle the display of comments, dates, check-out status, and sizes.

Selecting library entries

You can select one or more library entries to act on. You can select entries in different libraries at the same time.

❖ **To select multiple entries:**

- 1 To select *noncontiguous entries*, press CTRL and click each entry you want to select.
- 2 To select *contiguous entries*, click the first entry you want to select, press SHIFT, then click the last entry you want to select.
- 3 To select *all entries* in a library, select the library, then click the Select All button.

PowerBuilder highlights selected entries.

Using comments

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

You can associate comments with an object when you first save it in a painter. You can use the Library painter to add or modify comments for a saved object.

Modifying comments for saved objects

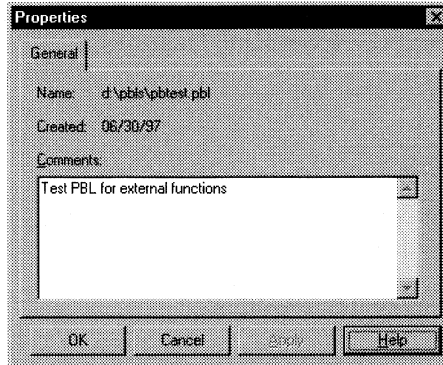
The Library painter is the only place where you can modify comments for a saved object.

Updating library comments

You can update comments for libraries or individual library entries with the following procedure.

❖ **To update comments for an existing PowerBuilder library:**

- 1 Select Properties from the Library's popup menu and select the General tab.



- 2 Add or modify the comments.
- 3 Click OK.

You return to the Library painter workspace.

Creating and deleting libraries

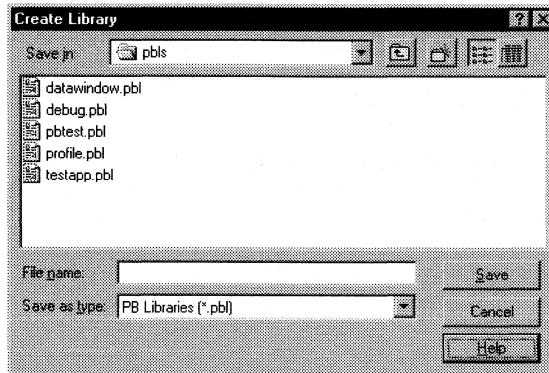
❖ To create a library:

- 1 Click the Create button.

or

Select Library>Create from the menu bar.

The Create Library dialog box displays showing the current directory and listing the libraries it contains. This dialog box may look different on your platform.



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

Since you are naming a file, you must follow the operating system rules on your platform. Make sure the file extension is PBL.

- 3 Click Save or OK.

The library property sheet displays.

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

It is good practice to add comments to describe the purpose of a library.

- 5 Click OK.

PowerBuilder creates the library.

❖ **To delete a library:**

- 1 Select the library you want to delete.
- 2 Select Library>Delete from the menu bar.

Restriction

You cannot delete a library that is in the current application'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. You cannot get them back.

Creating and deleting libraries during execution

You can use the LibraryCreate and LibraryDelete functions in scripts to create and delete libraries.

FOR INFO For information about these functions, see the *PowerScript Reference*.

Copying, moving, and deleting entries

As the needs of your application change, you will want to be able to rearrange libraries. Perhaps you are dividing your application into different libraries. To do that, you will want to copy and move entries between libraries or delete entries that you no longer need.

❖ To copy or move entries to a different library:

- 1 Select the entries you want to copy or move to another library.
- 2 Click the Copy button or the Move button.

or

Select Entry>Copy or Entry>Move from the menu bar.

The Copy Library Entries dialog box or the Move Library Entries dialog box displays.

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

If copying, PowerBuilder copies the entries. If moving, PowerBuilder moves the entries and deletes the entries from the source library. If a library entry with the same name already exists, PowerBuilder replaces it with the copied or moved entry.

❖ To delete entries:

- 1 Select the entries you want to delete.
- 2 Click the Delete button.

or

Select Entry>Delete from the menu bar.

You are asked to confirm the first deletion.

Being asked for confirmation

By default, PowerBuilder asks you to confirm each deletion. If you don't want to have to confirm deletions, select Design>Options to open the Options property sheet for the Library painter and clear the Confirm on Delete checkbox.

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

- 3 Click Yes to delete the entry. Click No to skip the current entry and go on to the next selected entry.

Searching library entries

You can search library entries to locate where a specified text string is used in your application. For example, you can search for:

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

❖ **To search library entries for a text string:**

- 1 Select the entries you want to search.

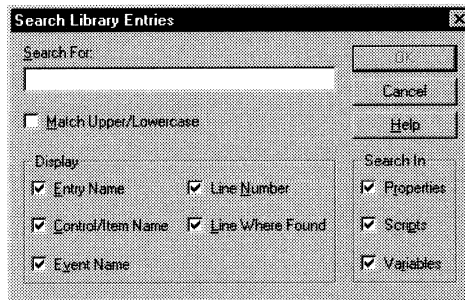
You can select entries across libraries by expanding the libraries and using SHIFT-click and/or CTRL-click to select multiple entries.

- 2 Click the Search button.

or

Select Entry>Search from the menu bar.

The Search Library Entries dialog box displays.



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

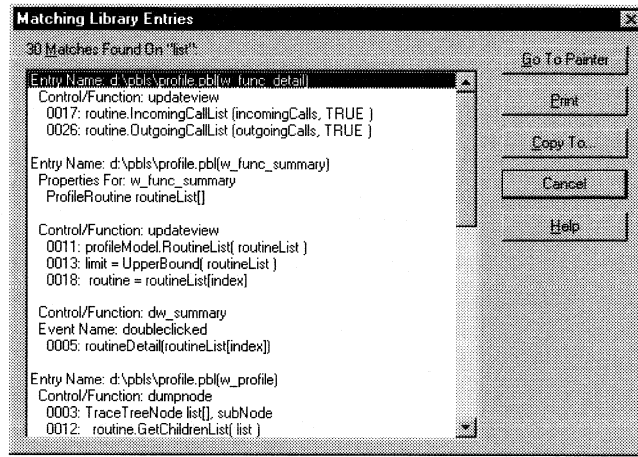
The string can be all or part of a word or phrase used in a property, script, or variable. You cannot use wildcards in the search string.

- 4 In the Display group box, select the information you want to display in the results of the search.
- 5 In the Search In group box, select the parts of the object that you want PowerBuilder to inspect: properties, scripts, and/or variables.

6 Click OK.

PowerBuilder searches the libraries for matching entries. When the search is complete, PowerBuilder displays the matching entries in the Matching Library Entries dialog box.

For example, the following dialog box displays the results of a search for the string list.



What you can do

From the Matching Library Entries dialog box, you can:

- ◆ Jump to the painter in which an entry was created (described next)
- ◆ Print the contents of the window
- ◆ Copy the search results to a text file

Jumping to a painter

You can jump from the Library painter directly to the painter where a specific entry was created.

❖ **To jump to a painter:**

- ◆ From the Match Library Entries dialog box that resulted from your browsing entries, double-click the entry.

or

Select the entry, then click the Go To Painter button.

FOR INFO For more on searching entries, see "Searching library entries" on page 78.

PowerBuilder opens the object in its painter.

Jumping to a painter from the workspace

You can also jump to a painter by double-clicking an entry in the Library painter workspace.

You can view the object and make changes as needed in the painter. When you close the painter, PowerBuilder returns you to the point in the Library painter from which you initiated the jump.

Library must be in current list

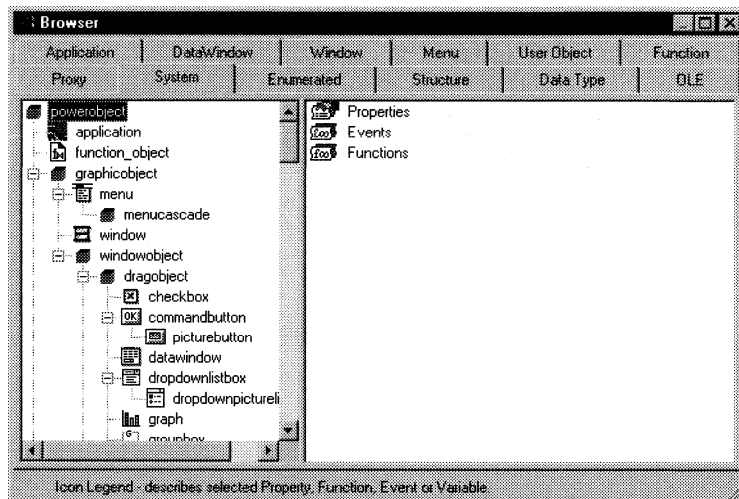
You can open and search any object that displays in the Library painter workspace, but you can only modify objects in libraries that are in your application's current library list.

Browsing the class hierarchy

You have probably used inheritance to define hierarchies of windows, menus, and user objects. You can examine these hierarchies using the Browser. In object-oriented terms, these are called **class hierarchies**: each PowerBuilder object defines a class.

❖ **To browse the class hierarchies:**

- 1 Click the Browse button in the PowerBar.
The Browser displays.
- 2 Choose the System tab to show the built-in PowerBuilder objects.
- 3 Display the popup menu in the object panel of the Browser and choose Show Hierarchy.
- 4 Select powerobject and choose Expand All from the popup menu.



PowerBuilder displays the hierarchy for built-in PowerBuilder objects.

Getting context-sensitive Help in the Browser

To get context-sensitive Help for an object, control, or function, select Help from its popup menu.

❖ **To display the class hierarchy for other object types:**

- 1 Choose a tab for another object type (for example, the Menu tab).
- 2 Select an object in the object panel and choose Show Hierarchy from its popup menu.

If there is no inheritance for the object type, Show Hierarchy is grayed out.

- 3 Select an object and choose Expand All from its popup menu.

PowerBuilder shows the selected object type in the current application. Descendent objects are shown indented under their ancestors.

Regenerating objects

The Browser provides a convenient way to regenerate objects and their descendants.

FOR INFO For more information, see "Regenerating library entries" on page 93.

Another way to browse hierarchies

You can also access a class browser for a specific object type when you are defining an inherited window, menu, or user object.

❖ **To display the class hierarchy for a specific object type:**

- 1 Open the Window painter, Menu painter, or User Object painter.
- 2 Click the Inherit button in the Select dialog box.
- 3 Click the Browse button in the Inherit From dialog box.

A class browser opens showing the inheritance hierarchy of objects of the specific object type.

Using check-out and check-in

If you are working with other developers on a large application, you will want to prevent multiple developers from modifying a library entry at the same time. To control access to library entries, PowerBuilder provides check-out and check-in facilities. Using them, you can check an object out of a public library and store a working copy in a private library. While the object is checked out, no one can modify it in the public library. When you finish updating an object, you can check the object back in to the public library.

FOR INFO For more about setting up libraries to support this type of development environment, see "Overview of libraries" on page 66 and *Application Techniques*.

About PowerBuilder and version control systems

On Windows and Macintosh systems, PowerBuilder provides interfaces to external version control systems so that you can manage your PowerBuilder objects through an external system while developing PowerBuilder applications. Most of the items on the Source menu in the Library painter apply only if you are using a version control system.

FOR INFO For more about using PowerBuilder with external version control systems, see *Version Control Interfaces*.

The check-out and check-in facilities described in this section are available even if you are not using a version control system to manage PowerBuilder objects.

How check-out works

When you want to work on an object and prevent others from making changes to the object, you check it out.

When you check out a library entry, PowerBuilder:

- ◆ Makes a working copy of the entry in a specified library (such as a private test library)
- ◆ Sets the status of the original entry to checked out

As long as the status of a library entry is checked out, changes can be made only to the working copy. If you or another user tries to open the original, PowerBuilder displays a message box warning that the entry is checked out and can be opened but not saved.

How check-in works

When you finish working with an entry that you checked out, you check the entry back in. PowerBuilder:

- ◆ Replaces the entry in the library from which you checked it out with the working copy
- ◆ Deletes the working copy from the library to which you checked it out

If you don't want to use the checked-out version

Instead of checking an entry back in, you can choose not to use the checked-out version by clearing the check-out status of the entry in the original library and deleting the working copy.

Connecting to a version control system

❖ **To connect to a version control system:**

- 1 Select Source>Connect from the menu bar.

The Connect dialog box displays.

- 2 Select a version control vendor from the Vendors dropdown list.

Supported source control vendors are:

- ◆ ObjectCycle from Powersoft
- ◆ CCC from Softool
- ◆ ENDEVOR from Computer Associates
- ◆ PVCS Version Manager from INTERSOLV
- ◆ Source Integrity from MKS
- ◆ Apple Source Server

In addition to these vendors, (PB Native) and SCC API display in the dropdown listbox.

(PBNative) is the identifier for PowerBuilder's native check-out and check-in facilities in the Library painter. If you select (PB Native) and click OK, the Set Current User ID dialog box displays. This allows you to enter or change a user ID to identify who checked out a PowerBuilder object.

SCC API is PowerBuilder's Source Code Control common API. It provides a standard interface to any version control system that implements features defined in the Microsoft Common Source Code Control Interface Specification.

FOR INFO For more about using PowerBuilder with external version control systems, see *Version Control Interfaces*.

Checking out entries

Checking out the Application object

Always check out the Application object by itself; PowerBuilder performs unique tasks when you check out the Application object.

FOR INFO See "Working with the Application object" on page 90.

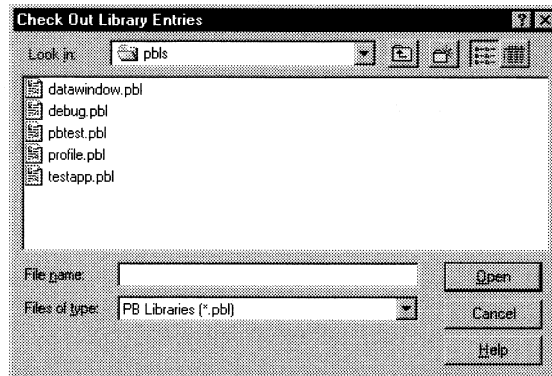
❖ To check out entries:

- 1 Select the entries you want to check out.
- 2 Click the Check Out button.
or
Select Source>Check Out from the menu bar.

- 3 If this is the first time you have checked out an entry, the User ID dialog box displays. Enter your user ID and click OK.

PowerBuilder saves your ID as the variable UserID in the Library section of the PowerBuilder initialization file and will not prompt you for an ID again unless you select Source>Connect from the menu bar. PowerBuilder uses the ID to identify who checked out objects.

The Check Out Library Entries dialog box displays the current directory and lists the libraries in that directory. The appearance of this dialog box depends on your platform.



- 4 Enter the name of the library in which you want to save the working copy in the File Name box or select the library from the list of libraries.

You must specify a library in the current application's library search path or you won't be able to save the working copy later.

- 5 Click Open.

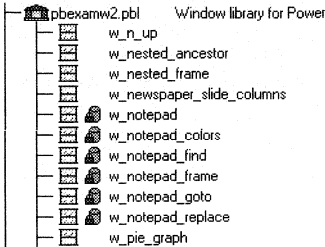
If you selected an entry that another user has checked out, PowerBuilder displays a message box and asks if you want to continue. Click Yes to process any remaining entries.

PowerBuilder makes a working copy of each selected entry and stores it in the destination library you specified.

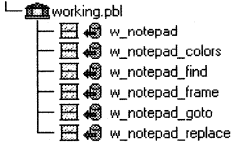
Checked-out items shown in workspace

The Library painter workspace marks with icons objects that have been checked out and objects that are working copies.

For example, the following shows that six windows have been checked out of PBEXAMW2.PBL:



The working copies are in WORKING.PBL:



Using the popup menu

You can also check out a Library entry by choosing Check Out from the entry's popup menu.

Viewing the checked-out entries

You can display a list of the entries in the current application that are checked out.

❖ **To display the checked-out entries:**

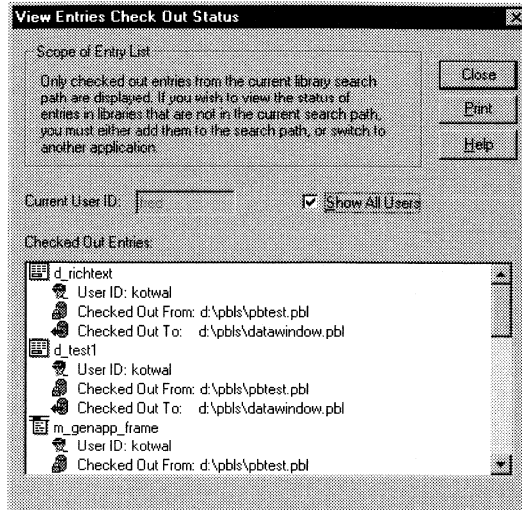
- 1 Click the Check Status button in the PainterBar.

or

Select Source>View Check Out Status from the menu bar.

The View Entries Check Out Status dialog box displays showing the name of each entry in the current application that has been checked out and the name of the library it is checked out to.

- 2 Select the Show All Users checkbox to display all the entries that are checked out from the current application.



- 3 (Optional) Click Print to print the list of checked-out entries.
- 4 Click Close.

The View Entries Check Out Status dialog box closes and the Library painter workspace displays.

Checking entries back in

When you finish working with an entry that you checked out, you check the entry back in.

Checking in the Application object

Always check in the Application object by itself; PowerBuilder performs unique tasks when you check in the Application object.

FOR INFO See "Working with the Application object" on page 90.

❖ **To check in entries:**

- 1 Select the entries (the working copies) you want to check in.
- 2 Click the Check In button in the PainterBar.

or

Select Source>Check In from the menu bar.

PowerBuilder replaces the entries in the original library with the working copies, deletes the working copies, then displays the Library painter workspace.

If you selected an entry that another user has checked out, PowerBuilder displays a message box and asks if you want to continue. Click Yes to process any remaining entries.

Using the popup menu

You can also check in a Library entry by choosing Check In from the entry's popup menu.

Clearing the check-out status of entries

Sometimes you will want to remove (clear) the check-out status of an entry without checking the entry back in. Perhaps you have decided not to update the object.

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

- 1 Select the entries whose check-out status you want to clear.
- 2 Select Source>Clear Check Out Status from the menu bar.
A message box asks whether you want to clear the check-out status of the entry.
- 3 Click Yes to clear the status.
You are asked whether you want to delete the working copy.
- 4 Click Yes to delete the working copy. Click No to retain it.

Working with the Application object

PowerBuilder manages various background tasks when you check out or check in the Application object.

Checking out an Application object

When you check out the Application object, make the checked-out application your current application to take advantage of PowerBuilder managing the background tasks involved. If you keep the public library application as the current application, you must perform those tasks yourself.

Check out only the Application object

Whenever you check out the current Application object, check it out by itself.

❖ **To check out the Application object:**

- 1 In the Library painter, select the current Application object.
- 2 Select Source>Check Out from the menu bar.
or
Click the Check Out button on the PainterBar.
- 3 Select the work library in the Check Out Library Entries dialog box and click Open.
- 4 (Optional) Enter comments in the Check Out Comment dialog box.
- 5 Click Yes to make the checked-out application your current application.

FOR INFO For how to modify the Application object, see "Modifying an Application object" next.

What happens

When you check out the Application object and make that the current application, PowerBuilder:

- ◆ Modifies the library path property of the Application object in your work library to include all of the libraries in the library path property of the Application object in the public library
- ◆ Associates with the Application object in your work library (now the current application) the configuration file generated for the Application object in the public library
- ◆ Maintains the connection to your version control system

Modifying an Application object

Once the current Application object is in your work library, you can modify it as you would any other PowerBuilder object.

❖ **To modify an Application object:**

- ◆ In the Library painter, double-click the Application object in your work library to open it.

Checking in an Application object

When you check in the Application object to the public library, PowerBuilder performs tasks similar to those it handled when you checked out the Application object.

Check in only the Application object

Whenever you check in the current Application object, check it in by itself.

❖ **To check in an Application object:**

- 1 Close the Application painter.
- 2 In the Library painter, select the Application object in your work library.
- 3 Select Source>Check In from the menu bar.
or
Click the Check In button on the PainterBar.
- 4 Specify your comments and preferences in the Check In Library Entries dialog box and click OK.
- 5 Click OK to continue.

What happens

When you check in the Application object to the public library, PowerBuilder:

- ◆ Makes the application in the public library the current application
- ◆ Maintains the connection with your version control system

Optimizing libraries

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

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

Once a week

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

❖ To optimize a library:

- 1 Choose the library you want to optimize.
- 2 Select Library>Optimize from the menu bar.

or

Select Optimize from the library's popup menu.

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

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 checkbox in the Library painter's Options tab dialog. If you clear this option, the new setting will remain in effect until you change it.

Regenerating library entries

Occasionally you may need to update library entries. 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 an application, you can *rebuild* entire libraries so objects are regenerated sequentially based on interdependence.
- ◆ When you upgrade to a new version of PowerBuilder, you need to *migrate* your applications.

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

❖ To regenerate library entries:

1 Select the entries you want to regenerate.

2 Click the Regen button.

or

Select Entry>Regenerate from the menu bar.

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

Regenerating
descendants

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

❖ To regenerate descendants:

1 Click the Browser button in the PowerPanel.

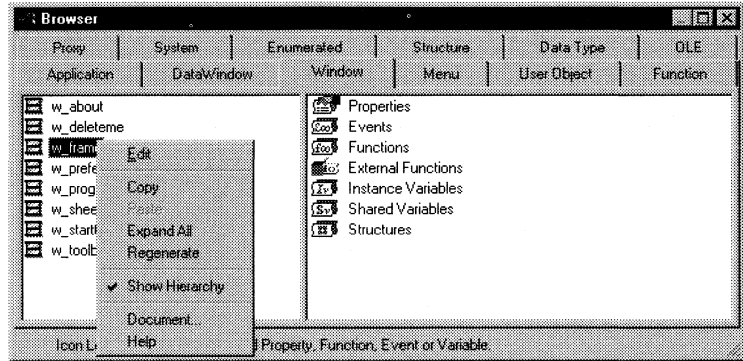
The Browser displays.

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

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

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

The Regenerate button displays on the popup menu.



- 4 Click the Regenerate button.

PowerBuilder regenerates all descendants of the selected ancestor.

FOR INFO For more about the Browser, see "Browsing the class hierarchy" on page 81.

Regenerate limitations

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

Rebuilding libraries

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

There are two methods to use when you rebuild an application:

- ◆ **Incremental rebuild** Updates all the objects and libraries referenced by any objects that have been changed since the last time you built the application
- ◆ **Full rebuild** Updates all the objects and libraries in your application

❖ To rebuild an application:

- 1 Select the libraries you want to rebuild.
- 2 Depending on your needs, choose either Design>Incremental Rebuild or Design>Full Rebuild from the menu bar.

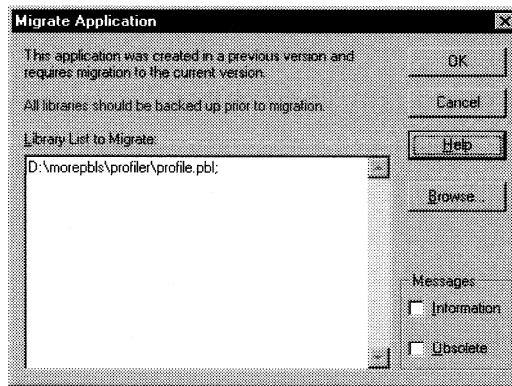
Migrating libraries

When you upgrade to a new version of PowerBuilder, your existing applications need to be migrated to the new version.

❖ To migrate an application:

- 1 Double-click the application you want to migrate.

The Migrate Application dialog box displays.



- 2 Select OK.

PowerBuilder migrates all objects and libraries in the application's path to the current version.

Exporting and importing entries

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

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

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

Caution

The primary use of the Export feature is to export source code, not to modify the source. Modifying source in an ASCII text file is not recommended.

Later on you can import the files back into PowerBuilder for storage in a library.

❖ To export entries to text files:

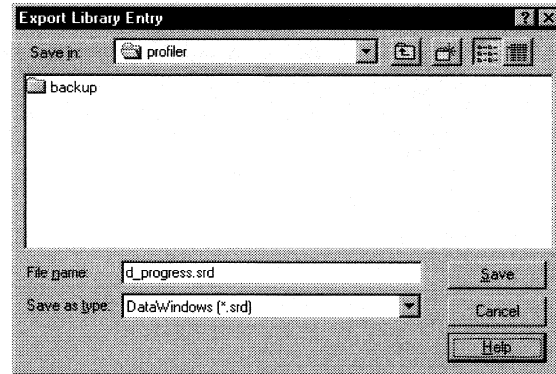
- 1 Select the Library entries you want to export.

- 2 Click the Export button.

or

Select Entry>Export from the menu bar.

The Export Library Entry dialog box displays 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 directory containing the last library you used.



PowerBuilder appends the file extension SR x , where x represents the object type.

- 3 Specify the filename and directory for the export file. Do not change the file extension from the one that PowerBuilder appended.
- 4 Click OK.

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

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

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

You can't see export files in the Library painter

Since export files are ASCII text files, the Library painter does not show them; it shows only libraries and directories.

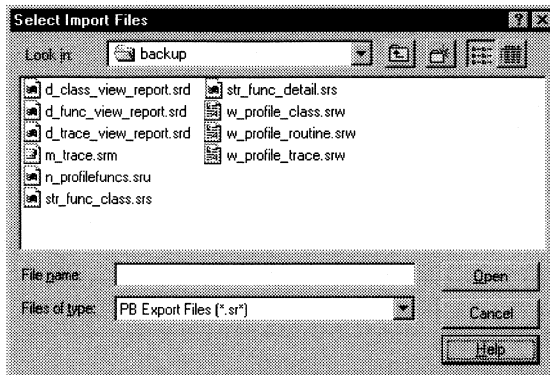
❖ **To import text files to library entries:**

- 1 Click the Import button.

or

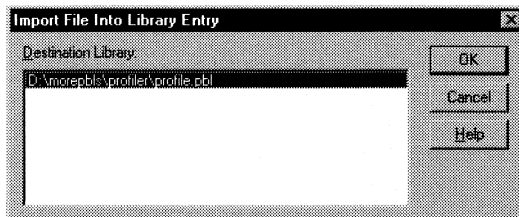
Select Entry>Import from the menu bar.

The Select Import Files dialog box displays showing the current directory and a list of files with the extension SR* in that directory. The current directory is the directory containing the PowerBuilder library you used last.



- 2 Select the files you want to import. Use SHIFT+click or CTRL+click to select multiple files.
- 3 Click Open.

The Import File Into Library Entry dialog box displays listing the libraries in the application's library search path.



- 4 Select the library you want to import the text files to.

5 Click OK.

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

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

Caution

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

Creating runtime libraries

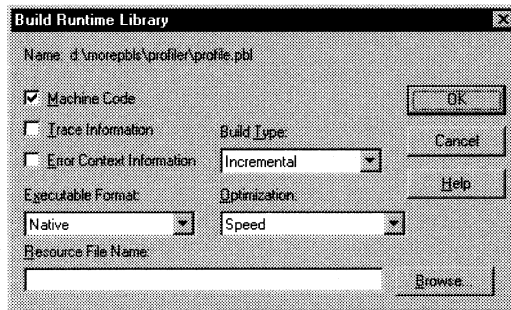
If you want your distributed application to use dynamic runtime libraries, you can create them in the Library painter.

FOR INFO For information about using runtime libraries, see Chapter 26, "Creating an Executable". That chapter also describes the Project painter, which you can use to automatically create dynamic runtime libraries.

❖ **To create a runtime library:**

- 1 Select the library you want to use to build a runtime library.
- 2 Select Library>Build Runtime Library from the menu bar.
or
Select Build Runtime Library from the library's popup menu.

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



- 3 If any of the objects in the source library use resources, specify a PowerBuilder resource file in the Resource File Name box (see "Including additional resources" next).
- 4 If you want to build a library for deployment on a 16-bit platform, select a format from the Executable Format drop down listbox.

Managing mixed environments

If you have both 16-bit and 32-bit versions of a project, you should be aware that PowerBuilder doesn't recognize when the executable format has been changed. It will overwrite the previous build. Be sure to specify a full build whenever you switch between 16-bit and 32-bit executable formats.

- 5 Select other options as appropriate.

FOR INFO For more information about build options, see "Project painter options" on page 878.

- 6 Click OK.

PowerBuilder closes the dialog box and creates a runtime library with the same name as the selected library. If you used the machine code compile option, the runtime library file extension is .dll on Windows and UNIX. On the Macintosh, the letters lib are appended directly to the library name with no period. If you did not select machine code, the file extension is .pbd on all platforms.

Including additional resources

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

FOR INFO For more on resource files, see "Using PowerBuilder resource files" on page 892.

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

Creating reports on library contents

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

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

The search results report contains the matching-entries information that PowerBuilder displays after it completes a search, described in "Searching library entries" on page 78. The other two types of reports are described in this section.

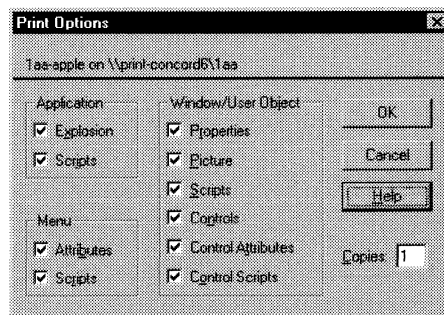
Creating library entry reports

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

❖ **To create library entry reports:**

- 1 Select the library entries you want information about.
- 2 Select Entry>Print from the menu bar.

The Print Options dialog box displays.



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

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

The settings are saved

PowerBuilder records these settings in the Library section of the PowerBuilder initialization file.

- 4 Click OK.

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

Creating the library directory report

The library directory report lists all entries in a selected library 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 you want the report for.
- 2 Select Library>Print Directory from the menu bar.

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

Coding Fundamentals

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

Writing Scripts

About this chapter

PowerBuilder applications are event driven. You specify the processing that takes place when an event occurs by writing a script. This chapter describes how to use the PowerScript painter to write scripts using the PowerScript language.

Contents

Topic	Page
Opening the PowerScript painter	108
Working in the PowerScript painter	110

For more information

For complete information about the PowerScript language, see the *PowerScript Reference*.

Opening the PowerScript painter

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



❖ **To open the PowerScript painter:**

- 1 Select the object or control you want to write a script for.

To write a script for a window (as opposed to a control in the window), make sure no control is selected.

- 2 Do one of the following:

- ◆ Display the popup menu for the object or control, then choose Script from the menu.
- ◆ Click the Script button in the PainterBar:

This button	Shows
	The selected object or control has no scripts
	The selected object or control has at least one script

What happens

The PowerScript painter opens in a separate window.

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

Item	What/Where it displays
Name of the current event	In the painter's title bar. If multiple events can occur in the object you are working with, the current event is the last event for which a script was displayed. If there is no script, the current event is an event that typically has a script
Name of the current object or control	In the painter's title bar
Select Event dropdown listbox	Lists events for the current object or control
Paste Argument dropdown listbox	Lists arguments for the current event

Item	What/Where it displays
Paste Object dropdown listbox	Lists objects and controls whose names you can paste into the script
Paste Global dropdown listbox	Lists global variables you can use in the script
Paste Instance dropdown listbox	Lists instance variables you can use in the script FOR INFO For more about instance variables, see the <i>PowerScript Reference</i>
Script for the current event	If there is no script, the workspace is blank

Changing the current event

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

Seeing which events have scripts

The Select Event dropdown listbox in the PowerScript painter indicates which events have scripts, as follows:

If there is a script	This happens
For the object or control you are working with	A script icon displays next to the event
In an ancestor object or control only	The script icon displays in color
In an ancestor as well as in the object or control you are working with	The script icon displays half in color

Working in the PowerScript painter

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

The PowerScript painter automatically:

- ◆ Color-codes scripts to identify data types, system-level functions, flow-of-control statements, comments, and literals
- ◆ Indents the script based on flow-of-control statements

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

Modifying Painter properties

❖ To specify painter properties:

- 1 Select Design>Options to display the property sheet.
- 2 Choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
The default font for static text as it appears in windows, user objects, and DataWindow objects	General
Font family, size, and color for the PowerScript painter	Font
Text and background coloring for PowerScript syntax elements	Coloring
Dropdown lists you want to include on the PowerScript painter	Dropdowns

These properties are discussed below.

Modifying painter properties

Some properties you specify in the PowerScript painter also affect the Function painter, File editor, Data Manipulation painter, Select painter, and Debug window.

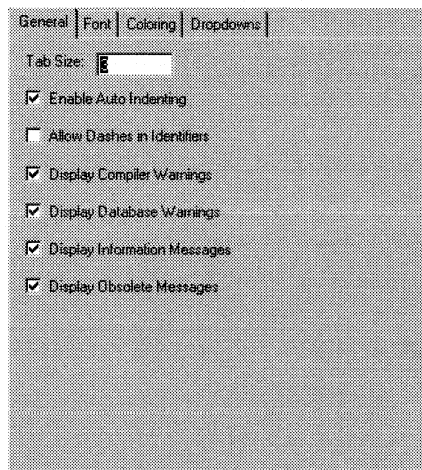
Specifying general script properties

The General page of the PowerScript painter property sheet allows you to specify:

- ◆ Tab size
- ◆ Whether you want to enable automatic indenting of scripts
- ◆ Whether you want to allow dashes in identifiers

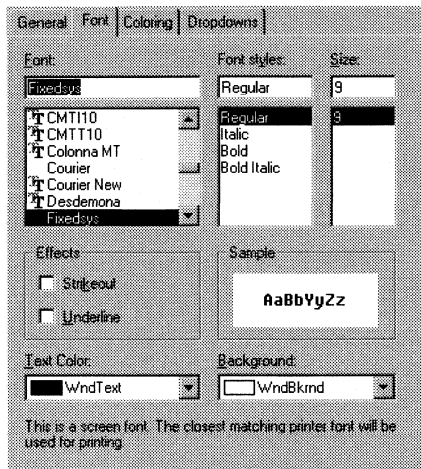
In addition, you can specify whether you want the compiler to display:

- ◆ Compiler warnings
- ◆ Database warnings
- ◆ Informational messages
- ◆ Obsolete messages



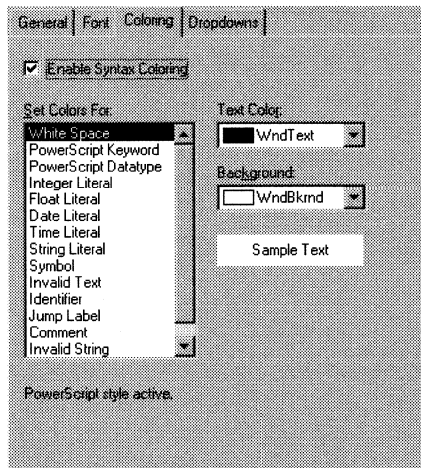
Specifying a font for your script

The PowerScript painter Font property page allows you to specify the font, size, style, color, and special effects you want to use when you write scripts.



Customizing colors in your script

Use the Coloring page of the PowerScript painter property sheet to enable syntax coloring in your scripts.

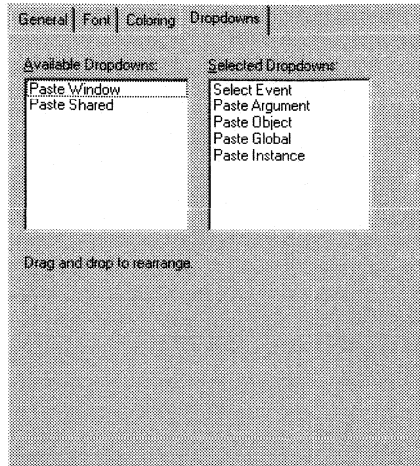


When you enable syntax coloring, the PowerScript editor color-codes scripts to identify data types, system-level functions, flow-of-control statements, comments, and literals.

You can customize colors by setting the text and background color of the script elements listed in the listbox.

Customizing the PowerScript painter

Use the PowerScript painter property sheet Dropdowns page to specify the paste boxes you want to include in the PowerScript painter.



Using the PainterBar and menu bar

You can perform standard editing tasks in the PowerScript painter. Like the other painters, the PowerScript painter has a PainterBar that provides a shortcut for performing frequently used activities. There is also a corresponding menu item (and often a shortcut key) for each activity:

Menu item	Shortcut key	Activity
Edit>Undo	CTRL+Z	Undoes the most recent edit
Edit>Cut	CTRL+X	Cuts selected text to the clipboard
Edit>Copy	CTRL+C	Copies selected text to the clipboard
Edit>Paste	CTRL+V	Pastes the contents of the clipboard at the current cursor location; replaces any selected text
Edit>Paste Function	—	Pastes a function statement at the current cursor location
Edit>Paste SQL	—	Pastes a SQL statement at the current cursor location

Menu item	Shortcut key	Activity
Edit>Paste Statement	—	Pastes a PowerScript statement at the current cursor location
Edit>Clear	DELETE	Deletes selected text; does not place the text in the clipboard
Edit>Select All	CTRL+A	Selects all text in the workspace
Edit>Comment Selection	—	Comments out the current line or all lines containing selected text by inserting two slashes before the first character in each line
Edit>Uncomment Selection	—	Uncomments the current line or all lines containing selected text by removing the two slashes before the first character in each line
Search>Find	CTRL+F	Specifies a string for which you want to search the script
Search>Find Next	CTRL+G	Finds the next occurrence of the specified search string
Search>Replace	CTRL+H	Replaces the specified search string
Search>Go to Line	CTRL+SHIFT+G	Goes to a specific line number
Design>Compile Script	CTRL+L	Compiles the script
Design>Select Object	—	Selects an object to paste at the cursor location
Design>Browse Object	—	Opens the Browser
File>Return	CTRL+SHIFT+T	Returns to the painter from which you opened the PowerScript painter

On Macintosh On Macintosh systems, use COMMAND instead of CTRL.

On UNIX On UNIX systems, you can also use the middle mouse button and dedicated editing keys on your keyboard for copying and pasting text if your window manager supports them.

Getting context-sensitive Help

In addition to accessing Help through the Help menu and F1 key, you can use context-sensitive Help in the PowerScript painter to display Help for reserved words and built-in functions.

❖ To use context-sensitive Help:

1 Place the insertion point within a reserved word (such as DO or CREATE) or built-in function (such as Open or Retrieve).

2 Press SHIFT+F1.

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

Copying Help text

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

Printing your script

You can print the current script on the default printer by selecting File>Print from the menu bar.

To change the printer or its settings, select File>Printer Setup from the menu bar before printing.

Pasting information

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

To paste	Use
Objects, controls, arguments, and global and instance variables	Paste listboxes above workspace
Properties, data types, functions, structures, variables, and objects	Browser
Contents of clipboard	Edit>Paste

To paste	Use
PowerScript statements	Paste Statement button <i>or</i> Edit>Paste Statement
SQL statements	Paste SQL button <i>or</i> Edit>Paste SQL
Built-in, user-defined, and external functions	Paste Function button <i>or</i> Edit>Paste Function
Contents of text files	File>Import

Undoing a paste

If you paste information into your script by mistake, immediately click the Undo button or select Edit>Undo from the menu bar.

These techniques are explained in the sections that follow.

Using the Paste listboxes

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

❖ To use the Paste listboxes using the mouse:

- 1 Move the cursor where you want to paste the object, control, or variable.
- 2 Click the Paste listbox for the type of information you want to paste. (See below for information about each of the listboxes.)

A list of available objects and controls or variables displays.

- 3 Click the entity you want to paste.

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

❖ **To use the Paste listboxes using the keyboard:**

- 1 Move the cursor where you want to paste the object, control, or variable.
- 2 Press CTRL+*number* to drop down a listbox (CTRL+1 drops down the Select Event listbox, CTRL+2 drops down the Paste Object listbox, and so on).
- 3 Use the arrow keys to select the entity.
- 4 Press ENTER.

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

The Paste Object listbox

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

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

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

The Paste Arguments listbox

The Paste Arguments listbox displays all arguments defined for the selected event. (To define additional arguments for a user-defined event, select Declare>User Events from the menu bar.)

The Paste Global listbox

The Paste Global listbox displays all global variables defined for the application. (To define additional global variables, select Declare>Global Variables from the menu bar.)

The Paste Instance listbox

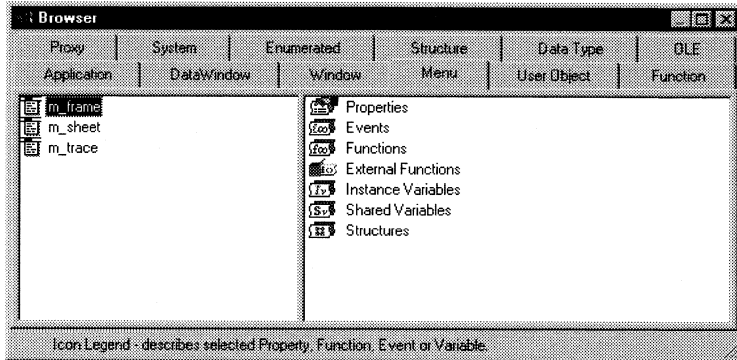
The Paste Instance listbox displays all instance variables defined for the corresponding Application object, window, menu, or user object, or one of its ancestors. (To define instance variables, select Declare>Instance Variables from the menu bar.)

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

Using the Browser

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

The Browser has two panes:



The left pane displays one type of object, such as 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 popup menu.

❖ To use the Browser to paste information into the PowerScript painter:

- 1 Move the cursor where you want to paste the information. Select any text you want replaced by the pasting.
- 2 Click the Browse Objects button located in the PainterBar.
or
Select Design>Browse Object.

The Browser opens showing the object that is currently open in the PowerScript painter. If you want to paste information about another object, select the appropriate tab and then select the object in the left pane.

- 3 Select the category of information you want to display by expanding the appropriate folder in the right pane.

- 4 Select the information and click Copy.

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

FOR INFO For information about using the Browser to paste OLE object information into a script, see *Application Techniques*.

Opening the Browser for pasting

There is a Browser button in the PowerBar and a Browse Objects button in the PainterBar. Both buttons display the Browser. However, the Browse Objects button in the PainterBar makes it easier for you to paste objects into your script because it displays information for the object you're working on.

Pasting statements

You can paste a template for the following PowerScript statements:

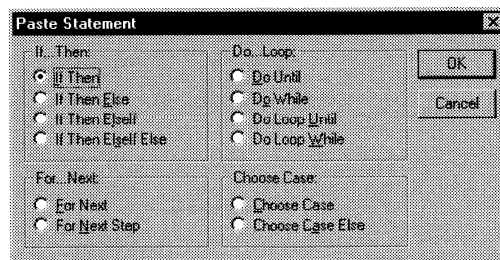
- ◆ IF...THEN
- ◆ DO...LOOP
- ◆ FOR...NEXT
- ◆ CHOOSE CASE

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

❖ To paste a PowerScript statement into the script:

- 1 Move the cursor where you want to paste the function.
- 2 Select the Paste Statement button from the PainterBar.
or
Select Edit>Paste Statement from the menu bar.

The Paste Statement dialog box displays.



- 3 Select the statement you want to paste into the script and click OK.
The statement prototype displays at the insertion point in the script.
- 4 Replace the prototype values with the conditions you want to test and the actions you want to take based on the test results.

FOR INFO For more about PowerScript statements, see the *PowerScript Reference*.

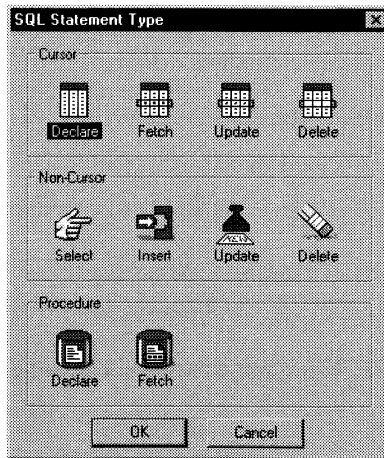
Pasting SQL

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

❖ To paste a SQL statement:

- 1 Place the insertion point where you want the SQL statement in the script.
- 2 Click the Paste SQL button in the PainterBar.
or
Press CTRL+Q.
or
Select Edit>Paste SQL from the menu bar.

The SQL Statement Type dialog box displays.



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

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

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

The statement displays at the insertion point in the workspace.

FOR INFO For more about embedding SQL in scripts, see the *PowerScript Reference*.

Pasting functions

You can paste any function into a script.

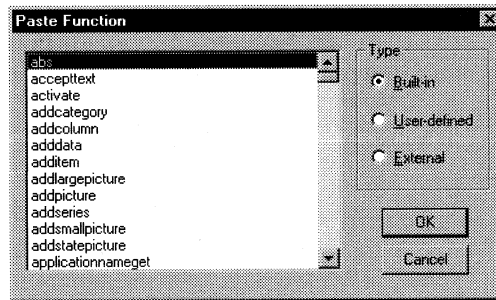
❖ To paste a function into a script:

- 1 Click the Paste Function button in the PainterBar.

or

Select Edit>Paste Function from the menu bar.

The Paste Function dialog box displays.



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

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

For more information

For more about pasting user-defined functions, see "Pasting user-defined functions" on page 146.

For more about external and built-in functions, see *Application Techniques*.

Pasting contents of files

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

Importing from a file

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

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

- 1 Place the insertion point where you want the file contents pasted.
- 2 Select File>Import from the menu bar.

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

- 3 Choose the file containing the code you want. You can change the type of files displayed by changing the file specification in the File Name box.

PowerBuilder copies the file into the PowerScript painter at the insertion point.

Exporting a script to a file

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

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

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

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

The File Export dialog box displays.

- 3 Name the file and click OK.

The default file extension is SCR.

Compiling the script

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

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

- ◆ Click the Compile button.

or

Select Design>Compile Script from the menu bar.

or

Press CTRL+L.

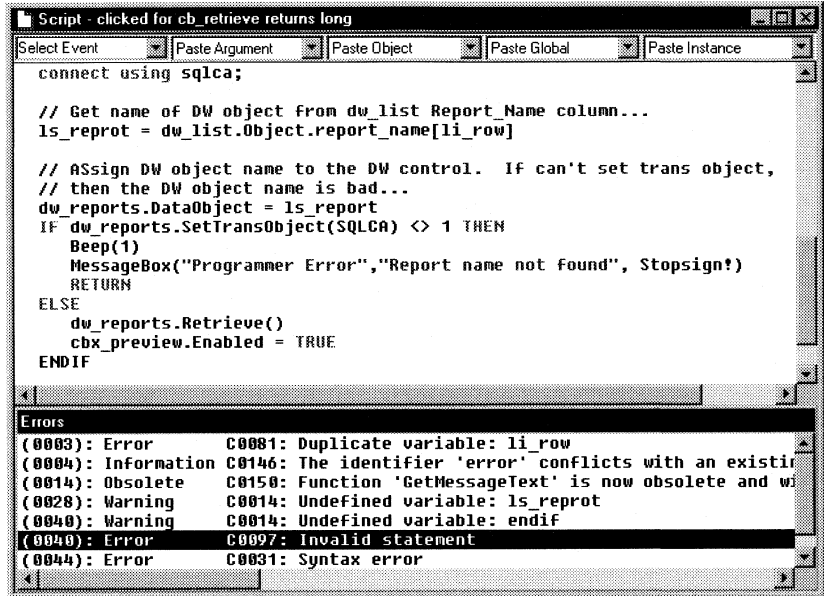
PowerBuilder compiles the script and reports any problems it finds, as described in "Handling problems" next.

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

- ◆ Click the Return button, as described in "Leaving the PowerScript painter" on page 127.

Handling problems

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



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 leave the PowerScript painter. Errors are shown in the Message window as:

line number: Error error number:message

Understanding warnings

Warnings indicate problems that you should be aware of but that do not prevent a script from compiling.

There are three kinds of warnings.

Compiler warnings Compiler warnings inform you of syntactic problems, such as undeclared variables. PowerBuilder lets you compile a script that contains compiler warnings and return to the painter from which you opened the PowerScript painter. But you must fix the problem in the script before you can save the object that the script is for, such as the window or menu. Compiler warnings are shown in the Message window as:

line number: Warning warning number:message

Obsolete warnings Obsolete warnings inform you when you use any obsolete functions or syntax in your script. Obsolete functions, although they still compile and run, have been replaced by more efficient functions and will be discontinued in the next release of PowerBuilder. You should replace all references to obsolete functions as soon as possible. Obsolete warnings are shown in the Message window as:

line number: Obsolete warning number:message

Database warnings Database warnings come from the database manager you are connected to. (PowerBuilder connects to the database manager when you compile a script containing embedded SQL.) Typically, these warnings arise because you are referencing a database you are not connected to (perhaps you are accessing two databases in the script; you can only be connected to one at a time in PowerBuilder). Database warnings are shown in the Message window as:

line number: Database warning number:message

PowerBuilder lets you compile scripts with database warnings and also lets you save the associated object. It does this because it can't know whether the problem will apply during execution (because the execution environment may be different from the compile-time environment).

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, you've used a global variable name as a local variable, which could result in a conflict later. You will also receive an information message if you use a function that has been replaced by a more efficient function and that will become obsolete in a future release.

Information messages are shown in the Message window as:

line number: Information number:message

Displaying warnings and messages

Use the Display Compiler Warnings, Display Obsolete Messages, Display Information Messages, and Display Database Warnings items on the PowerScript painter property sheets General tab to specify which messages display when you compile. The default is to display compiler and database warning messages. (Error messages always display.)

Fixing problems

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

To save a script with errors

Comment out the lines containing errors.

Leaving the PowerScript painter

❖ To leave the PowerScript painter:

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

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

When you click the Return button, PowerBuilder compiles the script. If there are problems, PowerBuilder reports them, as described in "Handling problems" on page 124.

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

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

Leaving without saving your work

❖ To leave the PowerScript painter without saving your changes:

- 1 Select File>Close from the menu bar.

You are asked whether you want to save your changes.

- 2 Click No to discard your changes.

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

Working with User-Defined Functions

About this chapter

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

Contents

Topic	Page
What are user-defined functions?	130
Defining user-defined functions	132
Modifying user-defined functions	143
Using your functions	146

What are user-defined functions?

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

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

Two kinds

There are two kinds of user-defined functions:

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

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

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

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

Deciding which kind you want

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

For example, say you want a function that returns the contents of a `SingleLineEdit` control in one window to another window. Make it a window-level function, defined in the window containing the `SingleLineEdit` 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, which performs processing unique to it.

There is never any ambiguity about which function is being called, because you always specify the object's name when you call an object-level function.

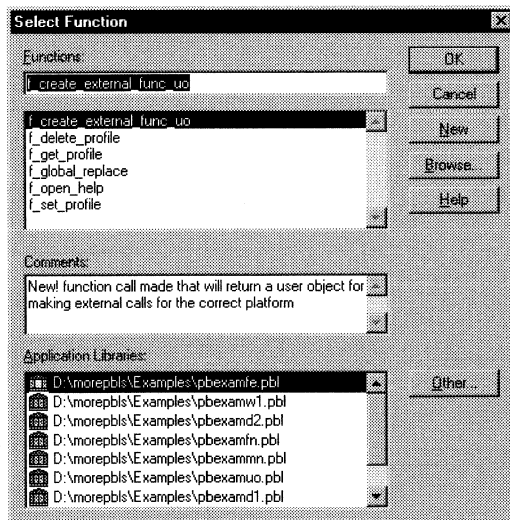
Defining user-defined functions

- ❖ **To define a user-defined function:**
 - 1 Open the Function painter.
 - 2 Name the function.
 - 3 Define a return type.
 - 4 For object-level functions, define an access level.
 - 5 Define arguments for the function.
 - 6 Code the function.

Opening the Function painter

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

- ❖ **To open the Function painter for a global function:**
 - 1 Click the Function painter button in the PowerBar or PowerPanel.The Select Function dialog box displays.

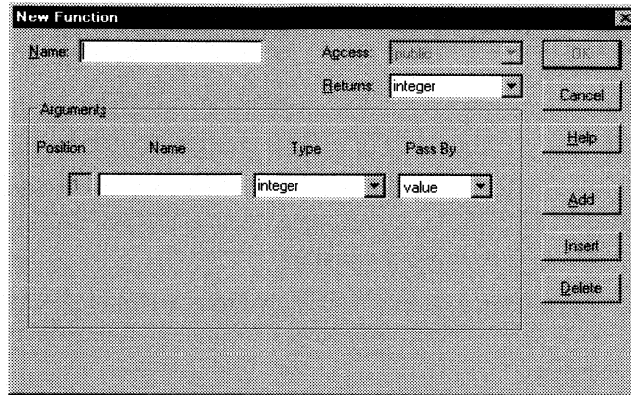


- 2 Select a function and click OK.

or

Click the New button to declare a new function.

The New Function dialog box displays.



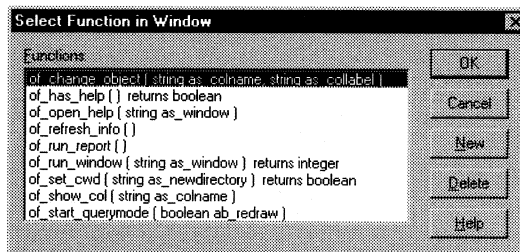
❖ **To open the Function painter for an object-level function:**

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

You can declare functions for windows, menus, user objects, or applications. For example, if you want to declare a function for the window `w_emp`, open the Window painter and select `w_emp`.

- 2 From the Declare menu, select the appropriate menu item: Window Functions, Menu Functions, User Object Functions, or Application Functions.

A Select Function dialog box displays.



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

- 3 Select a function and click OK.
or
Click the New button to declare a new function.

Naming the function

Function names can have up to 40 characters.

FOR INFO For valid characters, see the *PowerScript Reference*.

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

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

```
f_calc  
f_get_result
```

You could preface all object-level functions with `of_`. For example:

```
of_refreshwindow  
of_checkparent
```

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

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 data type of the returned value.

Later, you will code a RETURN statement in the function that specifies the value to return (see "Returning a value" on page 139). When you call the function in a script or another function, you can use an assignment statement to assign the returned value to a variable in the calling script or function. You can also use the returned value directly in an expression in place of a variable of the same type.

❖ **To define a function's return type:**

- ◆ Select the return type from the Returns dropdown listbox in the New Function dialog box.

or

Type in the name of an object type you have defined.

You can specify any PowerBuilder data type, including the standard data types, such as integer and string, as well as objects and controls, such as window or MultiLineEdit.

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

❖ **To specify that a function does not return a value:**

- ◆ Select (None) from the Returns list.

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

Examples of functions returning values

The following examples show the return type you would specify for some different functions.

If you are defining	Specify this return type
A mathematical function that does some processing and returns a real number	real
A function that takes a string as an argument and returns the string in reverse order	string
A function that is passed an instance of window <code>w_calculator</code> , does some processing (such as changing the window's color), then returns the modified window	<code>w_calculator</code>

Defining the access level

The New Function dialog box has a dropdown listbox named Access.



This box specifies the **access level** of the function—where you can call the function in the application.

For global functions

Global functions can always be called anywhere in the application. In PowerBuilder terms, they are **public**. So when you are defining a global function, you cannot modify Access; the box is read-only.

For object-level functions

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

Access	Means you can call the function
Public	In any script in the application
Private	Only in scripts for events in the object in which the function is defined. You cannot call the function from descendants of the object
Protected	Only in scripts for the object in which the function is defined and its descendants

If a function is only to be used internally within an object, you should define its access as private or protected. That way, you ensure that the function is never called inappropriately from outside the object. (In object-oriented terms, defining a function as protected or private **encapsulates** the function within the object.)

Defining arguments

Like built-in functions, user-defined functions can have any number of arguments, including none. You declare the arguments and their types when you define a function.

If the function takes no arguments

Leave blank the initial argument shown in the New Function dialog box and click OK to go to the Function painter workspace.

❖ **To define arguments:**

- 1 Name the argument.

The order in which you specify arguments here is the order you use when calling the function.

- 2 Declare the argument's type. You can specify any data type, including:
 - ◆ Built-in data types (such as integer and real)
 - ◆ Object types (such as window) or specific objects (such as w_emp)
 - ◆ User objects
 - ◆ Controls (such as CommandButtons)

Array arguments

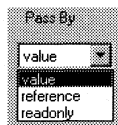
You can also declare that an argument is an array. When you pass an array, you must include the square brackets in the array definition (for example, price[] or price[50]), and the data type of the array must be the data type of the argument.

FOR INFO For information on arrays, see the *PowerScript Reference*.

- 3 Declare how you want the argument passed (see "Passing arguments" next).
- 4 If you want to add another argument, click the Add button and repeat steps 1 to 3.
- 5 Click OK.

Passing arguments

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



By reference

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

By value	When you pass by value, you are passing the function a temporary local copy of the argument. The function can alter the value of the local copy within the function, but the value of the argument is not changed in the calling script or function.
Readonly	When you pass readonly, the variable's value is available to the function but it is treated as a constant. Readonly provides a performance advantage over by value for strings, blobs, date, time, and DateTime, because it does not create a copy of the data.

Coding the function

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

Using the workspace

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

FOR INFO For information about using the workspace, see "Working in the PowerScript painter" on page 110.

Workspace for global functions	<p>When you declare a global function, the Function painter workspace has a title and two dropdown listboxes:</p> <ul style="list-style-type: none">◆ The title is the name of the function you are building or modifying◆ The Paste Argument dropdown listbox lists the arguments defined for the function◆ The Paste Global dropdown listbox lists the global variables defined in the application <p>You can paste arguments and variables from the lists into the statements in the function you are building.</p>
Workspace for object-level functions	<p>When you declare an object-level function, the workspace displays the name of the current object in the title and four listboxes.</p> <p>The first two listboxes are the same as in the workspace for global functions. The other two are Paste Instance and Paste Object. You can use these listboxes to paste names of instance variables, objects, and controls in the current object.</p>

What functions can contain

User-defined functions can include PowerScript statements, embedded SQL statements, and calls to built-in, user-defined, and external functions.

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

Returning a value

If you specified a return type for your function in the New Function dialog box, you must return a value in the body of the function.

Syntax

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

```
RETURN expression
```

where *expression* is the value you want returned by the function. The data type of the expression must be the data type you specified for the return value for the function.

Example

The following function returns the result of dividing arg1 by arg2 if arg2 does not equal zero; it returns -1 if arg2 equals zero:

```
IF arg2 <> 0 THEN
    RETURN arg1 / arg2
ELSE
    RETURN -1
END IF
```

Using text files

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

❖ To save a function in a text file:

- 1 If you want to save the entire function, make sure no text is selected. If you want to save only part of the function, select the part you want to save.
- 2 Select File>Export from the menu bar.
The File Export dialog box displays.
- 3 Name the file. By default, the file has the extension FUN (for function).

❖ **To read a text file into the workspace:**

- 1 Place the insertion point where you want the contents of the file to appear.
- 2 Select File>Import from the menu bar.
The File Import dialog box displays.
- 3 Choose the file and click OK.
PowerBuilder copies the contents of the file into the workspace at the insertion point.

Compiling and saving the function

When you finish building a function, compile it and save it in a library. Then you can use it in scripts or other user-defined functions in any application that includes the library containing the function in its library search path.

You can either compile the function without leaving the painter (as you can in the PowerScript painter) or compile the function in the process of saving it.

Compiling without leaving the painter

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

❖ **To compile without leaving the Function painter:**

- ◆ Click the Compile button.
or
Select Design>Compile Script from the menu bar.
or
Press CTRL+L.

On Macintosh

On the Macintosh, press the COMMAND key instead of the CTRL key.

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

Compiling and saving

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

❖ To save a global function:

- 1 Click the Return button.

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

- 2 You can rename the function, add comments about the function, or change the library you want to save the function in.
- 3 Click OK.

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

❖ To save an object-level function:

- 1 Click the last button in the PainterBar.

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

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

- 2 Save the object.

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

Correcting compiler errors

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

- ◆ Error messages
- ◆ Warning messages
- ◆ Information messages

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

FOR INFO For information, see "Handling problems" on page 124.

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

To save a function with unresolved errors

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

Modifying user-defined functions

You can change the definition of a user-defined function at any time. You change the processing performed by the function by simply modifying the statements in the workspace. You can also change the return type, argument list, or access level for a function.

❖ **To change a function's return type, arguments, or access level:**

- 1 Open the Function painter.

or

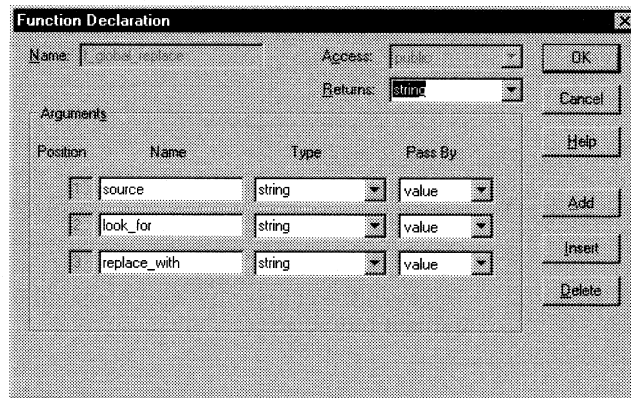
Open the painter for the object-level function you want to edit and select the function type from the Declare menu.

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

The Function painter workspace displays.

- 3 Click the Edit button in the PainterBar.

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



- 4 Make the changes you want, then click OK.

You return to the workspace.

- 5 Save the modified function.

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

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

Changing the arguments

You can change a function's arguments at any time:

- ◆ *Add* an argument by clicking the Add button. The next position number and boxes for defining the new argument display below the last argument in the list.
- ◆ *Insert* an argument by moving the pointer to the argument before which you want to insert the argument and clicking the Insert button.

The position number adjusts, and boxes for defining the new argument display above the selected argument.
- ◆ *Delete* an argument by selecting it and clicking the Delete button.

To change the position number of an argument

To change the position number of an argument, delete the argument and insert it as a new argument in the correct position.

Recompiling other scripts

Changing arguments and the return type of a function affect scripts and other functions that call the function. You should recompile any script in which the function is used. This guarantees that the scripts and functions work correctly during execution.

FOR INFO For an easy way to find all places a function is used, see "Seeing where a function is used" next.

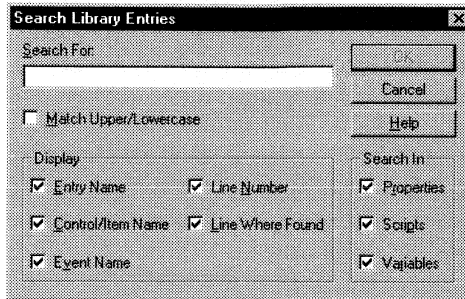
Seeing where a function is used

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

- ❖ **To determine which functions and scripts call a user-defined function:**
 - 1 Open the Library painter.
 - 2 Select all the entries you want to search for references to the user-defined function.

- 3 Click the Search button in the PainterBar.

The Search Library Entries dialog box displays.



- 4 Specify the user-defined function as the search text and specify the types of components you want to search.
- 5 Click OK.

PowerBuilder displays all specified components that reference the function. You can double-click a listed component to open the appropriate painter.

FOR INFO For more about browsing library entities, see "Searching library entries" on page 78.

Using your functions

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

FOR INFO For complete information about calling functions, see *Application Techniques*.

Pasting user-defined functions

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

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

- 1 In the PowerScript or Function painter, move the insertion point to where you want to paste the function.
- 2 Click the Browse Objects button.
or
Select Design>Browse Object from the menu bar.

The Browser displays. The selected tab page and object depend on which painter you were in when you opened the Browser.
- 3 Select a global function from the Function page.
or
Select the object that contains the object-level function you want to paste from the corresponding page (such as the Window page).
- 4 Double-click the Functions category in the right pane.

The Functions folder expands. On the Function page, it shows the signature of the selected function. On other pages, it shows all the functions available for the selected object.
- 5 Select the function you want to paste and select Paste from its popup menu.

The function and its prototype parameters display at the insertion point in your script.
- 6 Specify the required arguments.

Working with Structures

About this chapter

This chapter describes how to build and use structures.

Contents

Topic	Page
What are structures?	148
Defining structures	149
Modifying structures	152
Using structures	153

What are structures?

A **structure** is a collection of one or more related variables of the same or different data types grouped under a single name. In some languages, such as Pascal and COBOL, structures are called *records*.

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

Two kinds

There are two kinds of structures:

- ◆ **Global structures**, which are not associated with any object in your application. You can 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

❖ **To define a structure:**

- 1 Open the Structure painter.
- 2 Define the variables that compose (or make up) the structure.
- 3 Save the structure.

Opening the
Structure painter

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

❖ **To open the Structure painter for a global structure:**

- 1 Click the Structure painter button in the PowerBar or PowerPanel.
The Select Structure dialog box displays.
- 2 Select a structure and click OK.
or
Click the New button to declare a new structure.

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

- 1 Open the painter for the object you want to declare the structure for.
You can declare structures for windows, menus, user objects, or applications. For example, if you want to declare a structure for the window `w_emp`, open the Window painter and select `w_emp`.
- 2 From the Declare menu, select the appropriate menu item: Window Structures, Menu Structures, User Object Structures, or Application Structures.
A Select Structure dialog box displays.
Here you can select a structure for the current object or declare a new structure for the object.
- 3 Select a structure and click OK.
or
Click the New button to declare a new structure.
The New Structure dialog box displays.

Defining the variables

❖ To define the variables that compose the structure:

- 1 Enter the name of a variable that you want to include in the structure.
- 2 Enter the data type of the variable. The default for the first variable is string; the default for subsequent variables is the data type of the previous variable.

You can specify any PowerBuilder data type, including the standard data types, such as integer and string, as well as objects and controls, such as window or MultiLineEdit.

You can also specify any object types that you have defined. For example, if you are using a window named `w_calculator` that you have defined and want the structure to include the window, type **w_calculator** as the data type. (You cannot select `w_calculator` from the list, since the list only shows built-in data types.)

A structure as a variable

A variable in a structure can itself be a structure. Specify the structure's name as the variable's data type.

- 3 Enter the number of decimal places in the Dec box if the data type is decimal. The default is 2.
- 4 Repeat until you have entered all the variables.

Naming and saving the structure

Structure names can have up to 40 characters.

FOR INFO For information about valid characters, see the *PowerScript Reference*.

You might want to adopt a naming convention for structures so that you can recognize them easily. For example, you could preface all global structure names with `s_` and all object-level structure names with `os_`.

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

❖ To save a global structure:

- 1 Click OK in the New Structure dialog box.
The Save Structure dialog box displays.
- 2 Name the structure.
- 3 (Optional) Add comments to describe your structure.

- 4 Choose the library to save the structure in.
- 5 Click OK.

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

❖ **To save an object-level structure:**

- 1 Click OK in the New Structure dialog box.

The Save Structure in *ObjectType* dialog box displays.

- 2 Name the structure and click OK.

PowerBuilder saves the structure as part of the object in the library in which the object resides.

Comments and object-level structures

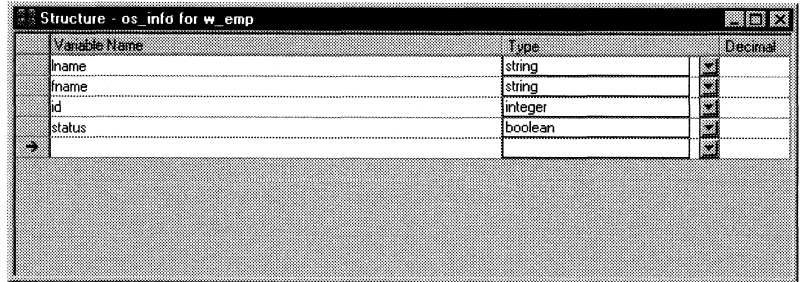
You cannot enter comments for an object-level structure, because it is not a PowerBuilder object.

Modifying structures

❖ **To modify a structure:**

- 1 Select the global structure you want to modify in the Select Structure dialog box.
or
Open the painter for the object that contains the structure and select the structure from the Select Structure dialog box.

The Structure dialog box displays.



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

To insert a variable between existing variables, highlight the variable that you want to follow the new variable and click the Insert button on the PainterBar.

To delete a variable, click the Delete button.
- 3 Save the modified structure by doing one of the following:
 - ◆ Choose File>Save to save the structure with its current name.
 - ◆ Choose File>Save As to save the structure with a new name.

Building a similar structure

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

Using structures

After you define the structure, you can:

- ◆ Reference 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 in the Structure painter, you are defining a new data type. You can use this new data type in scripts and user-defined functions as long as the structure definition is stored in a library in the application's library search path.

❖ To use a structure in a script or user-defined function:

- 1 Declare a variable of the structure type.
- 2 Reference the variable in the structure.

Referencing global structures

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

structure.variable

Example

Assume that `emp_data` is a global structure with the variables `emp_id`, `emp_dept`, `emp_fname`, `emp_lname`, and `emp_salary`.

To use this structure definition, declare a variable of type `emp_data` and use dot notation to reference the structure's variables, as shown in the following script:

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

emp1.emp_id = 100          // Assign values to the
emp1.emp_dept = 200       // structure variables.
emp1.emp_fname = "John"
emp1.emp_lname = "Paul-Jones"
emp1.emp_salary = 19908.23

// Retrieve the value of a structure variable.
```

```
emp2.emp_salary = emp1.emp_salary * 1.05

// Use a structure variable in a
// PowerShell function.
MessageBox ("New Salary", &
    String(emp2.emp_salary, "$#, ##0.00"))
```

Referencing object-level structures

You reference object-level structures in scripts for the object itself exactly as you do global structures: declare a variable of the structure type, then use dot notation as:

structure.variable

Example

Assume that the structure `cust_data` is defined for the window `w_history` and you are writing a script for a `CommandButton` in the window. To use the structure definition in the script, you write:

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

Allowing access to object-level structures outside the object

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

❖ To reference object-level structures outside the object:

- 1 In the object that defines the structure, declare an instance variable of the structure type.
- 2 In any script or user-defined function in the application, reference a variable in the structure using the following syntax:

object.instance_variable.variable

Example

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

```
cust_data    winstruct
```

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

```
w_history.winstruct.name
```

Copying structures

- ❖ **To copy the values of a structure to another structure of the same type:**
 - ◆ Assign the structure to be copied to the other structure using this syntax:

```
struct1 = struct2
```

PowerBuilder copies all the variable values from struct2 to struct1.

Example

These statements copy the values in emp2 to emp1:

```
emp_dataemp1, emp2
...
emp1 = emp2
```

Using structures with functions

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

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

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

Example

Assume the following:

- ◆ `Revise` is an external function that expects a structure as its argument.
- ◆ `emp_data` is a declared variable of a structure data type.

You can call the function as follows:

```
Revise(emp_data)
```

Declare the function first

The external function must be declared before you can reference it in a script.

FOR INFO For more about passing arguments to external functions, see *Application Techniques*.

Displaying and pasting structure information

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

❖ **To display information about a global structure in the Browser:**

- 1 Select the Structure tab and select a structure.
- 2 Double-click the properties folder in the right pane.

The properties folder expands to show the structure variables as properties of the structure.

❖ **To display information about an object-level structure in the Browser:**

- 1 Select the tab for the type of object for which the structure is defined.
- 2 Select the object that contains the structure.
- 3 Double-click the structure folder in the right pane.

The structure folder expands to display the structure variables using dot notation.

❖ **To paste the information into a script:**

- 1 Scroll to the structure variable you want to paste.
- 2 Select Paste from the variable's popup menu.

The variable name displays at the insertion point in your script.

Working with Windows

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

Defining Windows

About this chapter

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

Contents

Topic	Page
Overview of windows	160
Types of windows	162
Building a new window	169
Viewing your work	183
Writing scripts in windows	185
Running a window	189
Using inheritance to build a window	190

Overview of windows

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

A window consists of:

- ◆ **Properties**, which define the window's appearance and behavior (for example, a window might have a title bar or a Minimize box)
- ◆ **Events** (windows have events like other PowerBuilder objects)
- ◆ **Controls** placed in the window

At the window level

When you create a window, you specify its properties in the Window painter (you can also dynamically change window properties in scripts during execution). You also write scripts for window events that specify what happens when a window is manipulated. For example, you can connect to a database when a window is opened by coding the appropriate statements in the script for the window's Open event.

At the control level

You place PowerBuilder controls (such as CheckBoxes, CommandButtons, or MultiLineEdits) in the window to request and receive information from the user and to present information to the user.

After you place a control in the window, you can define the style of the control, move and resize it, and build scripts to determine how the control responds to events.

Designing windows

Each operating environment (such as Microsoft Windows or the Macintosh) has certain standards that graphical applications are expected to conform to. Windows, menus, and controls are supposed to look and behave in predictable ways from application to application.

FOR INFO This chapter describes some of the guidelines you should follow when designing windows and applications, but a full discussion is beyond the scope of this book. You should acquire a book that specifically addresses design guidelines for applications on your target platform and apply the rules when you use PowerBuilder to create your application. You might want to look at the following:

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

Building windows

When you build a window, you:

- ◆ Specify the appearance and behavior of the window by setting its properties.
- ◆ Add controls to the window.
- ◆ Build scripts that determine how to respond to events in the window and its controls. To support these scripts, you can define new events for the window and its controls, and declare functions, structures, and variables for the window.

Two ways

There are two ways to build a window. You can:

- ◆ Build a new window from scratch. You use this technique to create windows that aren't based on existing windows.
- ◆ Build a window that inherits its style, events, functions, structures, variables, and scripts from an existing window. You use inheritance to create windows that are derived from existing windows, thereby saving you time and coding.

For more information

For information on building windows from scratch, see "Building a new window" on page 169.

For information on using inheritance to build a window, see "Using inheritance to build a window" on page 190.

Types of windows

PowerBuilder provides the following types of windows:

- ◆ Main
- ◆ Popup
- ◆ Child
- ◆ Response
- ◆ Multiple Document Interface (MDI) frame
- ◆ MDI frame with MicroHelp

FOR INFO For a comparison of the names of window types on Macintosh systems with those PowerBuilder uses, see "Window types on the Macintosh" on page 166.

Main windows

Main windows are standalone windows that are independent of all other windows. They can overlap other windows and can be overlapped by other windows.

You use a main window as the anchor for your application. The first window your application opens is a main window—unless you are building a Multiple Document Interface (MDI) application, in which case the first window is an MDI frame.

On UNIX

On UNIX, if you open a window of type main as a child of another window, clicking on its parent can place the parent on top of the child. This is because a main window in Motif is a top-level widget that can be raised or lowered independently of its parent.

FOR INFO For more on building MDI applications, see *Application Techniques*.

Using main windows

Define your independent windows as main windows.

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

Popup windows

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

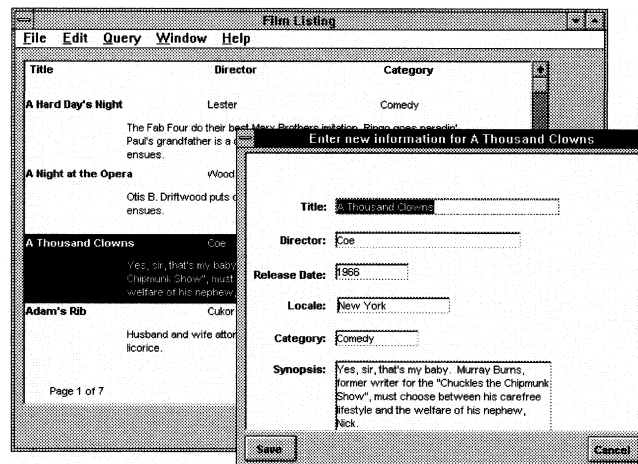
Using the application's Open event

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

A popup window can display outside its parent window. It cannot be overlaid by its parent. A popup window is hidden when its parent is minimized and when its parent is closed. When you minimize a popup window, the button for the window displays at the bottom of the screen.

Using popup windows

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



Explicitly naming a parent

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

```
Open (w_popup)
```

That statement defines `w_go` as the parent of `w_popup`.

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

```
Open ( popupwindow, parentwindow )
```

For example, the following statement:

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

opens `w_popup` and makes `w_parent` its parent.

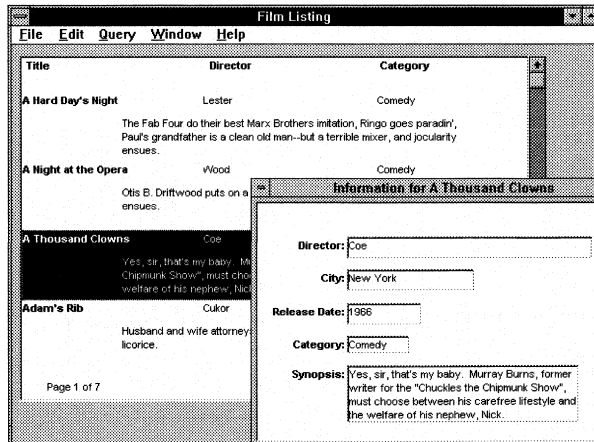
However, there are also other considerations regarding which window becomes the parent of an opened window.

FOR INFO For more information, see the `Open` function in the *PowerScript Reference*.

Child windows

Child windows are always opened from within a main or popup window, which becomes the child window's parent.

A child window exists only within its parent. You can move the child window within the parent window, but not outside the parent. When you move a portion of a child window beyond the parent, PowerBuilder clips the child so only the portion within the parent window is visible. When you move the parent window, the child window moves with the parent and maintains the same position relative to the parent.



Child windows cannot have menus and are never considered the active window. They can have title bars and can be minimizable, maximizable, and resizable. When they are maximized, they fill the space of their parent; when they are minimized, their button displays at the bottom of their parent.

The initial position of the child is relative to the parent and not to the entire screen. A child window closes when you close its parent.

Using child windows

Child windows are used by the PowerBuilder window plug-in and PowerBuilder window ActiveX tools that you can use when you are building applications for the internet.

FOR INFO For more about these tools, see *Building Internet Applications with PowerBuilder*.

You will probably not use child windows very often in other applications. Typically, if you want to display windows inside other windows, you will write MDI applications, where much of the window management happens automatically.

FOR INFO For more on building MDI applications, see *Application Techniques*.

Response windows

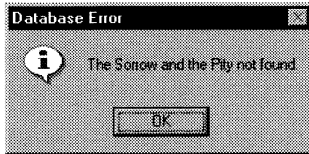
Response windows request information from the user. They are always opened from within another window (its parent). Typically, a response window is opened after some event occurs in the parent window.

Response windows are **application modal**. That is, when a response window displays, it is the active window (it has focus) and no other window in the application is accessible until the user responds to the response window. The user *can* go to other applications. But when the user returns to the application, the response window is still active.

Response windows act like modal popup windows.

Using response windows

For example, if you want to display a confirmation window when a user tries to close a window with unsaved changes, use a response window. The user is not allowed to proceed until the response window is closed.



Using message boxes

PowerBuilder also provides message boxes, which are predefined windows that act like response windows in that they are application modal. You open message boxes through the PowerScript function `MessageBox`.

FOR INFO For more information, see `MessageBox` in the *PowerScript Reference*.

MDI frames

An MDI window is a frame window in which you can open multiple document windows (sheets) and move among the sheets. There are two types of MDI frame windows: MDI frame and MDI frame with MicroHelp.

FOR INFO For more on building MDI applications, see *Application Techniques*.

FOR INFO For more about using MDI components on the Macintosh, see "MDI frames on the Macintosh" on page 167.

Window types on the Macintosh

The following table shows the correspondence between window types on the Macintosh and the terminology PowerBuilder uses.

Macintosh window type	PowerBuilder window type
Document window	Main window It can have a menu, which displays at the top of the screen (not at the top of the window as in PowerBuilder on the Windows platform)
Modal dialog	Response window without a title bar
Movable modal dialog	Response window with a title bar

Macintosh window type	PowerBuilder window type
Modeless dialog	Popup window (a popup cannot be overlaid by its parent window; it closes when the parent closes)
Alert	Not a PowerBuilder window type. Use the MessageBox function

The behavior of each type of window is the same as under Microsoft Windows, with the following differences:

- ◆ Child windows, although not a typical Macintosh window type, behave as they do in Windows
- ◆ An MDI frame is not visible—instead, the whole desktop is the MDI frame

MDI frames on the Macintosh

PowerBuilder makes it easy to implement applications with a multiple-document interface (MDI) typical of Windows applications. You can use MDI on the Macintosh and retain the look and feel of a Macintosh application because PowerBuilder makes some adjustments in how it displays the MDI frame.

The following table explains how the MDI components work on the Macintosh.

MDI component	Macintosh equivalent and behavior
MDI frame	Macintosh desktop The frame opens but is invisible. It occupies the full desktop, with a menu at the top of the screen and optionally a MicroHelp status bar at the bottom
Sheet	Document window A sheet is a PowerBuilder main window opened as a sheet within an MDI frame

When to use them In an application with several document windows, you will want to use an MDI frame or an MDI frame with MicroHelp. After you open the MDI frame, you open the main windows as sheets. Although the frame is not visible on the Macintosh, a frame with sheets is still a good application design. The frame is a conceptual container for the application, and you can take advantage of PowerBuilder's functions for managing open sheets.

Both the sheets and the frame can have menus. When you use a frame, you can control the way menus get associated with windows. If you don't have a frame and you open several main windows, each with its own menu, the menu for the last opened window is always the current menu, even if the window it belongs to is closed.

When there is a frame, as each open sheet is activated, the appropriate menu becomes current too. If a sheet doesn't have a menu, it uses the menu that is active when it is opened. To keep the association between menus and sheets clear, it is best to have either of the following:

- ◆ A menu for the frame only. That menu serves all the sheets.
- ◆ A menu for every sheet. There is never a sheet that gets a menu based on when it is opened.

MicroHelp If a window's type is MDI Frame with MicroHelp, PowerBuilder adds a status bar at the bottom of the screen. If your menu items have MicroHelp text associated with them, then when the user highlights the menu item, its MicroHelp displays in the status bar. You can also use the SetMicroHelp function to change the text in the status bar.

Controls and custom MDI frames A custom frame is a frame on which you've placed controls. Custom frames are not supported on the Macintosh. The frame background is invisible—controls you place on the frame won't be displayed.

Opening more than one frame Opening more than one frame window is not recommended on any PowerBuilder platform.

Building a new window

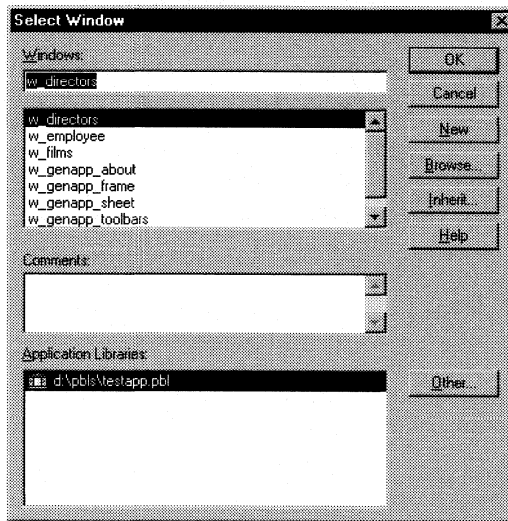
This section describes how to build windows from scratch. You will use this technique to create windows that are not based on existing windows.

Opening the Window painter

❖ **To open the Window painter:**

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

The Select Window dialog box displays listing the windows in the current library.

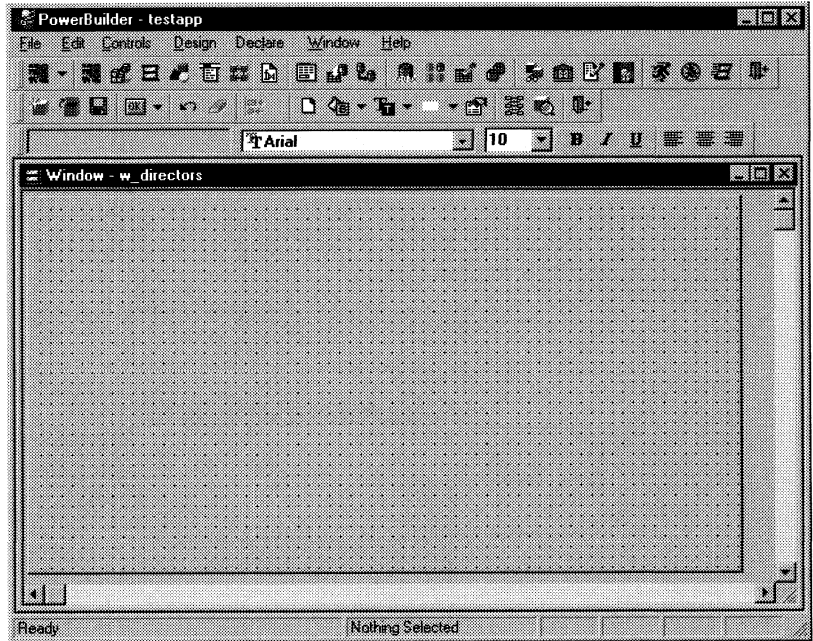


- 2 Click the New button to build a new window.

The Window painter workspace displays.

About the painter

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



About the PainterBar

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

FOR INFO For complete information about manipulating the PainterBar, see Chapter 1, "Working with PowerBuilder".

About the StyleBar

The Window painter has a StyleBar that you use to assign properties to text.

FOR INFO For complete information about manipulating the StyleBar, see Chapter 1, "Working with PowerBuilder".

Working in the Window painter

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

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

Defining the window's properties

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

A window's style encompasses its:

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

About defining a window's style

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

FOR INFO For a complete list of window properties, see *Objects and Controls*.

You define a window's style by choosing different settings on the Window property sheet pages.

❖ **To specify window properties:**

- 1 Select Properties from the window's popup menu.

or

Double-click the window's background.

or

Select Edit>Properties from the menu bar.

The Window property sheet displays.

- 2 Choose the tab appropriate to the property you want to specify:

To specify the window's	Choose this tab
Name, type, style, color, and whether a menu is associated with it	General
Position and size when it displays at execution time	Position
Default cursor whenever the mouse moves over the window	Pointer
Icon to represent the window when you minimize it	Icon
Horizontal and vertical scrollbar placement	Scroll
Toolbar placement	Toolbar

All of these properties are discussed below.

Using the General property page

Use the General property page to specify the following window information:

- ◆ Window type
- ◆ Title bar text
- ◆ Menu name
- ◆ Color

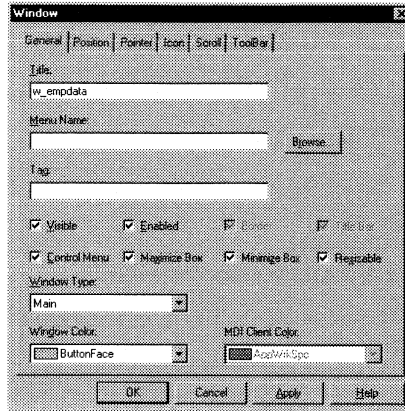
Specifying the window's type

The first thing you should do is specify the type of window you are creating.

❖ To specify the window's type:

- 1 Choose Properties from the window's popup menu and select the General tab.

The General property page displays.



- 2 Select the appropriate window type from the Window Type dropdown listbox.
- 3 Click Apply to view your changes.
or
Click OK to save your changes.

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

Specifying other basic window properties

By selecting and deselecting checkboxes on the General property page, you can specify whether the window is sizable or minimizable, is enabled, has a border, and so on.

Note the following:

- ◆ A main window must have a title bar
- ◆ A child window cannot have a menu
- ◆ A response window cannot have a menu, Minimize box, or Maximize box

On Macintosh systems, some style settings have a different effect or no effect at all.

Style setting	Result when checked on Macintosh systems
Control Menu	For most window types, displays the close box in the window's upper-left corner If the window's type is response, Control Menu has no effect
Resizable	Displays a grow box in the upper right and a resize box in the lower right
Maximize Box	No effect
Minimize Box	No effect

Associating a menu with the window

❖ **To associate a menu with the window:**

- 1 Enter the name of the menu in the Menu Name textbox on the General property page.
or
Click the Browse button and select the menu from the Select Menu dialog box, which displays a list of all menus available to the application.
- 2 Click Apply to view your changes.
or
Click OK to save your changes.

Changing the menu

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

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

Choosing a window color

❖ **To specify the color of a window:**

- ◆ Do either of the following:
 - ◆ Specify the color of the window from the Window Color dropdown listbox on the General property page.

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

- ◆ If the window is an MDI window, specify a color in the MDI Client Color dropdown listbox.

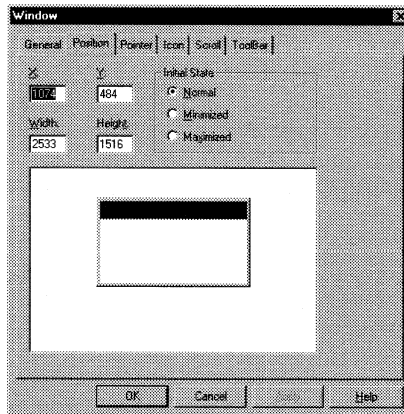
FOR INFO For more about using colors in windows, including how to define your own custom colors, see Chapter 8, "Working with Controls".

Choosing the window's size and position

❖ To move a window:

- 1 Choose Properties from the window's popup menu and select the Position tab.

The Position property page displays.



- 2 Move the window as appropriate. You can:
 - ◆ Drag the representation of the window with the mouse
 - ◆ Center the window in the screen by displaying the popup menu in the window representation pane and choosing Center Vertically or Center Horizontally
 - ◆ Enter values for x and y location (in PowerBuilder units—see below)
- 3 Click OK.

About x and y values

For main, popup, response, and MDI frame windows, x and y locations are relative to the upper-left corner of the screen. For child windows, x and y are relative to the parent.

❖ To resize a window:

- ◆ Drag the edge of the window in the Window painter workspace.
or
Drag a corner of the window's representation on the Position property page.

You can also change the window's Width and Height properties on the Position property page.

About PowerBuilder units

All window measurements are in PowerBuilder units (PBUs). Using these units, you can build applications that look similar on different resolution screens. A PBU is defined in terms of logical inches. The size of a logical inch is defined by your operating system as a specific number of pixels. The number is dependent on the display device. For example, on a Windows 95 system using Small Fonts on a standard VGA display, the number of pixels per logical inch is 96.

Almost all sizes in the Window painter and in scripts are expressed as PowerBuilder units. (The two exceptions are text size, which is expressed in points, and grid size in the Window and DataWindow painters, which is in pixels.)

FOR INFO For more about PowerBuilder units, see the *PowerScript Reference*.

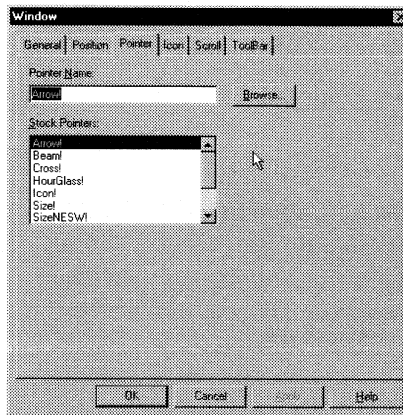
Choosing the window's pointer

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

❖ **To choose the window pointer:**

- 1 Choose Properties from the window's popup menu and select the Pointer tab.

The Pointer property page displays.



- 2 Choose the pointer from the Stock Pointers dropdown listbox or use the Browse button to select a cursor (CUR) file.

The pointer you chose displays next to the Stock Pointers listbox.

- 3 Click OK.

Specifying the pointer for a control

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

Choosing the window's icon

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

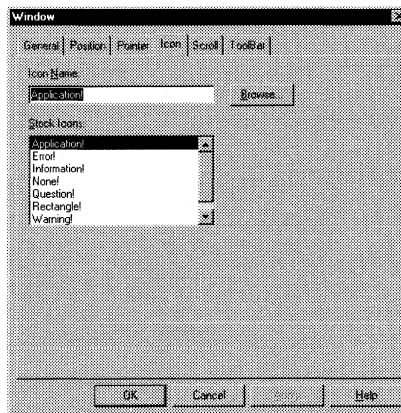
On Macintosh and UNIX

If you are developing on the Macintosh or UNIX platforms, the icon you specify will be used if you deploy your application on Windows.

❖ To choose the window icon:

- 1 Choose Properties from the window's popup menu and select the Icon tab.

The Icon property page displays.



- 2 Choose the icon from the Stock Icons dropdown listbox or use the Browse button to select an icon (ICO) file.

The icon you chose displays next to the Stock Icons listbox.

- 3 Click OK.

Changing the icon during execution

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

Specifying window scrolling

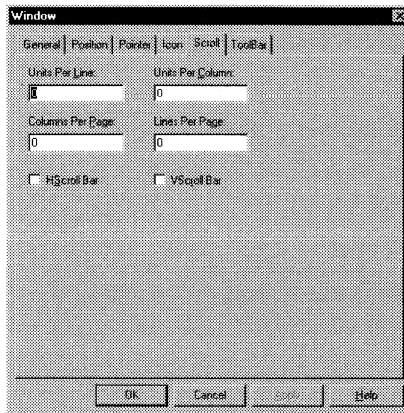
If your window is resizable, it is possible that not all the window's contents will be visible during execution. In such cases, you should make the window scrollable by providing scrollbars. You do this on the Scroll property page.

By default, PowerBuilder controls scrolling when scrollbars are present. You can control the amount of scrolling.

❖ To specify window scrolling:

- 1 Choose Properties from the window's popup menu and select the Scroll tab.

The Scroll property page displays.



- 2 Indicate which scrollbars you want to display by selecting the HScroll Bar and VScroll Bar checkboxes.
- 3 Specify scrolling characteristics as follows:

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

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

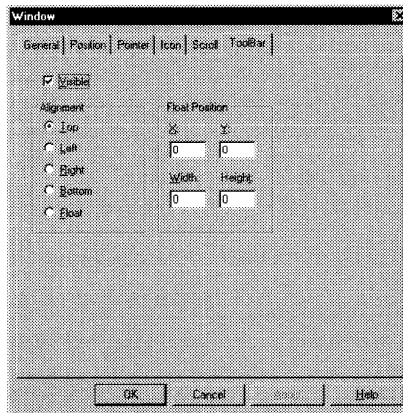
Specifying toolbar properties

You can specify whether or not you want to display a menu toolbar (if the menu you associate with your window assigns toolbar buttons to menu objects) in your window. If you choose to display the toolbar, you can specify the location for it.

❖ **To specify toolbar properties:**

- 1 Choose Properties from the window's popup menu and select the ToolBar tab.

The ToolBar property page displays.



- 2 To display the toolbar with your window, select the Visible checkbox.
- 3 Set the location of the toolbar by selecting one of the radio buttons in the Alignment group.

- 4 If you choose Float as your toolbar alignment, you must set the following values in the Float Position group:
 - ◆ X and Y coordinates for the toolbar
 - ◆ Width and Height for the toolbar

FOR INFO For more information about defining toolbars, see Chapter 10, "Working with Menus".

Adding controls

When you build a window, you place controls (such as CheckBoxes, CommandButtons, and MultiLineEdits) in the window to request and receive information from the user and to present information to the user.

After you place a control in the window, you can define its style, move and resize it, and write scripts to determine how the control responds to events.

FOR INFO For more information, see Chapter 8, "Working with Controls".

Saving the window

You can save the window you are working on anytime.

❖ To save a window:

- 1 Select File>Save from the menu bar.

If you have previously saved the window, PowerBuilder saves the new version in the same library and returns you to the Window painter workspace.

If you have not previously saved the window, PowerBuilder displays the Save Window dialog box.

- 2 Name the window in the Windows textbox (see below).
- 3 Type comments in the Comments textbox to describe the window.

These comments display in the Select Window window and in the Library painter. It is a good idea to use comments so you and others can easily remember the purpose of the window later.

- 4 Specify the library to save the window in.
- 5 Click OK.

Naming the window

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

FOR INFO For information about PowerBuilder identifiers, see the *PowerScript Reference*.

A recommendation When you name windows, you should use a two-part name: a standard prefix that identifies the object as a window (such as w_) and a suffix that helps you identify the particular window.

For example, you might name a window that displays employee data w_empdata.

Viewing your work

While building a window, you can preview it and print its definition.

Previewing a window

As you develop your window, you can preview its appearance from within the Window painter. By previewing the window, you get a good idea of how it will look during execution.

❖ **To preview a window:**

- ◆ Click the Preview button.

or

Select Design>Preview from the menu bar.

The window displays with the properties you have defined, such as title bar, menu, Minimize box, and so on.

Toggle buttons

Like many buttons in PowerBuilder, the Preview button is a toggle. When you click on it to preview an object, the button stays pressed while the object is being previewed. Clicking the Preview button again returns it (and the object you are previewing) to its normal state.

What you can do

While previewing the window, you can get a sense of its look and feel. You can:

- ◆ Move the window
- ◆ Resize it (if it is resizable)
- ◆ Maximize, minimize, and restore it (if these properties were enabled)
- ◆ Tab around the window
- ◆ Select controls

What you cannot do

You cannot:

- ◆ Change properties of the window (changes you make while previewing the window, such as resizing it, are not saved)
- ◆ Trigger events (for example, clicking a CommandButton while previewing a window does not trigger its Clicked event)
- ◆ Connect to a database

- ❖ **To return to the Window painter:**
 - ◆ Click the Preview button.
or
Select Design>Preview from the menu bar.

Printing a window's definition

You can print a window's definition for documentation purposes.

- ❖ **To print information about the current window:**
 - ◆ Select File>Print from the menu bar.

Information about the current window is sent to the printer specified in Printer Setup. The information sent to the printer depends on variables specified in the [Library] section of the PowerBuilder initialization file.

Print settings

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

Writing scripts in windows

You write scripts for window and control events. To support these scripts, you can define:

- ◆ Window-level and control-level functions
- ◆ Instance variables for the window

About events for windows and controls

Windows themselves have several events, including `Open`, which is triggered when the window is opened (before it is displayed), and `Close`, which is triggered when the window is closed.

For example, you might connect to a database and initialize some values in the window's `Open` event and disconnect from a database in the `Close` event.

Each type of control also has its own set of events. Buttons, for example, have `Clicked` events, which trigger when a user clicks the button. `SingleLineEdits` and `MultiLineEdits` have `Modified` events, which trigger when the contents of the edit control change.

Defining your own events

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

For example, assume you offer your user several ways to update the database from a window, such as clicking a button or selecting a menu item. In addition, when the user closes the window, you want to update the database as well after asking for confirmation. So here you want the same type of processing to happen after different events.

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

FOR INFO To learn how to use user events, see Chapter 12, "Working with User Events".

About functions for windows and controls

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

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

Passing parameters

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

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

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

FOR INFO For more information, see Chapter 5, "Working with User-Defined Functions".

About properties of windows and controls

In scripts, you can assign values to the properties of objects and controls to change their appearance or behavior. You can also test the values of properties to obtain information about the object.

For example, you can change the text displayed in a `StaticText` control when the user clicks a `CommandButton`, or use data entered in a `SingleLineEdit` to determine 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

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

The reserved word Parent

In the script for a 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's easier to reuse a script if you use `Parent` instead of the name of the window.

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

Declaring instance variables

Often you have data that needs to be accessible in several scripts within a window. For example, assume a window displays information about one customer. You might want several `CommandButtons` to manipulate the data; each of them needs to know the customer's ID. There are several ways to accomplish this:

- ◆ Declare a global variable containing the current customer ID.
All scripts in the application have access to this variable.
- ◆ Declare an instance variable within the window.
All scripts for the window and for controls in the window have access to this variable.
- ◆ Declare a shared variable within the window.
All scripts for the window and its controls have access to this variable. In addition, all other windows of the same type have access to the same variable.

The best approach

When declaring variables, you need to consider what the scope of the variable is. If the variable is only meaningful within a window, declare it as a window-level variable, generally an instance variable. If the variable is meaningful throughout the entire application, make it a global variable.

For more information

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

Examples of statements

The following assignment statement in the script for the Clicked event for a CommandButton changes the text in the StaticText object st_greeting when the button is clicked:

```
st_greeting.Text = "Hello User"
```

The following statement tests the value entered in the SingleLineEdit sle_state and displays the window w_state1 if the text is "AL":

```
if sle_State.Text= "AL" then Open(w_state1)
```


Running a window

During development, you can test a window without running the whole application.

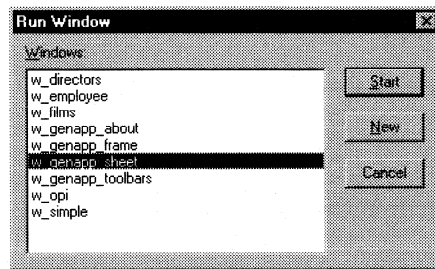
❖ **To run a window:**

- 1 Click the Run Window button.

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

- 2 If necessary, save your work.

The Run Window dialog box displays.



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

PowerBuilder runs the window.

You can trigger events, open other windows, connect to a database, and so on when running a window. The window is fully functional, except that it does not have access to global variables that you have defined for the application (though it *does* have access to built-in globals, such as SQLCA). Also, the SystemError event is not triggered if there is an error, because SystemError is an Application object event.

Using inheritance to build a window

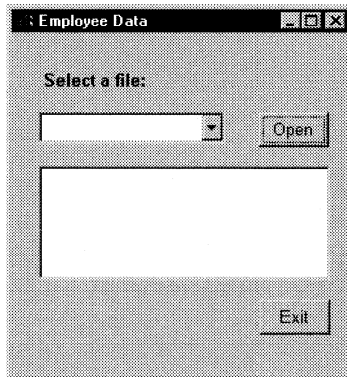
When you build a window that inherits its definition (style, events, functions, structures, variables, controls, and scripts) from an existing window, you save coding time. All you have to do is modify the inherited definition to meet the requirements of the current situation.

Example

Assume your application has a window `w_employee` that has:

- ◆ A title (Employee Data)
- ◆ Text that says Select a file:
- ◆ A dropdown listbox with a list of available employee files
- ◆ An Open button with a script that opens the selected file in a multiline textbox
- ◆ An Exit button with a script that asks the user to confirm closing the window and then closes the window

The window looks like this:



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

Your choices

To build this window, you have three choices:

- ◆ Build a new window from scratch as described in "Building a new window" on page 169
- ◆ Modify the existing window (`w_employee`), then save it under another name
- ◆ Use inheritance to build a window that inherits the definition from the existing window (in other words, build a descendent window)

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

Advantages of using inheritance

Using inheritance has a number of advantages:

- ◆ When you change the ancestor window, the changes are reflected in all descendants of the window. You do not have to manually make changes in the descendants as you would in a copy. This saves you coding time and makes the application easier to maintain.
- ◆ The descendant inherits the ancestor's scripts, so you do not have to re-enter the code to add to the script.
- ◆ You get consistency in the code and in the application windows.

Using the example

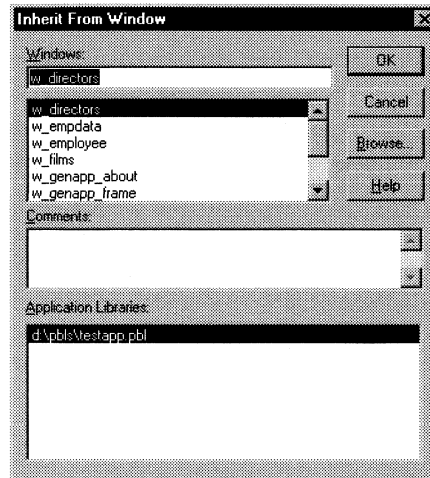
Continuing with the example, the best way to build the window to display customer information instead of employee information is to build a window that is inherited from `w_employee`. You change the title and modify the scripts, then save the descendent window using a new name (such as `w_customer`).

❖ To use inheritance to build a descendent window:

- 1 Open the Window painter.
The Select Window dialog box displays.

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

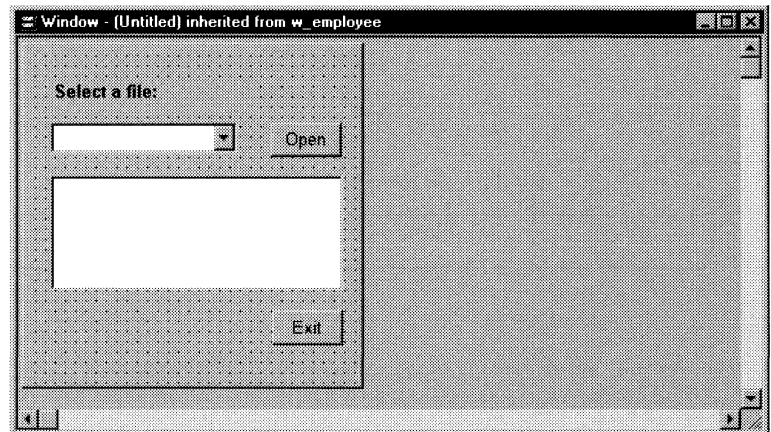
The Inherit From Window dialog box displays.



- 3 Double-click the window you want to use to build the new descendent window.

or

Select the window and click OK.



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

- 4 Make the changes to the descendent window and save the window with a new name.

What inherited objects contain	When you use inheritance to build an object, everything in the ancestor object is inherited in all its descendants.
What you can do in the descendent window	<p>You can:</p> <ul style="list-style-type: none">◆ Change the properties of the window◆ Add controls to the window and modify existing controls◆ Size and position the window and the controls in the window◆ Build new scripts for events in the window or its controls◆ Reference the ancestor's functions and events◆ Reference the ancestor's structures if the ancestor contains a public or protected instance variable of the structure data type◆ Access ancestor properties, such as instance variables, if the scope of the property is public or protected◆ Extend or override inherited scripts◆ Declare functions, structures, and variables for the window◆ Declare user events for the window and its controls
What you cannot do	The only thing you cannot do is delete inherited controls.

Unneeded inherited controls

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

About control names	<p>PowerBuilder uses this syntax to show names of inherited controls:</p> <p style="text-align: center;"><i>ancestorwindow::control</i></p> <p>For example, in <code>w_customer</code>, which is inherited from <code>w_employee</code>, if you open the property sheet for the Open button, you see that its name displays as:</p> <p style="text-align: center;"><code>w_employee::cb_open</code></p> <p>Names of controls must be unique in an inheritance hierarchy. For example, you cannot have a <code>CommandButton</code> named <code>cb_close</code> defined in an ancestor and a different <code>CommandButton</code> named <code>cb_close</code> defined in a child.</p> <p>So you should develop a naming convention for controls in windows that you plan to use as ancestors.</p>
---------------------	--

For more information

The issues concerning inheritance with windows are the same as the issues concerning inheritance with user objects and menus. Chapter 9, "Understanding Inheritance", describes the following issues in detail:

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

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

Topic	Page
Overview of controls	196
Placing controls in a window	197
Selecting controls	198
Defining a control's properties	199
Naming controls	201
Changing text	205
Moving and resizing controls	207
Copying controls	210
Defining the tab order	211
Defining accelerator keys	214
Specifying accessibility of controls	217
Choosing colors	218
Using the 3D look	220
Using the individual controls	221

Overview of controls

There are two types of controls:

- ◆ Controls that have events
- ◆ Controls that do not have events (these are the **drawing objects**)

About controls with events

Users can act on controls that have events. You write scripts that determine the processing that takes place when an event occurs in the control. The controls that have events are:

CheckBox	HScrollBar	RadioButton
CommandButton	ListBox	RichTextEdit
DataWindow	ListView	SingleLineEdit
DropDownListBox	MultiLineEdit	StaticText
DropDownPictureListBox	OLE 2.0	Tab
EditMask	Picture	TreeView
Graph	PictureButton	User Object
GroupBox	PictureListBox	VScrollBar

Creating user objects

If you often use a control or set of controls with certain properties (such as a group of related radio buttons), you might want to create a user object, which is one or more controls with predefined behavior that can be placed in a window.

FOR INFO For more about user objects, see Chapter 11, "Working with User Objects".

About the drawing objects

You use the drawing objects to enhance the design of your window. The drawing objects are:

Line	Rectangle
Oval	RoundRectangle

Placing controls in a window

❖ To place a control in a window:

- 1 Select the control from the Window painter PainterBar or from the Controls menu.

If you select a user object control, the Select User Object dialog box displays listing all user objects defined for the application. Select the user object and click OK.

- 2 Click where you want the control.

After you place the control, you can size it, move it, define its appearance and behavior, and create scripts for its events.

Duplicating controls

To place multiple controls of the same type in a window, place a control in the window and make sure it is selected. Then press CTRL+T (COMMAND+T on the Macintosh) or select Duplicate from the popup menu once for each duplicate control you want to place in the window. The controls are placed one under another. You can drag them to other locations if you want.

Placing DataWindow controls, Pictures, and PictureButtons

When you place a DataWindow control, a picture, or a PictureButton in a window, you are placing a control in the window. No DataWindow object or picture has yet been specified for the control.

If the control is a	You see	For more information
DataWindow control	An empty box to indicate no DataWindow object has been specified	Chapter 14, "Defining DataWindow Objects"
Picture	The dotted outline of a box to indicate no picture has been specified	"Using pictures" on page 235
PictureButton	A large button (resembling a CommandButton) with space at the top to indicate no picture has been specified	"Using PictureButtons" on page 223

Placing OLE controls

On Windows, you can place objects from applications that support OLE, such as Excel worksheets and Visio drawings, in your windows.

FOR INFO For information about using OLE with PowerBuilder, see *Application Techniques*.

Selecting controls

❖ **To select a control:**

- ◆ Click it.

The control displays with handles on it. Previously selected controls are no longer selected.

❖ **To select neighboring multiple controls:**

- 1 Press and hold the left mouse button at one corner of the neighboring controls.
- 2 Drag the mouse over the controls you want to select.
PowerBuilder displays a bounding box.
- 3 Release the mouse button.

All the controls in the region are selected and have handles.

❖ **To select non-neighboring multiple controls:**

- 1 Click the first control.
- 2 Press and hold the CTRL key and click additional controls.

On Macintosh

On the Macintosh, use the COMMAND key instead of the CTRL key.

All the controls are selected and have handles.

Acting on multiple controls

You can act on all selected controls as a unit. For example, you can move all of them or change the fonts for all the text displayed in the controls.

Another way to select controls

You can also display a list of controls and select one from the list by selecting Edit>Control List from the menu bar.

You can select all controls by selecting Edit>Select All from the menu bar.

Information displayed about the selected control

The name, x and y coordinates, width, and height of the selected control are displayed in the MicroHelp bar.

If multiple objects are selected, *Group Selected* displays in the Name area and the coordinates and size do not display.

Defining a control's properties

Just like the window object, each control has properties that determine how the control looks and behaves during execution (the control's **style**).

You define a control's properties by using the control's property sheet.

❖ To define a control's properties:

- 1 Display the popup menu as usual and select Properties.
The property sheet for the selected control displays.
- 2 Use the property sheet tab pages to change the control's properties.

About the property sheet

Each type of control has its own set of properties and thus its own property sheet.

Property sheets present information in a consistent arrangement of tab pages on a sheet. All controls have a General property page, which contains much of the style information (such as the visibility of the control, whether it is enabled, and so on) about the control. The General property page is always the first page on the property sheet.

You select items on the individual tab pages to change the control's definition.

Property sheet tab pages

The following table lists all the property pages in each control's property sheet:

Control	Property sheet tab pages
CheckBox CommandButton GroupBox MultiLineEdit PictureButton RadioButton SingleLineEdit StaticText	General, Font, Position, Pointer, Drag and Drop
DataWindow	General, Position, Pointer, Icon, Drag and Drop
Drawing object	General, Position
EditMask	General, Mask, Font, Position, Pointer, Drag and Drop
Graph	General, Position, Pointer, Drag and Drop, Graph, Series Axis, Category Axis, Value Axis, Text
ListBox DropDownListBox	General, Items, Font, Position, Pointer, Drag and Drop

Control	Property sheet tab pages
List View	General, LargePicture, SmallPicture, State, Items, Font, Position, Pointer, Drag and Drop
Picture HScrollBar VScrollBar	General, Position, Pointer, Drag and Drop
PictureListBox DropDownPictureListBox	General, Pictures, Items, Font, Position, Pointer, Drag and Drop
RichText	General, Document, Position, Drag and Drop
Tab	General, Page Order, Font, Position, Pointer, Drag and Drop
TreeView	General, Picture, State, Font, Position, Pointer, Drag and Drop
UserObject OLE	Dependent on the type of object

Naming controls

When you place a control in a window, PowerBuilder assigns it a unique name. The name is the concatenation of the default prefix for the control name and the lowest 1- to 4-digit number that makes the name unique.

For example, assume the prefix for ListBoxes is lb_ and you add a ListBox to the window:

- ◆ If the names lb_1, lb_2, and lb_3 are currently used, the default name is lb_4
- ◆ If lb_1 and lb_3 are currently used but lb_2 is not, the default name is lb_2

About the default prefixes

Each type of control has a default prefix for its name. The initial default prefix for each control is listed below (note that there is no prefix for a window):

Control	Prefix
CheckBox	cbx_
CommandButton	cb_
DataWindow	dw_
DropDownListBox	ddlb_
DropDownPictureListBox	ddplb_
EditMask	em_
Graph	gr_
GroupBox	gb_
HScrollBar	hsb_
Line	ln_
ListBox	lb_
ListView	lv_
MultiLineEdit	mle_
OLE 2.0	ole_
Oval	oval_

Control	Prefix
Picture	p_
PictureButton	pb_
PictureListBox	plb_
RadioButton	rb_
Rectangle	r_
RichTextEdit	rte_
RoundRectangle	rr_
SingleLineEdit	sle_
StaticText	st_
Tab	tab_
TreeView	tv_
User Object	uo_
VScrollBar	vsb_

Changing the default prefixes

You can change the default prefixes for controls in the Window painter's Options property sheet. Select Design>Options from the menu bar to open the Options property sheet. The changes you make are saved in the PowerBuilder initialization file.

FOR INFO For more about the PowerBuilder initialization file, see "Managing the initialization file" on page 33.

Changing the name

You should change the default suffix to a suffix that is meaningful in your application and develop a naming convention for control names.

For example:

Instead of	You could use
cb_6	cb_retrieve
cbx_1	cbx_enabled

Instead of	You could use
dw_1	dw_EmployeeData
sle_2	sle_LName

Using these conventions makes it much easier for you to understand scripts you write to manipulate these controls.

Using application-based names instead of sequential numbers also minimizes the likelihood that you will have name conflicts when you use inheritance to create windows.

❖ **To change a control's name:**

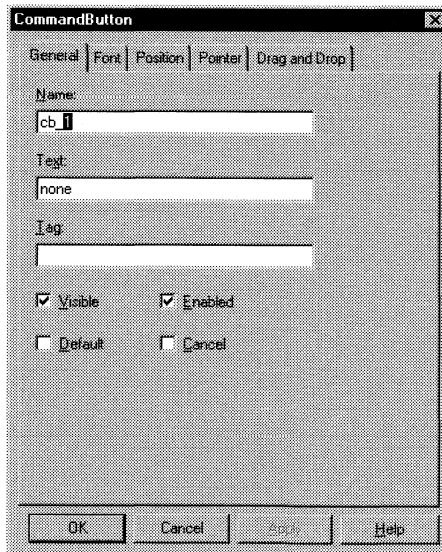
- 1 Choose Properties from the control's popup menu and select the General tab as described above.

The control's General property page displays with the default suffix selected.

Shortcut

You can also display a control's property sheet by double-clicking the control.

Here is the property sheet for a CommandButton:



- 2 Type the application-specific suffix.

What you type replaces the selected default suffix.

You can use any valid PowerBuilder identifier with up to 40 characters.

FOR INFO For information about PowerBuilder identifiers, see the *PowerScript Reference*.

- 3 Click OK.

Changing text

There are two ways to manipulate text for a control:

- ◆ Using property sheet pages to specify text and font for a control
- ◆ Using 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 (except that text in CommandButtons is always center aligned)

❖ **To change text properties of controls:**

- 1 Select each control whose properties you want to change.
- 2 Specify the new properties using the StyleBar.
or
Specify changes on the control's property sheet Font page.

Choosing fonts

Make sure that fonts you pick will be available to your users when you distribute your application. You can use the PowerBuilder font mapping file to map the fonts you use to fonts available on the deployment computer. If you pick a font that is on your computer but not on your user's computer and that is not mapped in the font mapping file, the operating environment will use the font it considers closest to the one you specified.

FOR INFO For more about font mapping files, see *Application Techniques*.

On UNIX

Underlining is not supported in Motif applications.

How text size is stored

A control's text size is specified in the control's `TextSize` property. PowerBuilder saves the text size in points, using negative numbers. For example, if you define the text size for the `StaticText` control `st_prompt` to be 12 points, PowerBuilder sets the value of `st_prompt`'s `TextSize` property to -12. (PowerBuilder uses negative numbers to record point size for compatibility with previous releases, which saved text size in pixels as positive numbers.)

So if you want to change the point size of text during execution in a script, remember to use a negative value. For example, to change the point size for `st_prompt` to 14 points, code:

```
st_prompt.TextSize = -14
```

You can specify text size in pixels if you want, by using positive numbers. The following statement sets the text size to be 14 pixels:

```
st_prompt.TextSize = 14
```

Moving and resizing controls

You can move or resize a control using the mouse or the keyboard.

Using the mouse

To move a control, drag it with the mouse to where you want it.

To resize a control, select it, then grab an edge and drag it with the mouse.

Using the keyboard

To move a control, select it, then press an arrow key to move it in the corresponding direction.

To resize a control, select it, and then:

To make the control	Press
Wider	SHIFT+RIGHT ARROW
Narrower	SHIFT+LEFT ARROW
Taller	SHIFT+DOWN ARROW
Shorter	SHIFT+UP ARROW

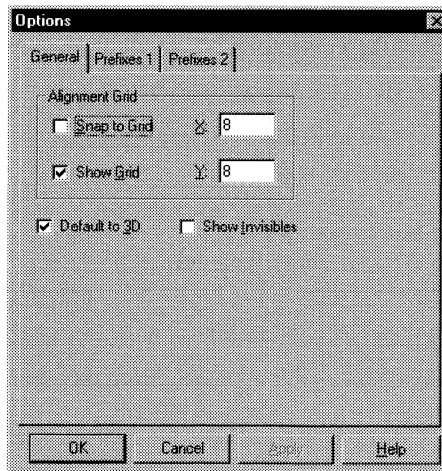
Using the grid

The Window painter provides a grid to help you align controls.

❖ To use the grid:

- 1 Choose Design>Options from the menu bar and select the General tab.

The General property page displays.



- 2 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

The grid options are:

Option	Meaning
Snap to Grid	If selected, controls are aligned with a horizontal and vertical grid line when you place or move them
Show Grid	If selected, the grid is displayed in the workspace
X	The width of each cell in the grid in pixels
Y	The height of each cell in the grid in pixels

Grid tips

Window painting is slower when the grid is displayed, so you may want to display the grid only when necessary.

Aligning controls

It is easy to align two or more controls.

❖ To align controls:

- 1 Select the control whose position you want to use to align the others. PowerBuilder displays handles around the selected control.
- 2 Press and hold the CTRL key and click the controls you want to align with the first one.

On Macintosh

On the Macintosh, use the COMMAND key instead of the CTRL key.

All the selected controls have handles on them.

- 3 Select Edit>Align Controls from the menu bar.

- 4 From the cascading menu, select the dimension along which you want to align the controls.

For example, to align the controls along the left side, select the first choice on the cascading menu.

PowerBuilder aligns all the selected controls with the first one.

Equalizing the space between controls

You can manually move controls by dragging them with the mouse. You can also easily equalize the spacing around specified controls.

❖ To equalize the space between controls:

- 1 Select the two controls whose spacing is correct.
To do so, select one control, then press and hold CTRL (COMMAND on the Macintosh) and click the second control.
- 2 Select the other controls whose spacing you want to be the same as the first two controls by pressing CTRL and clicking.
- 3 Select Edit>Space Controls from the menu bar.
- 4 From the cascading menu, select the horizontal or vertical spacing icon.

Equalizing the size of controls

Assume you have several CommandButtons and want them to be the same size. You can accomplish this manually or by using the Edit menu.

❖ To equalize the size of controls:

- 1 Select the control whose size is correct.
- 2 Select the other controls whose size you want to match the first control by pressing and holding CTRL and clicking.
- 3 Select Edit>Size Controls from the menu bar.
- 4 From the cascading menu, select the horizontal or vertical size icon.

Using the PainterBar

You can select the Layout dropdown toolbar from the PainterBar to size, space, and align controls.

Copying controls

You can copy controls within a window or to other windows. All properties of the control, as well as all of its scripts, are copied. You can use this technique to easily make a copy of an existing control and change whatever you want in the copy.

❖ **To copy a control:**

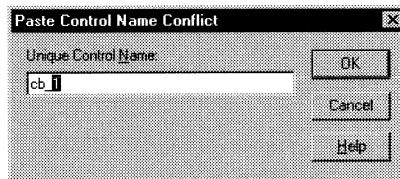
- 1 Select the control in the Window painter workspace.
- 2 Select Edit>Copy from the menu bar.
or
Press CTRL+C (COMMAND+C on the Macintosh).

The control is copied to a private PowerBuilder clipboard.

- 3 To copy the control within the same window, select Edit>Paste from the menu bar or press CTRL+V.

To copy the control to another window, open another instance of the Window painter and open the window in it. Make that window active and select Edit>Paste from the menu bar or press CTRL+V.

If the control you are pasting has the same name as a control that already exists in the window, the Paste Control Name Conflict dialog box displays.



- 4 If prompted, change the name of the pasted control to be unique.
PowerBuilder pastes the control in the destination window at the same location as in the source window (if you are pasting into the same window, you should move the pasted control so it doesn't overlay the original control). You can make whatever changes you want to the copy; the source control will be unaffected.

Defining the tab order

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

About user objects

When the user tabs to a custom user object in a window and then presses the TAB key, focus moves to the next control in the tab order for the user object. After the user tabs to all the controls in the tab order for the user object, focus moves to the next control in the window tab order.

On Macintosh

On the Macintosh, the TAB key changes the focus among the editable controls in a window, such as textboxes and listboxes. Other kinds of controls, such as buttons, never have focus. PowerBuilder assigns all controls a tab order value greater than 0 if they can have focus on any platform. If a control cannot have focus on the Macintosh, its tab value is ignored.

Establishing the default tab order

PowerBuilder uses the relative positions of controls in a window to establish the default tab order. It looks at the positions in the following order:

- ◆ The distance the control is from the top of the window (Y)
- ◆ The distance the control is from the left edge of the window (X)

The control with the smallest Y distance is the first control in the default tab order. If multiple controls have the same Y distance, PowerBuilder uses the X distance to determine the tab order among these controls.

Default tab values

The default tab value for drawing objects and RadioButtons in a GroupBox is 0 (skip the control).

When you add a control to the window, PowerBuilder obtains the tab value of the control that precedes the new control in the tab order and assigns the new control the next number.

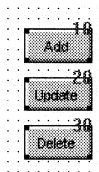
For example, if the tab values for controls A, B, and C are 30, 10, and 20 respectively and you add control D between controls A and B, PowerBuilder assigns control D the tab value 40.

Changing the window's tab order

❖ **To change the tab order:**

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

The current tab order displays. If this is the first time you have used Tab Order for the window, the default tab order displays.



- 2 Use the mouse or the TAB key to move the pointer to the tab value you want to change.
- 3 Enter a new tab value (0-9999).

The value 0 removes the control from the tab order (skips the control). It doesn't matter exactly what value you use (other than 0); all that matters is relative value. For example, if you want the user to tab to control B after control A but before control C, set the tab value for control B so it is between the value for control A and the value for control C.

Tab tips

A tab order of 0 does not prevent a control from being selected or activated or from receiving keyboard events. To prevent a user from activating a control with the mouse, clear the Enabled checkbox on its General property page.

To permit tabbing in a groupbox, change the tab value of the GroupBox to 0, then assign nonzero tab values to the controls in the GroupBox.

- 4 Repeat the procedure until you have the tab order you want.
- 5 Select Design>Tab Order from the menu bar again.

PowerBuilder saves the tab order.

Each time you select Tab Order, PowerBuilder renumbers the tabs to include any controls that have been added to the window and to allow space to insert new controls in the tab order.

Defining accelerator keys

You can define accelerator keys for your controls to allow users on Windows or UNIX to change focus to the control. An accelerator key is usually referred to as a mnemonic access key on Windows and a mnemonic on UNIX. There is no equivalent on the Macintosh.

On Windows On Windows, users press `ALT+` the accelerator key to use an accelerator (mnemonic access key). If the currently selected control is not an editable control, users don't need to press the `ALT` key.

On UNIX On UNIX, users press *Meta* and the accelerator key to use an accelerator (mnemonic). *Meta* is a logical key in the X Window System that can be mapped to an appropriate physical key on the keyboard. Some keyboards provide a physical key for *Meta* (such as `◆` on Sun keyboards), but often *Meta* is mapped to another physical key, such as `ALT`.

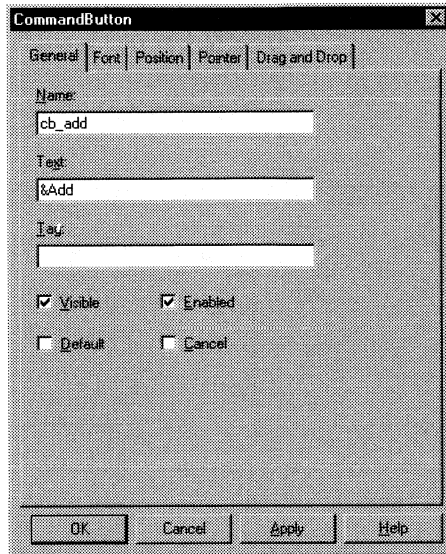
On Macintosh On the Macintosh, controls do not have accelerator keys, but you can define accelerator keys for use on other platforms.

How you define accelerator keys depends on whether the type of control has displayed text associated with it.

❖ **To define an accelerator key for a `CommandButton`, `CheckBox`, or `RadioButton`:**

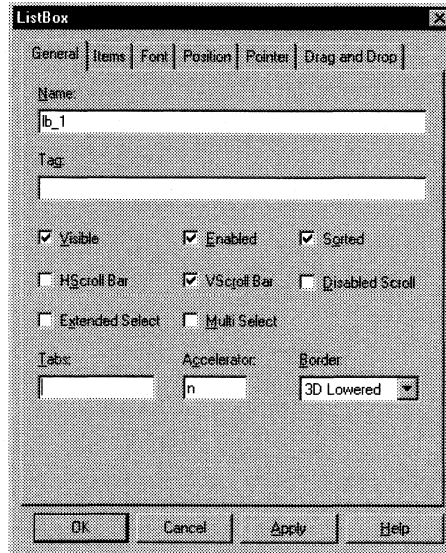
- ◆ When defining the text that displays for the control, precede the accelerator key with an ampersand character (&).

PowerBuilder displays an underline to indicate the accelerator key.



❖ **To define an accelerator key for a SingleLineEdit, MultiLineEdit, ListBox, or DropDownListBox:**

- 1 Double-click the control to display the General property page.
- 2 Type the letter of the accelerator key in the Accelerator box. For example, to make ALT+N the accelerator for the control, type **n** in the Accelerator box.



- 3 Click OK.

At this point you have defined the accelerator key, but the user has no way of knowing it. So next you need to label the control.

- 4 Place a StaticText control next to the control you just assigned the accelerator key.

When defining the text for the StaticText control, precede the accelerator key with an ampersand character (&).

PowerBuilder displays an underline to indicate the accelerator key.

Now your user knows that there is an accelerator key associated with the control.

Specifying accessibility of controls

Controls have two boolean properties that affect accessibility of the control:

- ◆ Visible
- ◆ Enabled

Using the Visible property

If the Visible property is selected, the control displays in the window. If you want a control to be initially invisible, do not select this property in the Window painter.

Hidden controls do not display by default in the Window painter.

❖ **To display hidden controls in the painter:**

- 1 Select Design>Options from the menu bar.

The Options property sheet displays.

- 2 Select Show Invisibles on the General property page.

To display a control during execution, assign the TRUE value to the Visible property, such as:

```
controlname.Visible = TRUE
```

Using the Enabled property

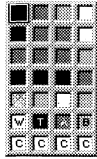
If the Enabled property is selected, the control is active. For example, an enabled CommandButton can be clicked.

If you want a control to display but be inactive, do not select the Enabled property. For example, a CommandButton might be active only after the user has selected an option. In this case, display the CommandButton initially disabled (it appears grayed out) then when the user selects the option, enable the CommandButton in a script:

```
CommandButtonName.Enabled = TRUE
```

Choosing colors

The Window painter has two Color dropdown toolbars, which display colors that you can use for the background and foreground of components of the window.



Initially, the dropdown toolbars display these color selections:

- ◆ 20 predefined colors
- ◆ Window color (labeled W)
- ◆ Window Text color (labeled T)
- ◆ Application Workspace color (labeled A)
- ◆ Button color (labeled B)
- ◆ Four custom colors (labeled C)

Some of these selections derive from each user's system settings. For example, on Windows systems, the Window, Window Text, Application Workspace, and Button colors are those defined by the user in the Windows Control Panel. So if you use these colors in your window, the window colors will change to match the user's settings during execution.

Selecting colors

There are two ways to assign colors to controls.

Colors cannot be assigned to some controls

You cannot change the colors for CommandButtons, PictureButtons, Pictures, Scrollbars, and OLE controls.

❖ **To assign a color using the PainterBar:**

- 1 Place the mouse pointer over the control and display the popup menu.
- 2 Select either the foreground or background color button from the PainterBar.
- 3 Select a color from the cascading menu.

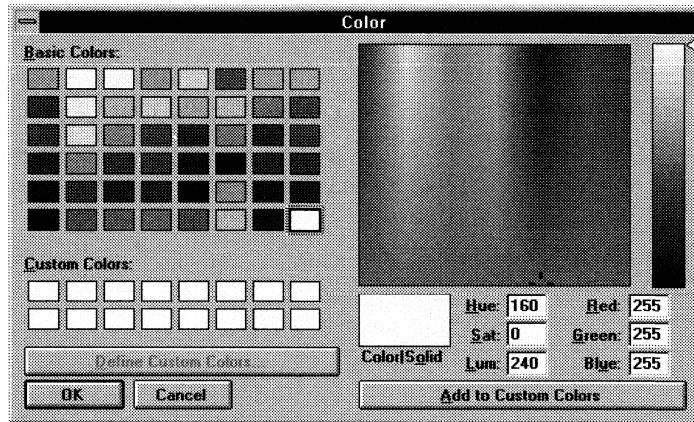
Defining your own colors

You can define your own custom colors for use in windows, user objects, and DataWindow objects.

❖ **To maintain your custom colors:**

- 1 Select Design>Custom Colors from the menu bar.

The Color dialog box displays.



- 2 Choose an existing color or create the color you want. You can start with one of the basic colors and customize it in the palette to the right by dragging the color indicator with the mouse. You can also specify precise values to define the color.
- 3 When you have the color you want, click Add to Custom Colors.
The new color displays in the list of custom colors.
- 4 Select the new color in the list of custom colors.
- 5 Click OK.

The new color displays in the Color dropdown toolbar and is available in all windows, user objects, and DataWindow objects you create.

PowerBuilder saves custom colors in the [Colors] section of the PowerBuilder initialization file, so they are available across sessions.

Using the 3D look

Applications commonly have a three-dimensional look and feel. By selecting a 3D border for your SingleLineEdit boxes and other controls, and by making windows gray, your applications can too.

❖ **To have the 3D look used by default:**

- 1 Select Design>Options from the menu bar.

The Options property sheet displays.

- 2 Select Default to 3D on the General property page.

When you build a new window, PowerBuilder automatically sets the window background color to gray and uses 3D borders when you place controls.

PowerBuilder records this preference in the Default3D variable in the [Window] section of the PowerBuilder initialization file, so the preference is maintained across sessions.

Using the individual controls

There are four basic types of controls with different purposes:

Function	Controls include
Invoke actions	CommandButtons, PictureButtons
Display data	ListBoxes, PictureListBoxes, DropDownListBoxes, DropDownPictureListBoxes, DataWindow controls, StaticText, ListViews, TreeViews, RichTextEdit, Graphs, Pictures, SingleLineEdits, MultiLineEdits, EditMasks, OLE controls
Indicate choices	RadioButtons, CheckBoxes (you can group these controls in a GroupBox)
Decorative	Line, Rectangle, RoundedRectangle, Oval

How to use the controls

You should use the controls only for the purpose shown in the table. For example, users expect radio buttons to only select an option. Don't use a radio button to also invoke an action, such as opening a window or printing. Use a command button for that.

There are several exceptions: 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 following sections describe some features that are unique to the individual controls.

Not covered here

This chapter does not cover the following:

DataWindow controls and objects—see Chapter 14, "Defining DataWindow Objects".

RichTextEdit controls—see Chapter 22, "Working with Rich Text".

User objects—see Chapter 11, "Working with User Objects".

Graph controls—see Chapter 20, "Working with Graphs".

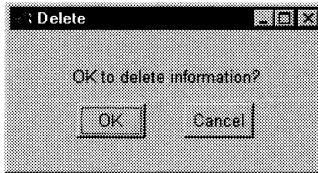
OLE 2.0 controls—see Chapter 23, "Using OLE in a DataWindow Object".

For more information

To learn more about the proper use of controls, consult a style guide that presents guidelines for your target environment. For a list of events, properties, and functions for each control, see *Objects and Controls*.

Using CommandButtons

CommandButtons are used to carry out an action. For example, you can use an OK button to confirm a deletion or a Cancel button to cancel a requested deletion.



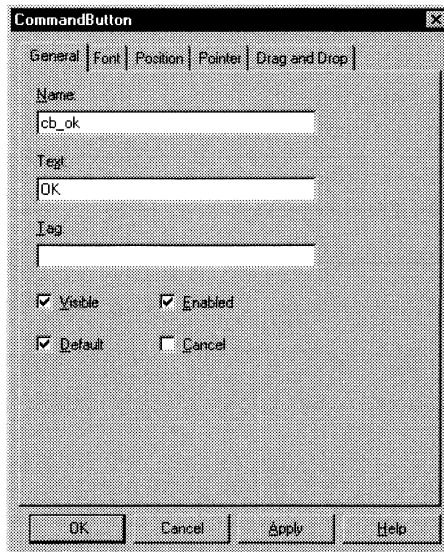
Place related CommandButtons along the bottom of the window if there are not too many. If there are a lot, place them along the right side of the window.

You cannot change the color or alignment of text in a CommandButton.

If clicking the button opens a window that requires user interaction before any other action takes place, use ellipsis points in the button text (for example, Print...).

Specifying Default and Cancel buttons

You can specify that a CommandButton is the default button in a window: select Default on the General property page of the button's property sheet.



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 `SetFocus()` or tab order to make sure the default button behaves as documented above.

FOR INFO For information on `SetFocus()`, see the *PowerScript Reference*. For how to define the tab order, see "Defining the tab order" on page 211.

A bold border is placed around the default `CommandButton` (or the button with focus if the user explicitly tabs to a `CommandButton`).

Similarly, you can define a `CommandButton` as being the cancel button by selecting `Cancel` on the `General` property page of the button's property sheet. If you define a cancel `CommandButton`, when the user presses the `ESC` key the cancel button's `Clicked` event is triggered.

Using `PictureButtons`

`PictureButtons` are *PowerBuilder*-specific controls that are identical to `CommandButtons` in their functionality. The only difference is that you can specify a bitmap (BMP) file to display on the button. Depending on your platform, you may also be able to specify a run-length encoded (RLE) file, an Aldus-style Windows metafile (WMF), or a Macintosh PICT file.

FOR INFO For which formats are supported on each platform, see the matrix of feature support in the chapter on building an application for multiple platforms in *Application Techniques*.

You can choose to display one picture if the button is enabled and a different picture if the button is disabled.

Use these controls when you want to be able to represent the purpose of a button using a picture instead of just text.

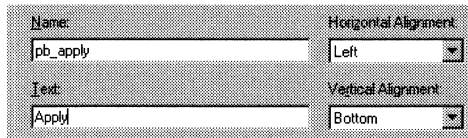
❖ **To specify a picture:**

- 1 Place a PictureBox control in the window.
- 2 Choose Properties from the popup menu and select the General tab.
The General property page displays.
- 3 Enter the name of the image file you want to display when the button is enabled (or use the Browse Enabled button and choose a file).
- 4 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).
- 5 Click OK.

If the PictureBox is defined as initially enabled, the enabled picture displays in the Window painter. If the PictureBox is defined as initially disabled, the disabled picture displays in the painter.

❖ **To specify button text alignment:**

- 1 Choose Properties from the popup menu and select the General tab.
The General property page displays.
- 2 Enter the text for the PictureBox in the Text textbox.



- 3 Use the Horizontal Alignment and Vertical Alignment listboxes to choose how you want to display the button text.

Using RadioButtons

RadioButtons are round buttons that represent mutually exclusive options. They always exist in groups. Exactly one RadioButton is selected in each group.

When a RadioButton is selected, it has a dark center; when it is not selected, the center is blank.

In the following example, the text can be either plain, bold, or italic (plain is selected):



When the user clicks a `RadioButton`, it becomes selected and the previously selected `RadioButton` in the group becomes deselected.

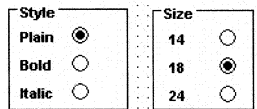
Only use `RadioButtons` to represent the state of an option. Do not use them to invoke actions.

When a window opens, one `RadioButton` in a group must be selected. You specify which is the initially selected `RadioButton` by selecting `Checked` from the `RadioButton`'s property sheet `General` property page.

Grouping RadioButtons

By default, all `RadioButtons` in a window are in one group, no matter what their location in the window. Only one `RadioButton` can be selected at a time.

You use a `GroupBox` control to group related `RadioButtons`. All `RadioButtons` inside a `GroupBox` are considered to be in one group. One button can be selected in each group.



The Automatic property

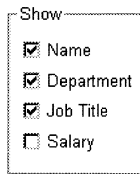
When a window contains several `RadioButtons` that are outside of a `GroupBox`, the window acts as a `GroupBox`. Only one `RadioButton` can be active at a time, *unless* the checkbox for the `Automatic` property on the `RadioButton`'s `General` property page is cleared.

When the `Automatic` property is not set, you must use scripts to control when a button is selected. Multiple `RadioButtons` can be selected outside of a group.

The `Automatic` property does not change how `RadioButtons` are processed inside a `GroupBox`.

Using CheckBoxes

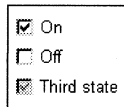
CheckBoxes are square boxes used to set independent options. When they are selected, they contain a checkmark; when they are not selected, they are empty.



CheckBoxes are independent of each other. You can group them in a **GroupBox** or rectangle to make the window easier to understand and use, but that does not affect the CheckBoxes' behavior; they are still independent.

Using three states

CheckBoxes usually have two states: on and off. But sometimes you want to represent a third state, such as **Unknown**. The third state displays as a grayed box with a checkmark.



❖ To enable the third state:

- ◆ Choose **Three State** from the **CheckBox's General** property page.

To specify that the button's current state is the third state, select **Third State** from the **CheckBox's** property sheet **General** property page.

Using StaticText

You use a **StaticText** control to display text to the user or to describe a control that doesn't have text associated with it, such as a listbox or edit 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 don't expect to interact with static text.

Indicating accelerator keys

One use of a `StaticText` control is to label a listbox or edit control. If you assign an accelerator key to a listbox 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 214.

Using `SingleLineEdits` and `MultiLineEdits`

A `SingleLineEdit` is a box in which users can enter a single line of text. A `MultiLineEdit` is a box in which users can enter more than one line of text.

`SingleLineEdits` and `MultiLineEdits` are typically used for input and output of data.

For these controls, you can specify many 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—whether to display entry using asterisks instead of the actual entry (`Password`)
- ◆ In which case to accept and display entry (`TextCase`)
- ◆ Whether the selection displays when the control does not have focus (`Hide Selection`)

FOR INFO For more information about properties of these controls, click the Help button in the control's property sheet.

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.

Edit mask character for Arabic and Hebrew

The b mask character allows the entry of Arabic characters when you run PowerBuilder on an Arabic-enabled version of Windows and Hebrew characters when running on a Hebrew-enabled version. It has no effect on other operating systems.

❖ To use an EditMask control:

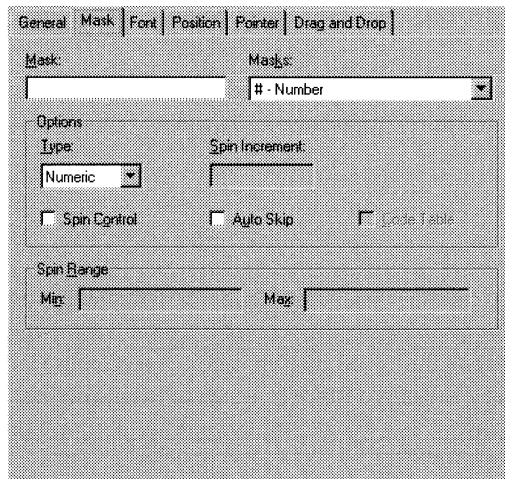
1 Click the EditMask button in the PainterBar and click where you want the control to be.

2 Double-click the control or select Properties from its popup menu.

The property sheet displays.

3 Name the control on the General property page.

4 Select the Mask tab.



5 In the Type dropdown listbox, specify the type of data that users will be entering into the control.

- 6 Specify the mask. The special characters used in the mask for the specified data type display in the Masks box.

About masks

Masks in EditMask controls in windows work the same as masks in display formats and in the EditMask edit style in DataWindow objects.

FOR INFO For more information about specifying masks, see the discussion of display formats in Chapter 16, "Displaying and Validating Data".

- 7 Specify other properties for the EditMask control and click OK.

FOR INFO For information on the other properties, click the Help button.

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 may not be able to enter text.

To correct this, either specify a smaller font size or resize the EditMask control.

Keyboard behavior

Some keystrokes have special behavior in EditMask controls. For more information, see "The EditMask edit style" on page 573.

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. The control would look like this:

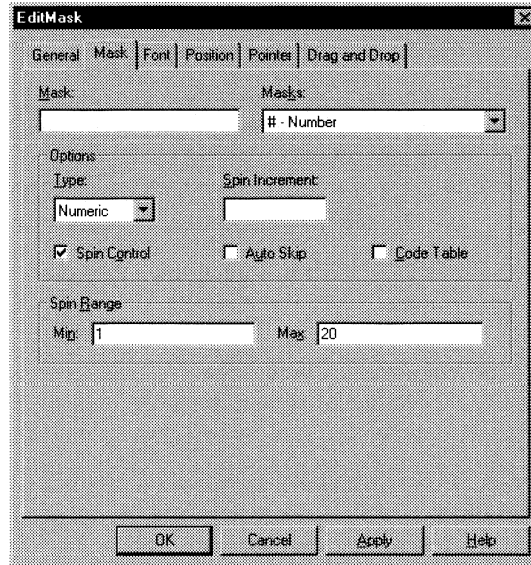


❖ To define an EditMask as a spin control:

- 1 Name the EditMask and provide the data type and mask, as described above.

- 2 Select the Spin Control checkbox in the Options group on the Mask property page.

Options for spin controls display.



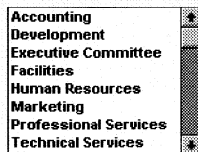
- 3 Specify the needed information.

For example, to allow users to select a number from 1 to 20 in increments of 1, specify a spin range of 1 to 20 and a spin increment of 1.

FOR INFO For more information on the options for spin controls, click the Help button in the EditMask Style dialog box.

Using ListBoxes

A ListBox displays available choices. You can specify that ListBoxes have scrollbars if more choices exist than can be displayed in the ListBox at one time.



ListBoxes are an exception to the rule that a control should either invoke an action or be used for viewing and entering data. ListBoxes can do both. ListBoxes display data, but can also invoke actions. Typically in Windows applications, clicking an item in the ListBox selects the item. Double-clicking an item acts upon the item.

For example, in the PowerBuilder File Open box, clicking a filename in a ListBox selects the file. Double-clicking a name opens the file.

PowerBuilder automatically selects (highlights) an item when a user selects it during execution. If you want something to happen when users *double-click* an item, you must code a script for the control's DoubleClicked event (note that the Clicked event is always triggered before the DoubleClicked event).

Populating the list

To add items to a ListBox, choose Properties from the control's popup menu, select the Items tab, and enter the values for the list. Press TAB to go to the next line.

Changing the list during execution

To change the items in the list during execution, use the functions AddItem, DeleteItem, and InsertItem.

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

Setting tab stops

You can set tab stops for text in ListBoxes (and in MultiLineEdits) by setting the TabStop property on the General property page. You can define up to 16 tab stops (the default is a tab stop every eight characters).

You can also define tab stops in a script. Here is an example that defines two tab stops and populates a ListBox:

```
// lb_1 is the name of the ListBox.
string f1, f2, f3
f1 = "1"
f2 = "Emily"
f3 = "Foulkes"
// Define 1st tab stop at character 5.
lb_1.tabstop[1] = 5
// Define 2nd tab stop 10 characters after the 1st.
lb_1.tabstop[2] = 10
// Add an item, separated by tabs.
// Note that the ~t must have a space on either side
// and must be lowercase.
lb_1.AddItem(f1 + " ~t " + f2 + " ~t " + f3)
```

Note that this script will not work if it is in the window's Open event (the controls have not yet been created). The best way to specify this is in a user event that is posted in the window's Open event using the PostEvent function.

Other properties

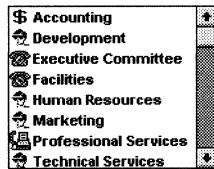
For ListBoxes, you can specify whether:

- ◆ Items in the ListBox are displayed in sorted order
- ◆ The ListBox allows the user to select multiple items
- ◆ The ListBox displays scrollbars if needed

FOR INFO For more information, click the Help button in the ListBox Property sheet.

Using PictureListBoxes

A PictureListBox, like a ListBox, displays available choices in both text and images. You can specify that PictureListBoxes have scrollbars if more choices exist than can be displayed in the PictureListBox at one time.



Adding images to a PictureListBox

You can choose from a group of stock images provided by PowerBuilder, or use any bitmap, cursor, or icon files when you add images to a PictureListBox.

Keep in mind, however, that the images should add meaning to the list of choices. If you use a large number of images in a list, they become meaningless.

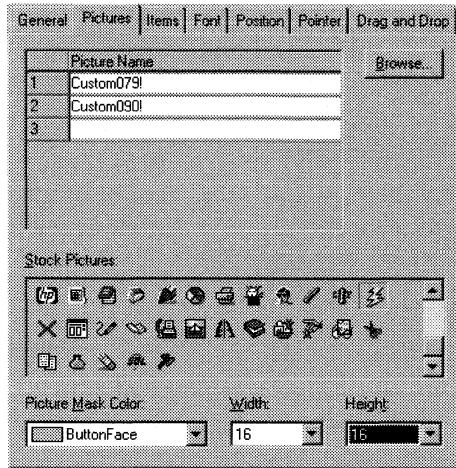
You could, for example, use images in a long list of employees to show the department to which each employee belongs. So you may have a list with 20 or 30 employees, associated with one of five images.

❖ To add an image to a PictureListBox:

- 1 Choose Properties from the control's popup menu and select the Pictures tab.

The Pictures property page displays.

- 2 Choose a picture from the Stock Pictures list.
or
Use the Browse button to select a bitmap, cursor, or icon file to include in the PictureBox.
- 3 Specify a picture mask color (the color that will be transparent for the picture).
- 4 Specify the height and width for the image in pixels.

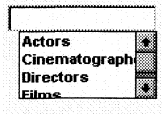


- 5 Click OK.

FOR INFO For information about other properties, click the Help button in the PictureBox property sheet.

Using DropDownListBoxes

DropDownListBoxes combine the features of a SingleLineEdit and a ListBox.



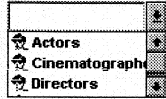
There are two types of DropDownListBoxes:

- ◆ Noneditable
- ◆ Editable

Noneditable boxes	<p>If you want your user to choose only from a fixed set of choices, make the <code>DropDownListBox</code> noneditable.</p> <p>In these boxes, the only valid values are those in the list.</p> <p>There are several ways for users to pick an item from a noneditable <code>DropDownListBox</code>:</p>
Editable boxes	<ul style="list-style-type: none">◆ Use the arrow keys to scroll through the list.◆ Type a character. The listbox scrolls to the first entry in the list that begins with the typed character. Typing the character again scrolls to the next entry beginning with the character.◆ Click the down arrow to the right of the edit control to display the list, then select the one you want. <p>If you want to give users the option of specifying a value that is not in the list, make the <code>DropDownListBox</code> editable by selecting the <code>Allow Editing</code> checkbox on the <code>General</code> property page.</p> <p>With editable <code>DropDownListBoxes</code>, you can choose to have the list always display or not. For the latter type, the user can display the list by clicking the down arrow.</p>
Populating the list	<p>You specify the list in a <code>DropDownListBox</code> the same way as for a <code>ListBox</code>, described above.</p>
Specifying the size of the dropdown box	<p>To indicate the size of the box that drops down, size the control in the <code>Window painter</code> using the mouse. When the control is selected in the painter, the full size—including the dropdown box—is shown.</p>
Other properties	<p>As with <code>ListBoxes</code>, you can specify whether the list is sorted and whether the edit control is scrollable.</p> <p>FOR INFO For more information, click the <code>Help</code> button in the <code>DropDownListBox</code> property sheet.</p>

Using DropDownPictureListBoxes

DropDownPictureListBoxes are similar to DropDownListBoxes in the way they present information. Where they differ is that while DropDownListBoxes only use text to present information, DropDownPictureListBoxes add images to the information.



Everything that you can do with DropDownListBoxes you can do with a DropDownPictureListBox.

Adding images to a DropDownPictureListBox

You can choose from a group of stock images provided by PowerBuilder, or use any bitmap, cursor, or icon files when you add images to a DropDownPictureListBox. You use the same technique that you use to add pictures to a PictureListBox.

FOR INFO For information, see "Adding images to a PictureListBox" on page 232.

Using pictures

Pictures are PowerBuilder-specific controls that display a bitmap (BMP) file. Depending on your platform, you may also be able to specify a run-length encoded (RLE) file, an Aldus-style Windows metafile (WMF), or a Macintosh PICT file.

FOR INFO For which formats are supported on each platform, see the matrix of feature support in the chapter on building an application for multiple platforms in *Application Techniques*.

❖ To display a picture:

- 1 Place a picture control in the window.
- 2 Select Properties from the control's popup menu and select the General tab.
The General property page displays.
- 3 Enter the name of the file you want to display in the File Name textbox or use the Browse button to select a file.

4 Click OK.

The picture displays.

You can choose to resize or invert the image.

Be careful how you use picture controls. They can be used for almost any purpose. They have events, so users can click on them. Or they can be used simply to display an image. Be consistent in their use so users know what they can do with them.

Using drawing objects

PowerBuilder provides the following drawing objects: Line, Oval, Rectangle, and RoundRectangle.

These objects have no events associated with them. They are for display purposes only. Use them to make your window more attractive or to group controls.

You can use the following functions to manipulate drawing objects during execution:

- Hide
- Move
- Resize
- Show

In addition, each drawing object has a set of 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

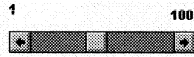
The Bring to Front and Send to Back items on the control's popup menu have no effect on drawing objects. They cannot be placed on top of another control, such as a GroupBox.

Using HScrollBars and VScrollBars

You can place freestanding scrollbar controls within a window. Typically, you use these controls to do one of the following:

- ◆ Act as a slider control for users to be able to specify a continuous value
- ◆ Graphically display information to the user

For example:



You can set the position of the scroll box by specifying the value for the control's Position property. When the user drags the scroll box, the value of Position is automatically updated.

Using Tab controls

A Tab control is a container for tab pages that display other controls. You can include a Tab control in your application to present information that can logically be grouped together but may also be divided into distinct categories. An obvious implementation of the Tab control is the use of property sheets for all objects in PowerBuilder.

Creating a Tab control

- ❖ **To add a Tab control to a window:**
 - ◆ In the Window painter, choose the Tab control tool and click in the window.

PowerBuilder creates a Tab control with one tab page.

Adding tab pages to a Tab control

- ❖ **To create a new tab page within the Tab control:**
 - 1 Right-click in the display area of the Tab control.
 - 2 Choose Insert TabPage from the popup menu.
 - 3 Add controls to the new property page.
- ❖ **To define a tab page independent of a Tab control:**
 - 1 In the User Object painter, create a custom visual user object.

- 2 Size the user object to match the size of the Tab controls you will use it in.
 - 3 Add the controls that will appear on the tab page to the user object.
 - 4 On the user object's property sheet, display the TabPage page and fill in information to be used by the tab page. You can set:
 - ◆ Tab Text—The text to be displayed on the tab
 - ◆ Tab Picture—A picture to appear on the tab with or instead of the text
 - ◆ PowerTipText—Text for a popup message that displays when the user moves the cursor to the tab
 - ◆ Colors for the tab and the text on the tab
- ❖ **To add a tab page that exists as an independent user object to a Tab control:**
- 1 Right-click in the display area of the Tab control.
 - 2 Choose Insert User Object from the popup menu.
 - 3 Select a user object.

The tab page is inherited from the user object you select. You can set tab page properties and write scripts for the inherited user object just as you do for tab pages defined within the Tab control.

You can't edit the content of the user object within the Tab control. If you want to edit the controls, go back to the User Object painter to make changes.

Manipulating the Tab control

- ❖ **To change the name and properties of the Tab control:**
- 1 Select the Tab control and press ENTER.
or
Double-click in the display area or anywhere in the tab area that isn't a tab.
or
With the mouse anywhere in the control except on a tab, display the popup menu and select Properties.
 - 2 Edit the properties.

FOR INFO For more information, click the Help button in the Tab property sheet.

❖ To change the scripts of the Tab control:

- 1 Select the Tab control and press CTRL+S (COMMAND+S on the Macintosh).

or

With the mouse anywhere in the control except on a tab, display the popup menu and select Script.

- 2 Select a script and edit it.

❖ To resize a Tab control:

- ◆ Grab a border of the control and drag it to the new size.

The Tab control and all tab pages are sized as a group.

❖ To move a Tab control:

- ◆ Drag anywhere except on the tabs to move the control to the new position.

The Tab control and all tab pages are moved as a group.

Manipulating the tab pages

❖ To view a different tab page:

- ◆ Click on the page's tab.

The tab page is brought to the front. The tabs are rearranged according to the Tab position setting you have chosen.

❖ To change the name and properties of a tab page:

- 1 Select the tab.

It may move to the position for a selected tab based on the Tab Position setting.

- 2 Double-click on the tab—in its new position, if necessary.

or

Select Properties from the tab's popup menu.

- 3 Edit the properties.

❖ To change the scripts of the tab page:

- 1 Select the tab.

It may move to the position for a selected tab based on the Tab Position setting.

- 2 Select Script from the tab's popup menu.

3 Select a script and edit it.

❖ **To delete a tab page from a Tab control:**

- ◆ With the mouse on the tab corresponding to the tab page to be deleted display the popup menu and choose Cut or Clear.

Managing controls on
tab pages

❖ **To add a control to a tab page:**

- ◆ Choose a control from the toolbar or the Control menu and click on the property page, just as you would add a control to a window.

You can only add controls to a tab page created within the Tab control. To add controls to an independent property 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.

❖ **To move a control between a tab page and the window containing the Tab control:**

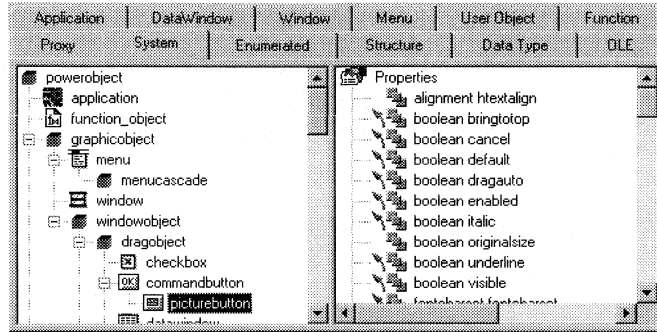
- ◆ Cut or copy the control and paste it on the destination window or tab page.

Moving the control between a tab page and the window changes the control's parent, which affects scripts that refer to the control.

FOR INFO For more information on the Tab control, see the chapter on using tabs in a window in *Application Techniques*.

Using TreeView controls

You can use TreeView controls in your application to represent relationships between hierarchical data. One example of a TreeView implementation is the Browser.



Creating a TreeView control

❖ To add a TreeView control to a window:

- ◆ In the Window painter, choose the TreeView control tool and click in the 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 that you add images to a PictureListBox.

FOR INFO For more information, see "Adding images to a PictureListBox" on page 232.

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 INFO For more information about PictureHeight and PictureWidth, see the *PowerScript Reference*.

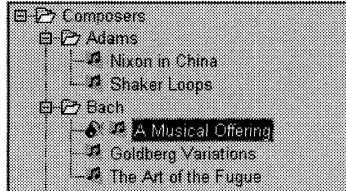
❖ To add items to a TreeView:

- ◆ Write a script in the TreeView constructor event to create TreeView items.

FOR INFO For more information about populating a TreeView, see the *PowerScript Reference*.

Adding state pictures to TreeView items

A *state* picture is an image that appears to the left of the TreeView item indicating that the item is not in its normal mode. A state picture could indicate that a TreeView item is being changed, or that it is performing a process and is unavailable for transaction.



❖ To specify a state picture for a TreeView item:

- 1 Choose Properties from the control's popup menu and select the State tab.
- 2 Use the stock pictures list to add images to the TreeView.
or
Use the Browse button to select a bitmap, icon, or cursor file.

❖ 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 INFO For more information on the TreeView control, see *Application Techniques*.

Setting other properties

❖ To specify other TreeView properties:

- 1 Choose Properties from the control's popup menu and select the General tab.

- 2 Enter a name for the TreeView in the Name textbox and specify other properties as appropriate. Among the properties you can specify on the General property page are:
 - ◆ The border style
 - ◆ Whether the TreeView has lines showing the item hierarchy
 - ◆ Whether the TreeView includes collapse and expand buttons
 - ◆ Whether the user can delete items
 - ◆ Whether the user can drag and drop items into the TreeView

FOR INFO For more information, click the Help button in the TreeView property sheet.

- 3 For other options, choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
The images used to represent TreeView items	Pictures
The state images for the TreeView items	State
The font size, family, and color for TreeView items	Font
The size and position of the TreeView	Position
The icon for the mouse pointer in the TreeView	Pointer
The icon for a drag item, and whether the drag-and-drop must be performed programmatically	Drag and Drop

FOR INFO For more information on the TreeView control, see *Application Techniques*.

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 freeform lists, and you can associate additional columns with items in the list. The following illustration from the Code Examples application shows a ListView control used in a sales order application.



Creating a ListView control

❖ **To add a ListView control to a window:**

- ◆ In the Window painter, choose the ListView control tool and click in the window.

Adding ListView Items and pictures

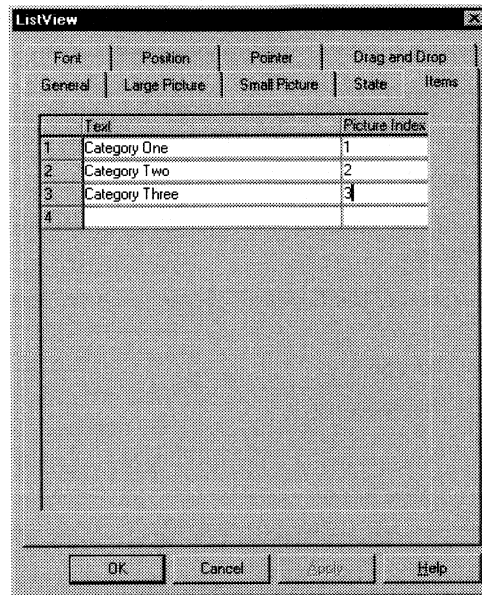
Adding images to a ListView control is the same as adding images to a PictureBox. The ListView property sheet has two tab pages for adding pictures: Large Picture (default size 32 by 32 pixels) and Small Picture (16 by 16 pixels).

FOR INFO For more information, see "Adding images to a PictureBox" on page 232.

❖ **To add ListView items:**

- 1 Choose Properties from the control's popup menu and select the Items tab.

The Items property page displays.



- 2 Enter the name of the ListView item and the picture index you want to associate with it. This picture index corresponds to the images you select on the Large Picture, Small Picture, and State property pages.
- 3 Repeat until all the items are added to the ListView.

Choosing a ListView style

You can display a ListView in four styles:

- ◆ Large icon
- ◆ Small icon
- ◆ List
- ◆ Report

❖ **To select a ListView style:**

- 1 Choose Properties from the control's popup menu and select the General tab.
- 2 Select the type of view you want from the View dropdown list.

FOR INFO For more information about other properties, click the Help button in the ListView property sheet.

Setting other properties

❖ **To specify ListView properties:**

- 1 Select Properties on the popup menu option to display the property sheet.
- 2 Choose the tab appropriate to the property you want to specify:

To specify	Choose this tab
The border style	General
Whether the TreeView will have 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	
The images for ListView items in LargeIcon view	Large Picture
The images for ListView items in Small Icon, list, and report views	Small Picture
The state images for ListView items	State
The names and associated picture index for ListView items	Items
The font size, family, and color for ListView items	Font
The size and position of the ListView	Position
The icon for the mouse pointer in the ListView	Pointer
The icon for a drag item, and whether the drag-and-drop must be performed programmatically	Drag and Drop

FOR INFO For more information on the ListView control, see *Application Techniques*.

Understanding Inheritance

About this chapter

This chapter describes how to use inheritance to build PowerBuilder objects.

Contents

Topic	Page
Overview of inheritance	248
The inheritance hierarchy	249
Working with inherited objects	251
Using inherited scripts	253

Overview of inheritance

One of the most powerful features of PowerBuilder is **inheritance**. It enables you to build windows, user objects, and menus that are derived from existing objects.

Using inheritance has a number of advantages:

- ◆ When you change an ancestor object, the changes are reflected in all the descendants. You do not have to manually make changes in the descendants as you would in a copy. This saves you coding time and makes the application easier to maintain.
- ◆ The descendant inherits the ancestor's scripts so you do not have to re-enter the code to add to the script.
- ◆ You get consistency in the code and objects in your applications.

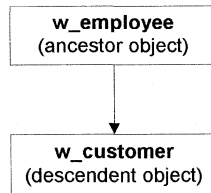
This chapter describes how inheritance works in PowerBuilder and how to use it to maximize your productivity.

The inheritance hierarchy

When you build an object that inherits from another object, you are creating a hierarchy (or tree structure) of ancestor objects and descendent objects. Chapter 7, "Defining Windows", uses the example of creating a window `w_customer` that inherits its properties from the existing window `w_employee`. In this example, `w_employee` is the ancestor and `w_customer` is the descendant.

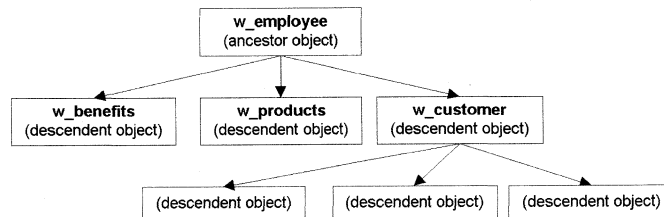
The object at the top of the hierarchy is a base class object, and the other objects are descendants of this object. Each descendant inherits information from its ancestor. The base class object typically performs generalized processing, and each descendant modifies the inherited processing as needed.

In the example, the `w_employee` window is the base class window. The hierarchy looks like this:



Multiple descendants

An object can have an unlimited number of descendants and each descendant can also be an ancestor. For example, if you build several windows that are direct descendants of the `w_employee` window and three windows that are direct descendants of the `w_customer` window, the hierarchy looks like this:



Viewing the hierarchy

PowerBuilder provides a Browser that shows the hierarchy of windows, menus, and user objects.

❖ **To view the hierarchy:**

- 1 Click the Browser button.
- 2 Choose the System tab to show the built-in PowerBuilder objects.
- 3 Display the popup menu in the object panel of the Browser and choose Show Hierarchy.

❖ **To display the class hierarchy for other object types:**

- 1 Choose a tab for another object type (for example, the Menu tab).
- 2 Display the popup menu in the object panel of the Browser and choose Show Hierarchy.

If there is no inheritance for the object type, Show Hierarchy will be grayed out.

- 3 Display the popup menu in the object panel of the Browser and choose Expand All.

PowerBuilder shows the selected object type in the current application; descendent objects are shown indented under their ancestors.

Getting context-sensitive Help in the Browser

To get context-sensitive Help for an object, control, or function, select Help from its popup menu.

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 their 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 MenuObjects to a menu

FOR INFO For specifics about what you can do in inherited windows, user objects, and menus, see Chapter 7, "Defining Windows", Chapter 11, "Working with User Objects", and Chapter 10, "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 be careful when you change the definition of an ancestor object. You may want to regenerate descendent objects if you do any of the following:
 - ◆ Delete or change the name of an instance variable in the ancestor
 - ◆ Modify a user-defined function in the ancestor
 - ◆ Delete a user event in an ancestor
 - ◆ Rename or delete a control in an ancestor

When you regenerate the descendants, the compiler will flag any references it cannot resolve so you can fix them.

FOR INFO For information about regenerating objects, see Chapter 3, "Managing Libraries".

About local changes

If you change a property in an ancestor object, the property will also change in all descendants—if you haven't already changed that property in a descendant, in which case the property in the descendant stays the same. In other words, local changes always override inherited properties.

Resetting a
descendent object's
properties

When you use inheritance to build a window, menu, or user object, you can change the values of inherited properties (such as the placement of controls in a window). If you change your mind, you can easily reset the properties to the values they had in the ancestor.

- ❖ **To reset the properties to their values in the ancestor:**
 - ◆ Select Edit>Reset Properties from the menu bar in the Window, Menu, or User Object painter.
The properties are reset to the values they had in the ancestor.

Using inherited scripts

In the hierarchy formed by ancestors and descendants, each descendant inherits its scripts from its immediate ancestor. If the inherited event does not have a script, you can write a script for the event for the descendant. If the inherited event does have a script, the ancestor script will execute in the descendant by default. You can:

- ◆ Override the ancestor script—execute a descendent script *instead* of the ancestor script
- ◆ Extend the ancestor script—build a script that executes *after* the ancestor script

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 INFO For more information, see "Calling an ancestor script" on page 256.

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 INFO For more information, see *Application Techniques*.

Viewing inherited scripts

When you open the PowerScript painter for an inherited object or control that has a script defined only in an ancestor, the workspace is blank: you do not see the ancestor script.

The PowerScript painter indicates which events have scripts written for an ancestor as follows:

- ◆ If the event has a script in an ancestor only, the script icon in the Select Event dropdown listbox is displayed in color.
- ◆ If the event has a script in an ancestor as well as in the object you are working with, the icon is displayed half in color.

❖ **To view the ancestor script:**

- 1 Select the object or control whose ancestor script you want to view.

To view an inherited script for the window, make sure no control is selected.

- 2 Open the PowerScript painter.
- 3 Select the event whose script you want to see.

The workspace is blank; the script for the ancestor does not display.

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

PowerBuilder displays in a dialog box the script defined in the closest parent (the object immediately above the current object in the hierarchy).



- 5 To climb the inheritance hierarchy, click the Ancestor button.

PowerBuilder shows the script for the grandparent of the current object. By clicking Ancestor, you can traverse the entire inheritance hierarchy.

- 6 To copy the script to the clipboard, select the part you want, then click the Copy button. You can then paste the contents into another script.

Overriding a script

❖ **To override an ancestor script:**

- 1 Select the object or control whose ancestor script you want to override.

To override a window-level script, make sure no control is selected.

- 2 Open the PowerScript painter.
- 3 Select the event for which you want to override the script.

- 4 Select Design>Override Ancestor Script from the menu bar.
- 5 Code a script for the event in the descendant.

You can call the script for any event in any ancestor as well as call any user-defined functions that have been defined for the ancestor.

FOR INFO For information about calling an ancestor script or function, see "Calling an ancestor script" on page 256 and "Calling an ancestor function" on page 256.

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 workspace.

During execution, PowerBuilder executes the descendent script when the event is triggered. The ancestor script is not executed.

Example of
overriding a script

If the script for the Open event in the ancestor window displays employee files and you want to display customer files in the descendant window, select Override Ancestor Script and create a new script for the Open event in the descendant to display customer files.

Extending a script

When you extend an ancestor script for an event, PowerBuilder executes the ancestor script, *then* executes the script for the descendant when the event is triggered.

❖ **To extend an ancestor script:**

- 1 Select the object or control whose ancestor script you want to extend.
To extend a script for a window, make sure no control is selected.
- 2 Open the PowerScript painter.
The PowerScript painter displays and the workspace is blank; the script for the ancestor does not display.
- 3 Select the event for which you want to extend the script.
- 4 Make sure Extend Ancestor Script is selected on the Design menu (it is the default).

- 5 Enter the appropriate statements in the workspace.

You can call the script for any event in any ancestor as well as call any user-defined functions that have been defined for the ancestor.

FOR INFO For information about calling an ancestor script or function, see "Calling an ancestor script" next and "Calling an ancestor function" on page 256.

Example of extending a script

If the ancestor window script for the Clicked event in a button beeps when the user clicks the button without selecting an item in a listbox, you might extend the script in the descendant to display a message box in addition to beeping.

Calling an ancestor script

When you write a script for a descendent object or control, you can call scripts written for any ancestor. You can use *ancestorwindow*, *ancestoruserobject*, and *ancestormenu* in the syntax to represent any ancestor of the descendent object; it is not restricted to the immediate ancestor (parent).

FOR INFO For more about calling scripts for an event in an ancestor window, user object, or Menu object, see the *PowerScript Reference*.

To reference the immediate ancestor (parent), you can use the Super reserved word.

FOR INFO For more about Super, see the *PowerScript Reference*.

Calling an ancestor function

When you write a script for a descendent window, user object, or menu, you can call user-defined functions that have been defined for any of its ancestors.

To call the first function up the inheritance hierarchy, just call the function as usual:

function (arguments)

If there are several versions of the function up the inheritance hierarchy and you don't want to call the first one up, you need to specify the name of the object defining the function you want:

ancestorobject::function (arguments)

Limitation

The second syntax, in which you directly specify the ancestor object, works only in scripts for the descendent object itself, not in scripts for *controls or user objects* in the descendent object or in MenuObject scripts.

To call a specific version of an ancestor user-defined function in a script for a control, user object, or Menu object 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 INFO For more about calling an ancestor function, see the *PowerScript Reference*.

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

Topic	Page
About menus and Menu objects	260
Building a new menu	262
Viewing your work	278
Writing scripts for Menu objects	280
Using inheritance to build a menu	285
Using menus	291

About menus and Menu objects

All windows in an application, except child and response windows, should 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 defined as a **Menu object** in PowerBuilder. Menu objects display in a menu bar or in dropdown or cascading menus.

Menus you define in PowerBuilder work exactly the same as standard menus in your operating environment.

FOR INFO For more information about menus and Menu objects, see *Getting Started*.

About using menus

You can use menus you build in PowerBuilder in two ways:

- ◆ **In the menu bar of windows** Window menus are associated with a window in the Window painter and display whenever the window is opened.
- ◆ **As popup menus** Popup menus display only when a script executes the `PopupMenu` function.

Both uses are described in this chapter.

About designing menus

PowerBuilder gives you complete freedom in designing menus. But you should follow conventions for your operating environment in order to make it easy to use your applications. For example, you should keep menus simple and consistent. You should group related items in a dropdown menu. You should use cascading menus sparingly and restrict them to one level.

This chapter describes some guidelines you should follow when designing menus.

FOR INFO A full discussion of menu design is beyond the scope of this book. You should acquire a book that specifically addresses design guidelines for graphical applications and apply the rules when you use PowerBuilder to create your menus.

About building menus

When you build a menu, you:

- ◆ Specify the appearance and behavior of the Menu objects by setting their properties.
- ◆ Build scripts that determine how to respond to events in the Menu objects. To support these scripts, you can declare functions, structures, and variables for the menu.

Two ways There are two ways to build a menu. You can:

- ◆ Build a new menu from scratch

FOR INFO The next section describes how to build menus from scratch.

- ◆ Build a menu that inherits its style, functions, structures, variables, and scripts from an existing menu. You use inheritance to create menus that are derived from existing menus, thereby saving you time and coding.

FOR INFO "Using inheritance to build a menu" on page 285 describes how to use inheritance to build a menu.

Building a new menu

This section describes how to build menus from scratch. You will use this technique to create menus that aren't based on existing menus.

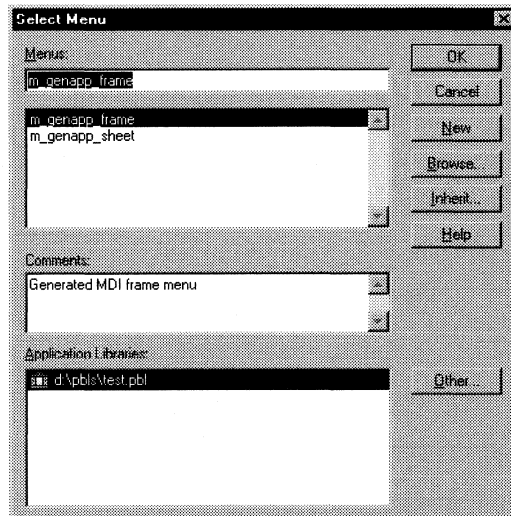
You use the Menu painter to build menus from scratch.

Opening the Menu painter

❖ **To open the Menu painter:**

- 1 Click the Menu painter button in the PowerBar or PowerPanel.

The Select Menu dialog box displays listing the menus in the current library.



- 2 Click the New button to build a new menu.

The Menu painter workspace displays.

About the Menu painter

The Menu painter has several work areas where you specify different parts of a menu.

Menu bar

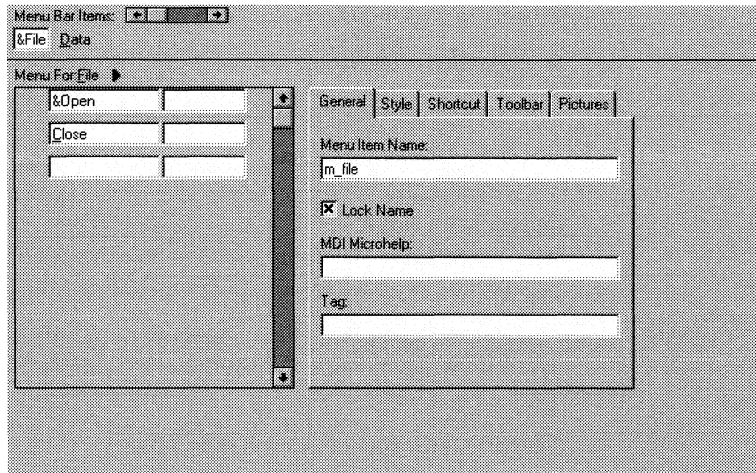
Select an existing Menu object from the menu bar or specify a new one at the top of the workspace. The menu in the following illustration has two objects in the menu bar: File and Data.

Dropdown and cascading menus

Select an existing Menu object from a dropdown or cascading menu or specify a new one in the left pane. The illustration shows two items in the File dropdown menu: Open and Close.

Properties of Menu objects

Specify the appearance and behavior of the selected Menu object in the right pane. You can specify properties for items in the menu bar and items in menus. The illustration shows the General tab page for the File menu.



Working in the Menu painter

You can specify the following in the Menu painter:

- ◆ The Menu objects that display in the menu bar
- ◆ The Menu objects that display under each item in the menu bar
- ◆ Properties of the Menu objects (how they display)
- ◆ Accelerator and shortcut keys
- ◆ Scripts for Menu object events

Adding Menu objects

Each menu consists of at least one Menu object on the menu bar and Menu objects in a dropdown menu. You can add Menu objects in three places:

- ◆ To the menu bar
- ◆ To a dropdown menu
- ◆ To a cascading menu

Accelerator keys

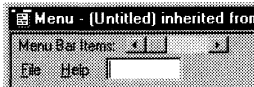
The ampersands and underscores in the illustrations indicate that an accelerator key (mnemonic) has been defined for the menu item.

FOR INFO For information about how to specify an accelerator key, see "Assigning accelerator and shortcut keys" on page 272.

❖ To add Menu objects to the menu bar:

- 1 Click the empty space to the right of the last defined Menu object on the menu bar at the top of the painter workspace.

PowerBuilder displays an empty box.



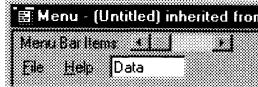
- 2 Type the text you want to display for the Menu object.
- 3 To add another Menu object to the menu bar, click to the right of the Menu object you just defined.

PowerBuilder displays another empty box.

- 4 Type the text for the new Menu object.
- 5 Repeat steps 3 and 4 to add additional items to the menu bar.

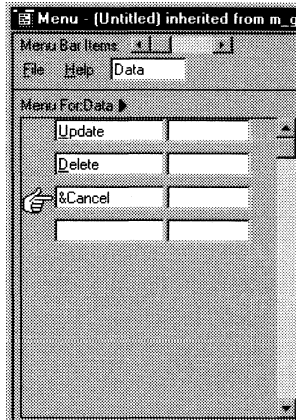
❖ **To add Menu objects to a dropdown menu:**

- 1 Click in the box for the item in the menu bar.



PowerBuilder displays the currently defined dropdown menu for the selected Menu object.

- 2 Press TAB to go to the first empty box under the Menu For heading.
- 3 Type the text you want to display for the Menu object.
- 4 Repeat steps 2 and 3 to add additional items to the dropdown menu.



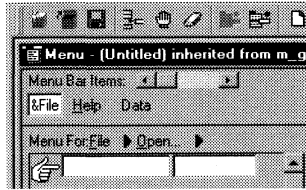
❖ **To add Menu objects to a cascading menu:**

- 1 Click in the box for the item in a dropdown menu that you want to attach a cascading menu to.

- 2 Click the Next Level button.

or

Select Edit>Next Level from the menu bar.



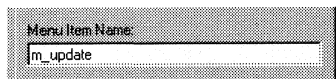
An empty box displays under the Menu For heading and the pointer moves to the box so you can build the cascading menu.

The Menu For heading displays the name of the Menu object, an arrowhead, the Menu object you selected, and another arrowhead to remind you that you are entering items in a cascading menu.

- 3 Enter items in the menu the same way you enter items in the dropdown menu.
- 4 To return to the previous menu level, click the Prior Level button or select Edit>Previous Level from the menu bar.

How Menu objects are named

When you add a Menu object, PowerBuilder gives it a default name, which displays in the Menu Item Name box.



This is the name by which you refer to a Menu object in a script (for example, it corresponds to `cb_name` for a `CommandButton`).

About the default Menu object names

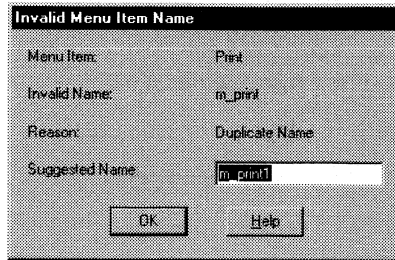
The default name is a concatenation of the text specified for the Prefix variable in the Options property sheet (initially the default prefix is `m_`) and the valid PowerBuilder characters and symbols in the text you typed for the Menu object.

If there are no valid characters or symbols in the text you typed for the Menu object, PowerBuilder creates a unique name `prefix_n`, where `n` is the lowest number that can be combined with the prefix to create a unique name.

The complete Menu object name (prefix and suffix) can be up to 40 characters. If the prefix and suffix exceed this size, PowerBuilder uses only the first 40 characters (without displaying a warning message).

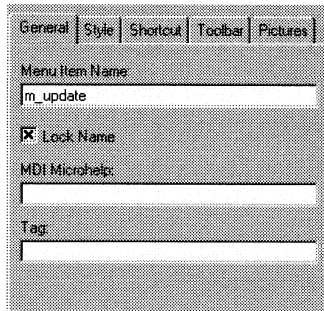
Duplicate Menu object names

If you add a Menu object to the menu and the name that PowerBuilder assigns has already been used in the menu, PowerBuilder displays a window and suggests a unique name for the Menu object.



Menu object names are locked by default

After you add a Menu object and move to another item, the name PowerBuilder assigns to the Menu object is locked.



Even if you later change the text that displays for the Menu object, PowerBuilder will not rename the Menu object. This is so you can change the text that displays in a menu without having to revise all your scripts that reference the Menu object (remember, you reference a Menu object through the name that PowerBuilder assigns to it).

You may want to change the text that displays for a Menu object and have PowerBuilder rename the Menu object based on the new text.

❖ To have PowerBuilder rename a Menu object:

- 1 Click the Lock Name checkbox to deselect it.
- 2 Change the text that displays for the Menu object.

Inserting Menu objects

You can insert Menu objects between existing Menu objects on the menu bar or in dropdown and cascading menus.

❖ **To insert a Menu object:**

- 1 Select the menu bar object before which you want to insert an object.
- 2 Click the Insert button in the PainterBar.
or
Select Edit>Insert from the menu bar.
An empty box displays.
- 3 Type the text of the new Menu object.

Moving Menu objects

To change the order of items in the menu bar or in a dropdown or cascading menu, you can drag the items to the order you want within their current menu. You cannot drag items from one menu to another (for example, you cannot drag an item in the menu bar to a dropdown menu).

❖ **To move a Menu object:**

- 1 Click the Move button in the PainterBar.
or
Select Edit>Move from the menu bar.
You are now in Move mode.
- 2 Press and hold the left mouse button on the Menu object you want to move.
The pointer becomes a hand pointer.
- 3 Drag the item to a new location in its menu.
- 4 Release the mouse button.
The Menu object displays in its new location and you leave Move mode.

Deleting Menu objects

❖ To delete a Menu object:

- 1 Select the Menu object you want to delete.
- 2 Click the Delete button in the PainterBar.
or
Select Edit>Delete from the menu bar.

The selected item is deleted.

Defining the appearance of Menu objects

You can use the Menu painter to change the appearance and behavior of your menu and Menu objects by choosing different settings on the property sheet tabbed pages.

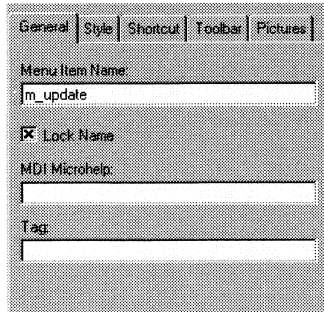
FOR INFO For a list of all Menu object properties, see *Objects and Controls*.

To do this	Choose this tab
Specify the name of the Menu object along with its MicroHelp and tag	General
Specify the type of Menu object and whether it is checked, enabled, visible, shifted, or merged	Style
Specify a shortcut key for the Menu object	Shortcut
Specify the toolbar entry for the Menu object	Toolbar
Associate a toolbar picture with the Menu object	Pictures

All of these properties are discussed below.

Setting General properties

When you select the General tab on the Menu painter property sheet, you can unlock menu item names (as described "How Menu objects are named" on page 266) as well as create MicroHelp Text and a tag for a Menu object.



Creating MicroHelp and tags

MicroHelp is a brief text description of the Menu object that displays on the status bar at the bottom of a Multiple Document Interface (MDI) application window. You can quickly add MicroHelp text to any Menu object in your application. A tag is a text string that you can associate with an object and use in any way you want.

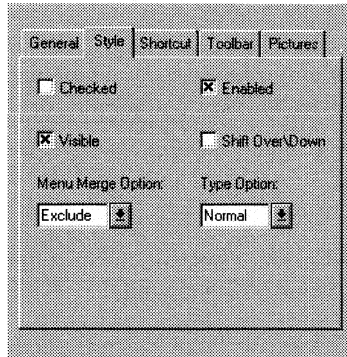
❖ **To create MicroHelp text:**

- 1 Select the General tab on the Menu Painter property sheet.
- 2 Select the Menu object for which you want to create MicroHelp.
- 3 Enter the MicroHelp text in the MDI MicroHelp edit control.

FOR INFO For information about defining MicroHelp text and tag properties, see the chapter on building MDI applications in *Application Techniques*.

Setting Style properties

When you select the Style tab on the property sheet, you can specify how a Menu object appears during execution. Each of the tab page options corresponds to a property of a Menu object.



Property	Meaning
Checked	Whether the Menu object displays with a checkmark next to it
Enabled	Whether the Menu object can be selected
Visible	Whether the Menu object is visible
Shift Over\Down	Whether the Menu object shifts to the right or down when you add Menu objects in a menu that is inherited from this menu. Selecting this property allows you to insert Menu objects in descendent menus, instead of only being able to add Menu objects to the end of menus in descendent menus FOR INFO For more information, see "Inserting Menu objects in a descendent menu" on page 287
Type Option	Whether the Menu object you are creating is Normal, About, or Exit type
Menu Merge Option	The way menus are modified when an OLE object is activated. Options are: File, Edit, Window, Help, Merge, Exclude FOR INFO For more information, see the chapter about using OLE in an application in <i>Application Techniques</i>

The settings you specify here determine how the Menu objects display by default. You can change the values of the properties in scripts during execution.

Assigning accelerator and shortcut keys

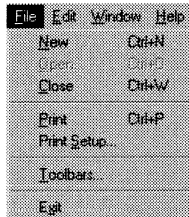
Every Menu object should have an **accelerator key**, also called a mnemonic access key, which allows users to select the item from the keyboard by pressing `ALT+key` when the menu is displayed. Accelerator keys display with an underline in the Menu object text.

On Macintosh Accelerator keys are not recognized on the Macintosh, but you can still define them for use on other platforms.

On UNIX On UNIX systems, users press `META+key` instead of `ALT+key` to select the menu item.

You can also define **shortcut keys**, which are combinations of keys that a user can press to select a Menu object anytime.

For example, in the following menu all Menu objects have accelerator keys (New's accelerator key is N, Open's is O, and so on). New, Open, Close, and Print each have shortcut keys (the CTRL key in combination with another key or keys).



You should adopt conventions for using accelerator and shortcut keys in your applications. All menu items should have accelerator keys, and commonly used menu items should have shortcut keys.

If you specify the same shortcut for more than one MenuItem, the command that occurs later in the menu hierarchy is executed.

Assigning accelerator keys

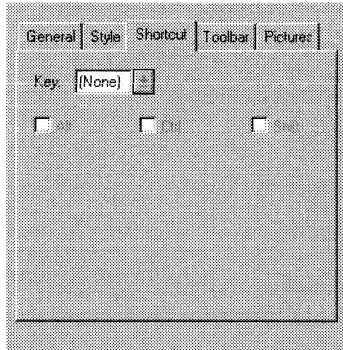
❖ To assign an accelerator key:

- ◆ Type an ampersand (&) before the letter in the Menu object text that you want to designate as the accelerator key.

For example, `&File` designates the F in File as an accelerator key and `Ma&ximize` designates the x in Maximize as an accelerator key.

Assigning shortcut keys

- ❖ **To assign a shortcut key:**
 - 1 Select the Menu object you want to assign a shortcut key to.
 - 2 Select the Shortcut tab on the property sheet.



- 3 Select a key from the Key dropdown listbox.
- 4 Select Alt, Ctrl, and/or Shift to create a key combination.

PowerBuilder displays the shortcut key in the box next to the Menu object text.

About Macintosh modifier keys

In the Menu painter, the modifier keys are the names of the Windows modifier keys. This table shows the modifier key that will display in your Macintosh PowerBuilder application.

When you select	The shortcut modifier is
ALT	None
CTRL	COMMAND
SHIFT	SHIFT

PowerBuilder uses the CONTROL key on the Macintosh only for functionality assigned to the right mouse button in Windows. When working in the development environment, pressing CONTROL and clicking on an object displays its popup menu. In a PowerBuilder application developed and deployed on the Macintosh, CONTROL+click triggers the RButtonDown event. If you want to display a popup menu, you can call the PopMenu function in the RButtonDown event's script.

Creating separation lines in menus

You should separate groups of related Menu objects with lines.

- ❖ **To create a line between items on a menu:**
 - ◆ Type a single dash (-) as the Menu object text.

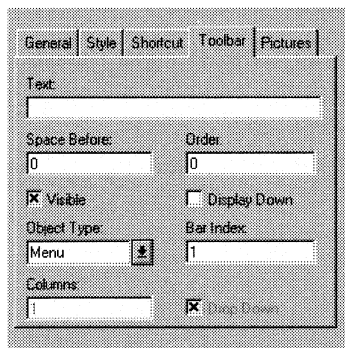
Setting toolbar and picture properties

In an MDI application you can associate a menu item with a button on a toolbar by performing two steps:

- ◆ Associate the Menu object with a toolbar button
- ◆ Associate the toolbar button with a picture

You can specify whether the Menu object will be visible on the toolbar, its placement on the toolbar, and the pictures it will use. You can also specify different pictures for the toolbar button up and down states.

- ❖ **To associate a Menu object with a toolbar button:**
 - 1 Select the Toolbar tab on the Menu painter property sheet.



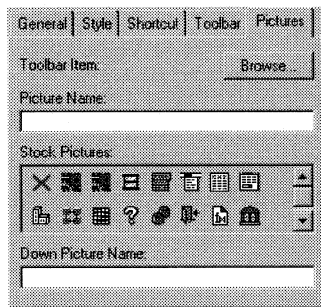
- 2 Enter the following information:

In this box	Do this
Text	Enter the descriptive text for the toolbar button
Space Before	Specify the amount of space you want before the button in the toolbar

In this box	Do this
Order	Specify the order for the button in the toolbar by entering a number. The default is to display buttons in the same order as in the menu
Visible	If you want the toolbar button to be visible, select the checkbox
Display Down	If you want the toolbar button to appear pressed, select the checkbox
Object Type	Specify whether the toolbar button will cascade by selecting either Menu or MenuCascade
Bar Index	Specify the number of the toolbar in which you want the toolbar button to appear
Columns	If you chose MenuCascade in the Object Type dropdown listbox, indicate the number of columns you want to display in the cascading toolbar
Drop Down	If you want the button to be a DropDown toolbar button, select the checkbox

❖ **To associate a Menu object with a toolbar picture:**

- 1 Select the Pictures tab on the Menu painter property sheet.



- 2 Enter the name of the picture you want to associate with the toolbar button's normal state in the Picture Name textbox by:
 - ◆ Selecting a stock picture
 - ◆ Using the Browse button to associate a cursor, bitmap, or icon file
- 3 Enter the name of the picture you want to associate with the toolbar button's pressed state in the Down Picture Name textbox by:
 - ◆ Selecting a stock picture

- ◆ Using the Browse button to associate a cursor, bitmap, or icon file

FOR INFO For information about defining toolbars and picture properties, see the chapter on building MDI applications in *Application Techniques*.

Saving menus

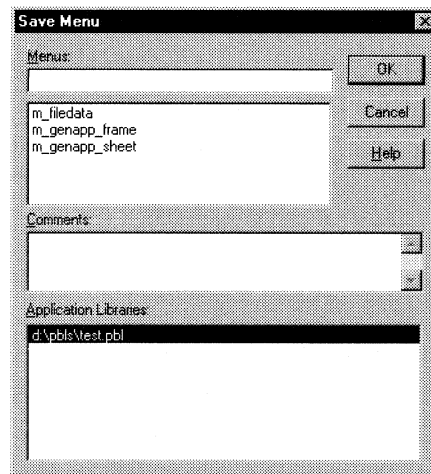
You can save the menu you are working on at any time. When you save a menu, PowerBuilder saves the compiled Menu objects and scripts in the library you specify.

❖ To save a menu:

- 1 Select File>Save from the menu bar.

If you have previously saved the menu, PowerBuilder saves the new version in the same library and returns you to the Menu painter.

If you have not previously saved the menu, PowerBuilder displays the Save Menu dialog box.



- 2 Name the menu in the Menus box (see "Naming the menu" next).
- 3 Write comments to describe the menu.

These comments display in the Select Menu dialog box and in the Library painter. It is a good idea to use comments so you and others can easily remember the purpose of the menu later.

- 4 Specify the library in which to save the menu and click OK.

Naming the menu

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

FOR INFO For information about PowerBuilder identifiers, see the *PowerScript Reference*.

A recommendation When you name menus, you should use a two-part name: a standard prefix that identifies the object as a menu (such as m_) and a suffix that helps you identify the particular menu.

For example, you might name a menu used in a sales application m_sales.

Viewing your work

While building a menu, you can preview it and print out its definition.

Previewing a menu

You can preview a menu at any time to see how it looks.

❖ **To preview a menu:**

- ◆ Click the Preview button.

or

Select Design>Preview from the menu bar.

The menu displays in a window.

Toggle buttons

Like many buttons in PowerBuilder, the Preview button is a toggle. When you click on it to preview an object, the button stays pressed while the object is being previewed. Clicking the Preview button again returns it (and the object you are previewing) to its normal state.

What you can do

You can navigate through the menu using the mouse or the keyboard.

What you cannot do

You cannot trigger events in the Menu objects. For example, clicking a Menu object while previewing does not trigger the item's Clicked event.

❖ **To return to the Menu painter:**

- ◆ Click the Preview button.

or

Select Design>Preview from the menu bar.

You return to the Menu painter.

Printing a menu's definition

You can print a menu's definition for documentation purposes.

❖ **To print information about the current menu:**

- ◆ Select File>Print from the menu bar.

Information about the current menu is sent to the printer specified in Printer Setup. The information that is sent to the printer depends on variables specified in the [Library] section of the PowerBuilder initialization file.

Print settings

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

Writing scripts for Menu objects

You write scripts that specify what happens when users select a Menu object.

❖ **To write a script for a Menu object:**

- 1 Select the Menu object.
- 2 Click the Script button in the PainterBar.
or
Select Edit>Script from the menu bar.

The PowerScript painter opens.

Menu object events

Menu objects have the following events:

◆ Clicked

Typically, your application will contain Clicked scripts for each Menu object in a dropdown or cascading menu. For example, the script for the Clicked event for the Open Menu object on the File menu will open a file.

◆ Selected

You will probably use few Selected scripts since users don't expect things to happen when they simply highlight a Menu object. (One possible use of Selected scripts, though, is to change MicroHelp displayed in an MDI application as the user scrolls through a menu.)

About the Clicked event

When it is triggered The Clicked event is triggered whenever:

- ◆ The Menu object is clicked with the mouse
- ◆ The Menu object is selected (highlighted) using the keyboard and then ENTER is pressed
- ◆ The shortcut key for the Menu object is pressed
- ◆ The menu containing the Menu object is displayed and the accelerator key is pressed

Whether a Menu object responds A Menu object responds to a mouse click or the keyboard only if both its Visible and Enabled properties are TRUE.

When its script is executed If the Menu object has a dropdown or cascading menu under it, the script for its Clicked event (if any) is executed when the mouse button is pressed, and then the dropdown or cascading menu displays. If the Menu object does not have a menu under it, the script for the Clicked event is executed when the mouse button is released.

Using Clicked event to specify Menu object properties

When the user clicks an item on the menu bar to display a dropdown menu, the Clicked event for the Menu object on the menu bar is triggered and then the dropdown menu is displayed.

You can use the menu bar's Clicked event to specify the properties of the Menu objects in the dropdown menu. For example, if you want to disable items in a dropdown menu, you can disable them in the script for the Clicked event for the Menu object in the menu bar.

About the Selected event

The Selected event is triggered when the user selects (highlights) a Menu object.

Using functions and variables

You can use functions and variables in your scripts.

Using functions

PowerBuilder provides built-in functions that act on Menu objects. You can use these functions in scripts to manipulate Menu objects during execution. For example, to hide a menu, you can use the Hide built-in function.

FOR INFO For a complete list of the menu-level built-in functions, see the Browser.

Defining menu-level functions

You can define your own menu-level functions to make it easier to manipulate your menus.

FOR INFO For more information, see Chapter 5, "Working with User-Defined Functions".

Using variables

Scripts for Menu objects 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 also declare instance variables for the menu when you have data that needs to be accessible to scripts in several Menu objects in a menu. Instance variables are accessible to all Menu objects in the menu.

FOR INFO For a complete description of variables and how to declare them, see the *PowerScript Reference*.

Defining menu-level structures

If you need to manipulate a collection of related variables, you can define menu-level structures.

FOR INFO For more information, see Chapter 6, "Working with Structures".

Referring to objects in your application

You can refer to any object in the application in scripts for Menu objects. You must fully qualify the reference, using the object name, as follows.

Referring to windows

When referring to a window, you simply name the window:

window

When referring to a property in a window, you must always qualify the property with the window's name:

window.property

For example, to move the window `w_cust` from within a Menu object script, code:

```
w_cust.Move(300, 300)
```

To minimize `w_cust`, code:

```
w_cust.WindowState = Minimized!
```

Using ParentWindow You can use the reserved word `ParentWindow` to refer to the window that the menu is associated with during execution. For example, the following statement closes the window the menu is associated with:

```
Close(ParentWindow)
```

You can also use `ParentWindow` to refer to 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, to enable a `CommandButton` in window `w_cust` from a `Menu` object script, code:

```
w_cust.cb_print.Enabled = TRUE
```

Referring to Menu objects

When referring to a `Menu` object, use this syntax:

```
menu.Menu object
```

```
menu.Menu object.property
```

Reference within the same menu

When referring to a `Menu` object within the same menu, you don't have to qualify the reference with the menu name.

When referring to a Menu object in a dropdown or cascading menu, you must specify each Menu object on the path to the Menu object you are referencing, separating the names with periods.

For example, to place a checkmark next to the Menu object `m_bold`, which is on a dropdown menu under `m_text` in the menu saved in the library as `m_menu`, code:

```
m_menu.m_text.m_bold.Check( )
```

If the above script were for a Menu object in the same menu (`m_menu`), you wouldn't need to qualify the Menu object with the name of the menu. You could simply code:

```
m_text.m_bold.Check( )
```


Using inheritance to build a menu

When you build a menu that inherits its style, events, functions, structures, variables, and scripts from an existing menu, you save coding time. All you have to do is modify the descendent object to meet the requirements of the current situation.

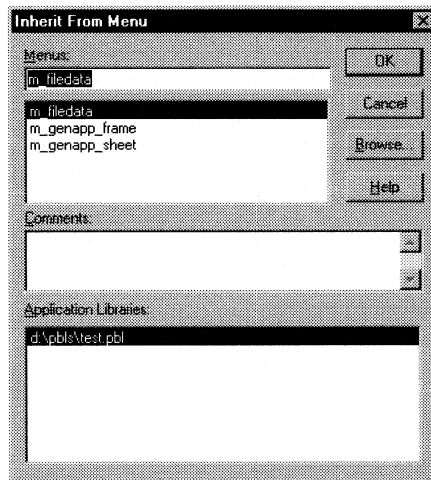
❖ **To use inheritance to build a descendent menu:**

- 1 Open the Menu painter.

The Select Menu dialog box displays.

- 2 Click the Inherit button.

The Inherit From Menu window displays listing the menus in the current library.



- 3 Select the menu you want to use to create the descendant.

The selected menu displays in the workspace. The title of the workspace indicates that the menu is a descendant.

- 4 Make the changes you want to the descendant menu as described in the next section.
- 5 Save the menu under a new name.

Using the inherited information

When you build and save a menu, PowerBuilder treats the menu as a unit that includes:

- ◆ All Menu objects and their scripts
- ◆ Any variables, functions, and structures declared for the menu

When you use inheritance to build a menu, everything in the ancestor menu is inherited in all of its descendants.

What you can do

You can do the following in a descendent menu:

- ◆ Add Menu objects to the end of a menu
- ◆ Insert Menu objects in a menu

FOR INFO For more information, see "Inserting Menu objects in a descendent menu" on page 287.

- ◆ Modify existing Menu objects

For example, you can change the text displayed for a Menu object or change its initial appearance, such as making it disabled.

- ◆ Build scripts for Menu objects that don't have scripts in the ancestor menu
- ◆ Extend or override inherited scripts
- ◆ Declare functions, structures, and variables for the menu

What you cannot do

You cannot do the following in a descendent menu:

- ◆ Change the order of inherited Menu objects
- ◆ Delete an inherited Menu object

Hiding a Menu object

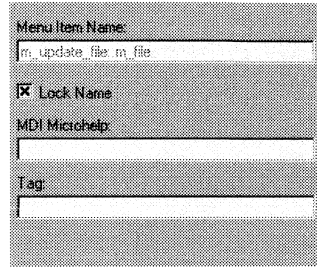
If you don't need a Menu object in a descendent menu, you can hide it using the Hide built-in function.

About Menu object names

PowerBuilder uses the following syntax to show names of inherited Menu objects:

ancestormenu::Menu object

For example, in a menu derived from `m_update_file`, you see the following for the `m_file` Menu object, which is defined in `m_update_file`:



Understanding inheritance

The issues concerning inheritance with menus are the same as the issues concerning inheritance with windows and user objects.

FOR INFO Chapter 9, "Understanding Inheritance", describes the issues in detail:

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

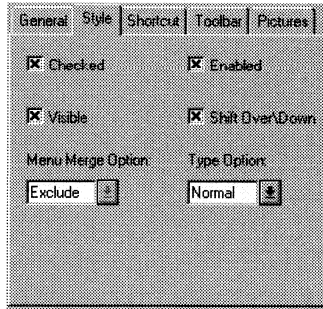
Inserting Menu objects in a descendent menu

When defining a descendent menu, you might want to insert Menu objects in the middle of a dropdown or cascading menu or in the middle of the menu bar.

❖ To insert Menu objects in a menu in a descendent menu:

- 1 In the parent menu, select Shift Over\Down for all of the following Menu objects:
 - ◆ Menu objects in the menu bar that need to be shifted right when you insert an item in the menu bar in a descendent menu

- ◆ Menu objects in a dropdown or cascading menu that need to be shifted down when you insert an item in a dropdown or cascading menu



- 2 In the descendent menu, add Menu objects to the end of the menu bar or at the end of the appropriate dropdown or cascading menus in the Menu painter workspace.

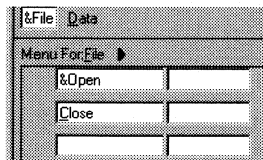
At execution time, PowerBuilder inserts the added Menu objects before all the Menu objects specified as shifting over/down in the parent.

About the display in the Menu painter

When you insert Menu objects in a descendent menu in the Menu painter, the workspace does not show the insertions; it shows all added Menu objects at the end of the menu. However, when you preview the menu, as described in "Previewing a menu" on page 278, the Menu objects are shown in the order in which they will display in your application.

An example of inserting Menu objects

Say you have the following menu, m_update_file.



You want to define a menu that is inherited from this menu, with the following modifications:

- ◆ A new Menu object, File>New, which displays above File>Open and File>Close
- ◆ A new item on the menu bar, Edit, which displays between File and Data

In other words, you want File>Open and File>Close to shift down in the descendant's dropdown menu, and you want Data to shift right in the descendant's menu bar.

Here is how you would do it:

- 1 Open `m_update_file` in the Menu painter.
- 2 Select Shift Over\Down for each of the following Menu objects:
 - ◆ File>Open and File>Close (you want these items to shift down when you insert File>New in the descendant)
 - ◆ Data in the menu bar (you want this item to shift right when you insert Edit in the menu bar in the descendant)
- 3 Save and close `m_update_file`.
- 4 Create a new menu inherited from `m_update_file`.
- 5 Add a New Menu object to the File dropdown menu.

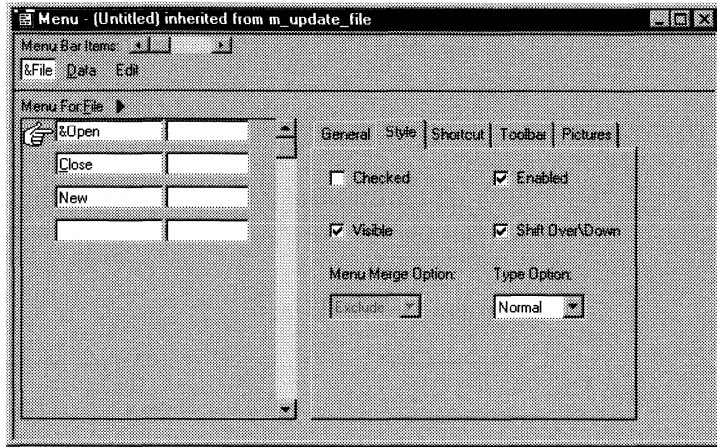
The New item displays at the end of the dropdown menu in the workspace.

- 6 Add an Edit Menu object to the menu bar.

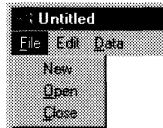
The Edit item displays at the end of the menu bar in the workspace.

Because you defined File>Open, File>Close, and Data as Shift Over\Down, when you execute the descendent menu, the new Menu objects are inserted as you want. You can see this when you preview the menu.

Here is the workspace:



During preview and at execution time, the Menu objects defined in the descendant are inserted appropriately:



Using menus

You can use menus in two ways:

- ◆ Place them in the menu bar of a window
- ◆ Display a menu as a popup menu

Adding a menu bar to a window

To 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 Open the Window painter and select the window you want to associate the menu with.
- 2 Select Properties from the window's popup menu and select the General tab.

The window's property sheet displays with the General property page on top.

- 3 Enter the name of the menu in the Menu Name textbox.
or
Click the Browse button and select the menu from the Select Menu dialog box, which lists all menus available to the application.
- 4 Click OK to associate the selected menu with the window.

You reference Menu objects in scripts in windows and controls using the following syntax:

menu.Menu object

You must always fully qualify the Menu object with the name of the menu.

When referring to a Menu object in a dropdown or cascading menu, you must specify each Menu object on the path to the Menu object you are referencing, separating the names with periods.

For example, to refer to the Enabled property of Menu object `m_open`, which is under the menu bar item `m_file` in the menu saved in the library as `m_menu`, code:

```
m_menu.m_file.m_open.Enabled
```

Identifying Menu objects in window scripts

Changing a window's menu during execution

You can use the `ChangeMenu` function in a script to change the menu associated with a window during execution.

FOR INFO For information about `ChangeMenu`, see the *PowerScript Reference*.

Displaying popup menus

To display a popup menu in a window, use the `PopupMenu` 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 `PopupMenu` function.

The following statement in a `CommandButton` script displays `m_appl.m_help` as a popup menu at the current pointer position (assuming menu `m_appl` is already associated with the window):

```
m_appl.m_help.PopupMenu(PointerX(), PointerY())
```

If the menu is not associated with the window

If the menu is not already associated with the window, you must create an instance of the menu before you can display it as a popup menu.

The following statements create an instance of the menu `m_new`, then pop up the menu `m_new.m_file` at the pointer location (assuming `m_new` is not associated with the window containing the script):

```
m_new    mymenu

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

For more information

For complete information about the `PopupMenu` function, see the *PowerScript Reference*.

Working with User Objects

About this chapter

One of the features of object-oriented programming is reusability: you define a component once, then reuse it as many times as you need without any additional work. One of the best ways to get reusability in PowerBuilder is with user objects. This chapter describes how to define and use user objects.

Contents

Topic	Page
Overview of user objects	294
Building a new user object	298
Using inheritance to build user objects	307
Using user objects	309
Communicating between a window and a user object	314

Overview of user objects

Applications often have features in common. For example, you might often reuse features like the following:

- ◆ A Close button that performs a certain set of operations and then closes the window
- ◆ A listbox that lists all departments
- ◆ DataWindow controls that perform the same type of error checking
- ◆ A predefined file viewer that you plug into a window
- ◆ A processing package that calculates commissions or performs statistical analysis

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 types of user objects:

- ◆ Visual
- ◆ Class

Examples of user objects

The PowerBuilder Code Examples contain many interesting user objects in PBEXAMUO.PBL. Take a look at them to get an appreciation for the power of user objects.

Visual user objects

A **visual user object** is a reusable control or set of controls that has a certain behavior. You define it in the User Object painter, where you place controls in the user object and write scripts for those controls. Then you can place the user object in windows you build in your applications as often as needed.

There are three types of visual user objects:

- ◆ **Standard** Most useful if you frequently use a PowerBuilder control to perform the same processing

- ◆ **Custom** Most useful if you frequently group controls together in a window and always use the controls to perform the same processing
- ◆ **External** Useful when you have a custom DLL

Standard visual user objects

A standard visual user object inherits its definition from one standard PowerBuilder control. You modify the definition to make the control specific to your applications.

Assume you frequently use a `CommandButton` named `Close` to display a message box and then close the parent window. If you build a standard visual user object that derives from a `CommandButton` to perform this processing, you can use the user object whenever you want to display a message box and then close a window.

Custom visual user objects

Custom visual user objects are objects that have several controls that function as a unit. You can think of a custom visual user object as a window that is a single unit and is used as a control.

Assume you frequently use a group of buttons, each of which performs standard processing. If you build a custom user object that contains all the buttons, you can place the buttons in the window as a unit when you place the user object in a window.

External visual user objects

External visual user objects contain controls from objects in the underlying windowing system that were created outside PowerBuilder. You can use a custom DLL in PowerBuilder to create an external user object.

You must know what classes the DLL supports, the messages or events the DLL responds to, and the style bits that you can set in the DLL.

On UNIX

On UNIX systems, the files you access for such controls must be shared libraries that have been compiled under UNIX.

FOR INFO For more information about custom DLL controls on UNIX, see the chapter on adding other processing extensions in *Application Techniques*.

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.

You build class user objects in the User Object painter, specifying instance variables (*properties* in object-oriented terminology) and object-level functions (*methods*). Then you create an instance of the class user object in your application, thereby making its processing available.

Whenever you need to do this type of processing, you would instantiate the user object in a script and call its functions.

There are two kinds of class user objects:

- ◆ Standard
- ◆ Custom

Standard class user objects

A standard class user object inherits its definition from one built-in, nonvisual PowerBuilder object, such as the Transaction object or Error object. You modify the definition to make the object specific to your application, and optionally add instance variables and functions to enhance the behavior of the built-in object. Once you define a standard class user object, you can go to the Application painter and specify that you want to use it instead of the corresponding built-in system object in your application.

One important use of a standard class user object is using one inherited from the built-in Transaction object to do database remote procedure calls from within an application.

Custom class user objects

Custom class user objects are objects of your own design that encapsulate properties and functions not visible to the user. They are not derived from PowerBuilder objects. You define them to create units of processing that have no visual component.

For example, to calculate commissions in an application, you can define a Calculate_Commission 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.

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 INFO For information on building a user object from scratch, see "Building a new user object" next. To find out more about creating a user object based on an existing PowerBuilder object, see "Using inheritance to build user objects" on page 307.

Building a new user object

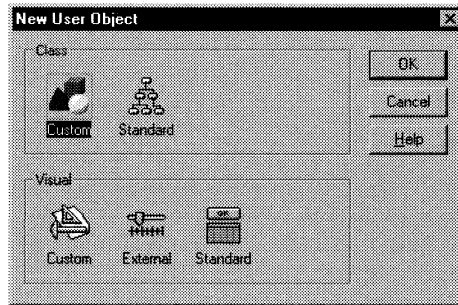
This section describes how to build a user object from scratch. You use this technique to create user objects that aren't based on existing user objects.

Opening the User Object painter

❖ **To open the User Object painter and create a new object:**

- 1 Click the UserObject button in the PowerBar or PowerPanel.
- 2 Click the New button to build a new user object.

The New User Object dialog box displays.



- 3 Select the type of user object you want to build and click OK.

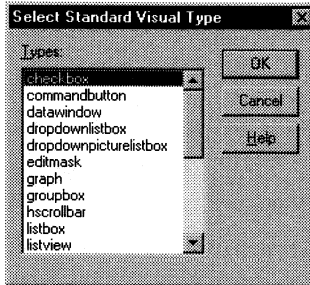
What you do next depends on the type of user object you selected. The remainder of this section describes how to build each type of user object.

Building a standard visual user object

❖ To build a standard visual user object:

- 1 In the New User Object dialog box, select Standard in the Visual group box and click OK.

The Select Standard Visual Type dialog box displays listing the control types available.



- 2 Select the PowerBuilder control you want to use to build your standard visual user object and click OK.

The selected control displays in the workspace. Your visual user object will have the properties and events associated with the PowerBuilder control you are modifying.

- 3 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 User Object painter workspace or PowerScript painter.

- ◆ Declare any needed user events for the user object.

FOR INFO For information about user events, see "Communicating between a window and a user object" on page 314.

- ◆ Create and compile the scripts for the user object.

Standard visual user objects have the same events as the PowerBuilder control you modified to create the object.

- 4 Save the user object.

FOR INFO See "Saving a user object" on page 305.

Building a custom visual user object

This procedure for building a custom visual user object is similar to the process of building a window, described in Chapter 7, "Defining Windows".

❖ **To build a custom visual user object:**

- 1 In the New User Object dialog box, select Custom in the Visual group box and click OK.

The User Object painter workspace displays. It looks like the Window painter workspace, but the empty box is the new custom visual user object.

- 2 Place the controls you want in the custom visual user object.
- 3 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

You can declare these in the User Object painter workspace or PowerScript painter.

- ◆ Declare any needed events for the user object or its controls

FOR INFO For information about user events, see "Communicating between a window and a user object" on page 314.

- ◆ 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 INFO For more information on events associated with custom visual user objects, see "Events in user objects" on page 304.

- 4 Save the user object.

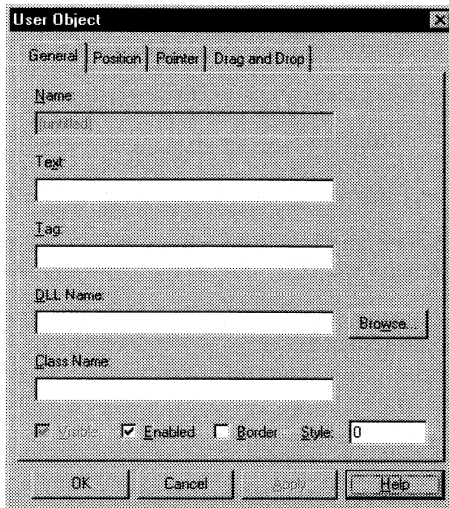
FOR INFO See "Saving a user object" on page 305.

Building an external visual user object

❖ To build an external visual user object:

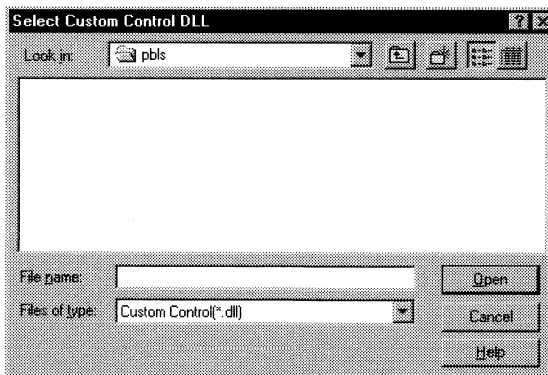
- 1 In the New User Object dialog box, select External in the Visual group box and click OK.

The User Object property sheet displays.



- 2 Click the Browse button next to the DLL Name textbox.

The Select Custom Control DLL dialog box displays.



- 3 Select the DLL that defines the user object and click OK.

You return to the User Object property sheet.

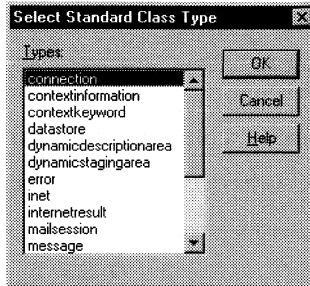
- 4 Enter the following information, as necessary, and click OK:
 - ◆ The class name registered in the DLL
Information about the class name is usually provided by the vendor of the purchased DLL.
 - ◆ Text in the Text box
This will only be displayed if the object has a text style property.
 - ◆ Display properties (border and scrollbars)
 - ◆ Decimal values for the style bits associated with the class
Information about style bits is usually provided by the vendor of the purchased DLL. PowerBuilder will OR these values with the values selected in the display properties for the control.
- 5 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 User Object painter workspace or in the PowerScript painter. Information about functions is usually provided by the vendor of the purchased DLL.
- 6 Declare any needed events for the user object.
FOR INFO For information about user events, see "Communicating between a window and a user object" on page 314.
- 7 Create and compile the scripts for the user object.
FOR INFO For more information on events associated with external visual user objects, see "Events in user objects" on page 304.
- 8 Save the user object.
FOR INFO See "Saving a user object" on page 305.

Building a standard class user object

❖ To build a standard class user object:

- 1 In the New User Object dialog box, select Standard in the Class group box and click OK.

The Select Standard Class Type dialog box displays.



- 2 Select the built-in system object that you want your user object to inherit from and click OK.
- 3 Declare functions, structures, or variables you need for the user object.

You can declare these in the User Object painter workspace or PowerScript painter.

For a list of properties and functions

In the PowerScript painter, use the Browser to list the built-in properties and functions inherited from the selected system object.

- 4 Declare any needed user events for the user object.
FOR INFO For information about user events, see "Communicating between a window and a user object" on page 314.
- 5 Create and compile scripts for the user object.
Class user objects have constructor and destructor events.
- 6 Save the user object.
FOR INFO See "Saving a user object" on page 305.

Building a custom class user object

❖ To build a custom class user object:

- 1 In the New User Object dialog box, select Custom in the Class group box and click OK.
The User Object painter workspace opens.
- 2 Declare functions, structures, or variables you need for the user object.
You can declare functions, structures, and variables for the user object in the User Object painter workspace or in the PowerScript painter.
- 3 Create and compile scripts for the user object.
Class user objects have constructor and destructor events.
- 4 Save the user object.
FOR INFO See "Saving a user object" on page 305.

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:

- ◆ Select AutoInstantiate from the user object's popup menu.

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

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

Class user objects have only constructor and destructor events.

Event	Occurs
Constructor	When the user object is created
Destructor	When the user object is destroyed

Events in visual user objects

Standard visual user objects have the same events as the PowerBuilder control from which they inherit. Custom and external visual user objects have the following common set of events:

Event	Occurs when
Constructor	Immediately before the Open event of the window and when the user object is dynamically placed in a window
Visual	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
Other	A Windows message occurs that is not a PowerBuilder event
RButtonDown	The right mouse button is pressed (on the Macintosh, when the CONTROL key and the mouse button are pressed)

FOR INFO For more about drag and drop, see *Application Techniques*.

Saving a user object

❖ To save a user object:

- 1 In the User Object painter, select File>Save from the menu bar.

If you have previously saved the user object, PowerBuilder saves the new version in the same library and returns you to the User Object painter.

If you have not previously saved the user object, PowerBuilder displays the Save User Object dialog box.

- 2 Enter a name in the User Objects box.

FOR INFO For naming considerations, see "Naming the user object" next.

- 3 Enter comments to describe the user object.

These display in the Select User Object dialog box and in the Library painter, and will document the purpose of the user object.

- 4 Specify the library in which to save the user object.

To make a user object available to all applications, save it in a common library and include the library in the library search path for each application.

- 5 Click OK to save the user object.

Naming the user object

A user object name can be any valid PowerBuilder identifier up to 40 characters.

FOR INFO For information about PowerBuilder identifiers, see the *PowerScript Reference*.

Recommended naming convention

You should adopt naming conventions to make it easy to understand a user object's type and purpose. Here is a convention you could follow.

Use `u_` as the prefix for all user objects and apply the prefix for the various types of user objects as follows:

Type of user object	Format	Example
Standard visual	<code>u_control_purpose</code>	<code>u_cb_close</code> , a <code>CommandButton</code> that closes a window
Custom visual	<code>u_purpose</code>	<code>u_toolbar</code> , a toolbar
External visual	<code>u_ex_purpose</code>	<code>u_ex_sound</code> , outputs sound
Standard class	<code>u_systemobject_purpose</code>	<code>u_trans_test</code> , derived from the <code>Transaction</code> object and used for testing
Custom class	<code>u_cust_purpose</code>	<code>u_cust_commission</code> , calculates commissions

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, assume your application has a user object `u_file_view` that has three `CommandButtons`:

- ◆ List—displays a list of files in a `ListBox`
- ◆ Open—opens the selected file and displays the file in a `MultiLineEdit`
- ◆ Close—displays a message box and then closes the window

Then assume you want to build another user object that is exactly like the existing `u_file_view` except it has a fourth `CommandButton`. If you use inheritance to build the new user object, all you have to do is add the fourth `CommandButton`.

❖ To use inheritance to build a descendent user object:

- 1 Open the User Object painter.
- 2 Click the Inherit button.
- 3 Select the user object you want to use to create the descendant and click OK.

The selected object displays in the workspace and the workspace title bar indicates that the object is a descendant.

- 4 Make any changes you want to the user object.
- 5 Save the user object with a new name.

Using the inherited information

When you build and save a user object, PowerBuilder treats the object as a unit that includes:

- ◆ The object (and any controls within the object if it is a custom visual user object)
- ◆ The object's 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.

What you can do

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 (if it is a custom visual user object)
- ◆ Reference the ancestor's functions and events
- ◆ Reference the ancestor's structures if the ancestor contains a public or protected instance variable of the structure datatype
- ◆ Access ancestor properties, such as instance variables, if the scope of the property is public or protected
- ◆ Declare variables, events, functions, and structures for the descendant

What you cannot do

In a descendent user object, you cannot delete controls inherited from a custom visual user object. If you don't 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.

Chapter 9, "Understanding Inheritance", describes in detail:

- ◆ Inheritance basics
- ◆ Viewing the inheritance hierarchy
- ◆ Considerations when using inherited objects
- ◆ Using inherited scripts
- ◆ Calling an ancestor script
- ◆ Calling an ancestor function

Using user objects

Once you have built your user object, you are ready to use it in your 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 Controls>User Object from the menu bar.
- 3 Select the user object you want to use and click the location where you want the user object to display.

PowerBuilder creates a descendent user object that inherits its definition from the selected user object and places it in the window or user object.

Creating user object buttons

You can place buttons for the user objects you use frequently in the PainterBar in the Window painter and User Object painter. Then you can simply click the button to place a user object.

FOR INFO For more information, see "Adding a custom button" on page 23.

What you can do

After you place a user object in a window or a custom visual user object, you can name it, size it, position it, write scripts for it, and anything else you can do with a control.

When you place a user object in a window, PowerBuilder assigns it a unique name, just as it does when you place a control. The name is a concatenation of the default prefix for a user control (initially, `uo_`) and a default suffix, which is a number that makes the name unique.

You should change the default suffix to a suffix that has meaning for the user object in your application.

FOR INFO For more information about naming, see "Naming controls" on page 201.

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 INFO For more information, see "Using inherited scripts" on page 253.

You place a user object *as a unit* in a window (or another user object). You cannot write scripts for individual controls in a custom user object after placing it in a window or custom user object; you do that only when you are defining the user object itself.

Placing a user object during execution

You can add a user object to a window during execution using the PowerScript functions `OpenUserObject` and `OpenUserObjectWithParm` in a script. You can remove a user object from a window using the `CloseUserObject` function.

FOR INFO For information about these functions, see the *PowerScript Reference*.

Using class user objects

Class user objects have no visible component, so they are not placed in windows as visual user objects are. To use a class user object, you create an instance of it in a script.

Using standard class user objects

If you have built a standard class user object inherited from one of the built-in global objects, you can install it as the default for your application instead of manually creating an instance of it.

FOR INFO For more information, see "Using global standard class user objects" on page 312.

❖ To use a class user object:

- 1 Declare a variable of the user object type and create an instance of it using the CREATE statement.

The properties and functions defined in the user object are now available within the scope of the variable declaration. You reference them using dot notation, just as you access properties and functions of other PowerBuilder objects.

- 2 Use the user object's properties to do the processing you want.
- 3 When you have finished using the user object, destroy it using the DESTROY statement.

Autoinstantiating custom class user objects

If you select Autoinstantiate from the popup menu of a custom class user object, you must declare a variable of the user object type, but you do not need to use the CREATE and DESTROY statements.

Examples

For example, if you have defined a custom class user object named `u_cust_commission` that calculates commissions, you can use the user object whenever you need to calculate a commission in a script, as follows:

```
// First, declare and create the user object.
u_cust_commission MyClassUO
MyClassUO = CREATE u_cust_commission

// Now have access to the properties and functions
// encapsulated in the user object.
// Do the processing here.
...
// When through, destroy the user object.
DESTROY MyClassUO
```

FOR INFO For more information about autoinstantiating user objects, see the *PowerScript Reference*.

Using global standard class user objects

Five of the standard class user object types are inherited from predefined global objects used in all PowerBuilder applications:

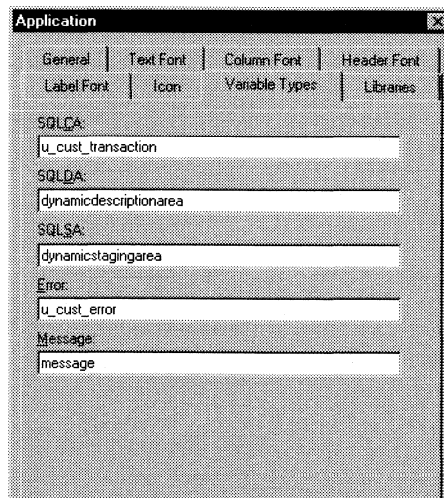
- ◆ Transaction (SQLCA)
- ◆ DynamicDescriptionArea (SQLDA)
- ◆ DynamicStagingArea (SQLSA)
- ◆ Error
- ◆ Message

Replacing the built-in global object

If you want your standard class user object to *replace* the built-in global object, you tell PowerBuilder to use your user object *instead of* the built-in system object that it inherits from. You will probably use this technique if you have built a user object inheriting from the Error or Message object.

❖ To replace the built-in global object with a standard class user object:

- 1 In the Application painter, select Properties from the Application popup menu and choose the Variable Types tab.
- 2 Specify the standard class user object you defined in the corresponding field and click OK.



Changing your mind

Click Reset to reset all listed global objects to the defaults shipped with PowerBuilder.

Once you have installed your user object as the default global object, it replaces the built-in object and is created automatically when the application starts up. You do not create it (or destroy it) yourself.

The properties and functions defined in the user object are available anywhere in the application. Reference them using dot notation, just as you access those of other PowerBuilder objects such as windows.

Supplementing the built-in global object

You can use a user object inherited from one of these global objects by declaring an instance of it in a script, as described in "Using class user objects" on page 310. 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 information about	See
Using the Error object	"Using the Error object" on page 845
Creating your own Transaction object to support database remote procedure calls	<i>Application Techniques</i>
Using the Message object	<i>Application Techniques</i>
DynamicDescriptionArea and DynamicStagingArea (used in dynamic SQL)	<i>PowerScript Reference</i>

Communicating between a window and a user object

Often you need to exchange information between a window and a visual user object in the window. Consider these situations:

- ◆ You have a set of buttons in a custom user object. Each of the buttons acts upon a file that is listed in a SingleLineEdit control in the window (but not in the user object).

You need to pass the contents of the SingleLineEdit control from the window to the user object

- ◆ You have a user object color toolbar. When the user clicks one of the colors in the user object, a control in the window should change to that color.

You need to pass the color from the user object to the window control

This section discusses two methods of handling this communication and presents a simple example.

Two methods

Method	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, may be unnecessary for simple operations
User events	Very flexible and powerful	Uses no type checking so is prone to error

Communication with both methods can be either synchronous (using Send for functions and TriggerEvent for events) or asynchronous (using Post for functions and PostEvent 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"
```

This method is not recommended.

The functions method

Exchanging information using functions is straightforward. Once a user object calls a function, any return value is available to any control within that object.

FOR INFO For how to use this technique, see "Example 1: using functions" on page 317.

❖ 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 method

You can define your own events, called **user events**, to communicate between a window and a user object. You can declare user events for any PowerBuilder object or control.

In most cases, you do not need to define your own events; PowerBuilder predefines the events you need.

A *custom visual* user object often requires a user event. After you place a custom visual user object in a window or in another custom user object, you can write scripts only for events that occur in the user object itself; you cannot write scripts for events in the *controls* in the user object.

So you define user events for the user object and trigger those events in scripts for the controls contained in that user object; then in the Window painter, you write scripts for the user events, referencing components of the window as needed.

FOR INFO See Chapter 12, "Working with User Events", and *Application Techniques*. For how to use this technique, see "Example 2: using user events" on page 319.

❖ **To define and trigger a user event in a user object:**

- 1 In the User Object painter, select the user object.
Make sure no control in the user object is selected.
- 2 Select Declare>User Events from the menu bar.
- 3 Type the name of the user event in the Event Name box.
To map this user event to a Windows event, double-click the associated event ID in the Paste Event ID box.
If you will not map this user event to a Windows event, leave the Event ID field blank.
- 4 Repeat step 3 to define additional user events as appropriate, and click OK when done.
- 5 Use the Event call 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.

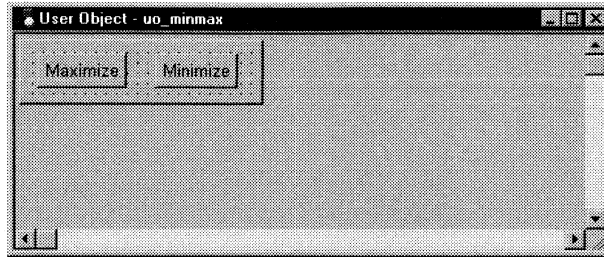
- 6 Implement these user events in the Window painter.

❖ **To implement the user event in the window:**

- 1 Open the Window painter and select the appropriate window.
- 2 Place the custom visual user object in the window.
- 3 Select the user object and open the PowerScript painter.
- 4 Write scripts for the user events you defined in the User Object painter.

Two examples: user object controls affecting a window

To illustrate these techniques, consider a simple custom visual user object, `uo_minmax`, that contains two buttons, `Maximize` and `Minimize`. If the user clicks the `Maximize` button, the current window becomes maximized. If the user clicks `Minimize`, the window closes to an icon.



Because the user object can be associated with any window, the scripts for the buttons cannot reference the window that has the user object. The user object must retrieve the name of the window so the buttons can reference the window.

"Example 1: using functions" next shows how PowerBuilder uses functions to pass a window name to a user object, allowing controls in the user object to affect the window.

"Example 2: using user events" on page 319 shows how PowerBuilder uses unmapped user events to allow controls in a user object to affect the window that has the user object

Example 1: using functions

What you do

- 1 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.

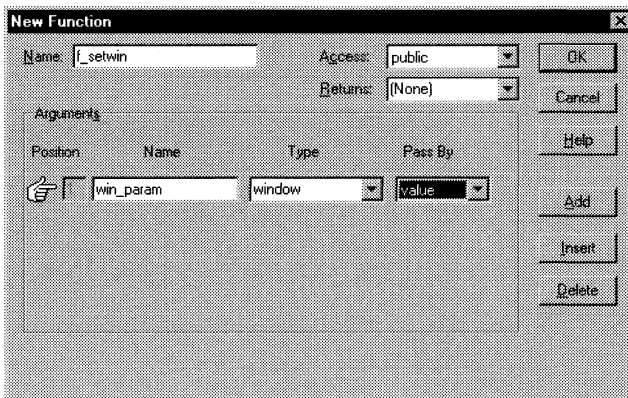
FOR INFO For information about declaring variables of type `window`, see Chapter 7, "Defining Windows".

- 2 Declare 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 Define 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:

Button	Script
<code>cb_max</code>	<code>mywin.WindowState = Maximized!</code>
<code>cb_min</code>	<code>mywin.WindowState = Minimized!</code>

5 Save the user object as `uo_minmax` and close the User Object painter.

6 In the Window painter, place `uo_minmax` in the window and name it `uo_func`.

7 In the Open event for the window, call the user object-level function, passing the name of the window:

```
uo_func.f_setwin(This)
```

The pronoun `This` refers to the window's name, which will be passed to the user object's `f_setwin()` function.

What happens

When the window opens, it calls the user object-level function `f_setwin()`, which passes the window name to the user object. The user object stores the name in its instance variable `mywin`. When the user clicks a button control in the user object, the control references the window through `mywin`.

Example 2: using user events

What you do

- 1 In the User Object painter, define two unmapped user events for the user object: Max_requested and Min_requested.

Leave the Event ID fields blank to define them as unmapped.

- 2 Trigger user events of the user object in the CommandButton scripts:

Button	Script for the Clicked event
cb_max	Parent.Event Max_requested()
cb_min	Parent.Event Min_requested()

- 3 Save the user object and close the User Object painter.
- 4 In the Window painter, place the user object in the window and name it uo_event.
- 5 Select uo_event and open the PowerScript painter.
The two new user events display in the Select Event listbox.
- 6 Write scripts for the two user events:

Event	Script
max_requested	Parent.WindowState = Maximized!
min_requested	Parent.WindowState = Minimized!

These scripts reference the window containing the user object with the pronoun Parent.

What happens

When a user clicks a button, the Clicked event script for that button triggers a user event in its parent, the user object. The user object script for that event modifies its parent, the window.

Working with User Events

About this chapter

This chapter introduces user events, describes how to define them, and discusses using them in an application.

Contents

Topic	Page
Overview of user events	322
Defining user events	325
Using a user event	328

Overview of user events

Windows, user objects, and controls each have a predefined set of events. In most cases, the predefined events are all you need.

But there are times when you want to declare your own event for a window, user object, or control. These events are called **user events**. You can use predefined event IDs to trigger a user event, or you can trigger it exclusively from within your application scripts.

Here are some possible uses of user events:

- ◆ **Keystroke processing** You want to modify the way keystrokes are processed in your application. For example, in a DataWindow control, you want the user to be able to press DOWN ARROW and UP ARROW to scroll among radio buttons in a DataWindow column (normally, pressing these keys moves the focus to the next or preceding row).

To do this, you define user events corresponding to Windows events that PowerBuilder does not define.

- ◆ **Multiple methods** 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** You have placed a custom visual user object in a window and need to communicate between the user object and the window.

FOR INFO For more information, see "Communicating between a window and a user object" on page 314.

User events and event IDs

An event ID connects events related to user actions or system activity to a system message. PowerBuilder defines (or maps) events to commonly used event IDs, and when it receives a system message, it uses the mapped event ID to trigger an event. Some events, such as those associated with PowerBuilder activity, are not mapped to an event ID.

Event ID names

The PowerBuilder naming convention for user event IDs is similar to the convention Windows uses to name messages. One or more Windows messages maps to each PowerBuilder event ID.

This Windows message	Maps to this event ID
wm_keydown	pbm_keydown
bm_getcheck (button class message)	pbm_bmgetcheck
bn_clicked (notification)	pbm_bnclicked

PowerBuilder has its own events, each of which has an event ID. For example, the PowerBuilder event DragDrop has the event ID pbm_dragdrop.

FOR INFO See the Microsoft Windows Software Developer's Kit (SDK) documentation for more information about Windows messages.

About custom user events

Obsolete technique

Using custom user events is an obsolete PowerBuilder technique, included for backward compatibility. The recommended technique is using unmapped user events.

FOR INFO For more information about the recommended technique, see "Defining user events" on page 325.

Custom user events are mapped from the Windows `wm_user` message numbers to `pbm_customxx` event IDs. They are not meant to be used with standard controls such as `CommandButtons` and `DropDownListBoxes`. They are meant to be used only with `DataWindow` controls, windows, and user objects other than standard visual user objects (which behave like the built-in controls they inherit from).

Defining custom events for standard controls

Defining custom user events for standard controls is not recommended. To define custom user events for standard controls, you must have some Windows SDK experience and understand the following:

- ◆ All standard controls respond to standard events in the range 0 to 1023. Most controls also define their own range of custom events beyond 1023 (corresponding to `wm_user` messages).
- ◆ Some controls have custom events. You need to know which controls do, because these custom events overlap with the PowerBuilder custom events (`pbm_customxx`).

The `pbm_custom01` event ID maps to `wm_user+0`, `pbm_custom02` maps to `wm_user+1`, and so on, through `pbm_custom75`, which maps to `wm_user+74`.

- ◆ You need to make sure that you use a `pbm_custom` event ID that does not conflict with a custom event defined by Windows for the standard control. Otherwise, you might encounter unexpected behavior in your application.

FOR INFO For more information about Windows messages, see the Microsoft Windows Software Developer's Kit (SDK) documentation.

Defining user events

In PowerBuilder, you can define both mapped and unmapped user events for windows, user objects, and controls.

Mapped user events

When a system message occurs, PowerBuilder triggers any user event that has been mapped to the message and passes the appropriate values to the event script as arguments. When you define a user event and map it to an event ID, you must use the return value and arguments that are associated with the event ID.

FOR INFO For a list of the event IDs defined by PowerBuilder for each PowerBuilder event, see the event's description in the *PowerScript Reference*.

Unmapped user events

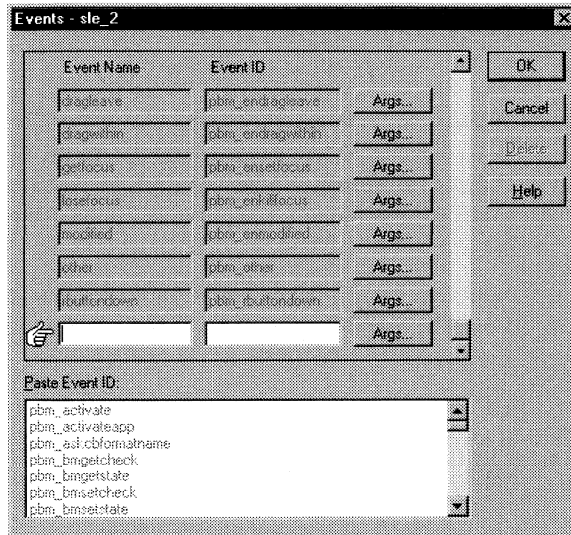
Unmapped user events are associated with a PowerBuilder activity and do not have an event ID. When you define an unmapped user event, you specify the arguments and return data type; only your application scripts can trigger the user event.

FOR INFO For information on using unmapped user events, see *Application Techniques*.

❖ To define a user event:

- 1 Open the Window painter or User Object painter.
- 2 Select the window, user object, or control for which you want to declare the user event.

- 3 Select Declare>User Events from the menu bar.



The name and ID of the PowerBuilder and user events defined for the selected object or control display in the Event Name and Event ID listboxes. The event name and event ID of the predefined PowerBuilder events are protected; they cannot be modified. You can change the event ID for a user event or delete a user event, but you cannot rename a user event.

- 4 Enter the name of the user event you are declaring in the Event Name box at the end of the list.
- 5 Double-click the ID of the event you want to map to your user event in the Paste Event ID listbox.

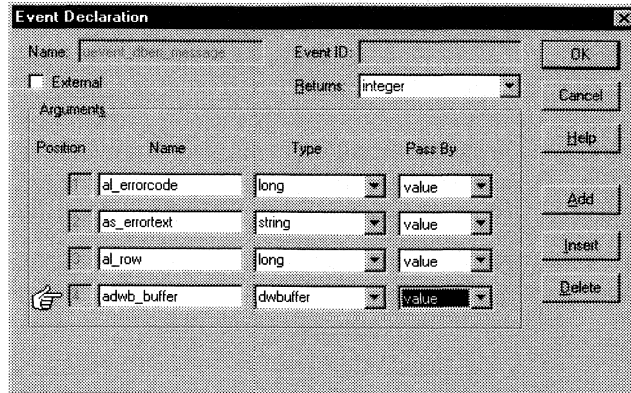
or

Click the Args button if you are defining an unmapped event and it will return a value type other than integer or will take arguments.

When you can specify arguments

If you map the user event to an event ID, you cannot change its return type or specify arguments.

If you click Args, the Event Declaration dialog box displays.



Specify the return type and arguments for the event and click OK when you have finished. The Event Declaration dialog box is similar to the New Function dialog box.

FOR INFO For how to use the dialog box, see "Defining a return type" on page 134 and "Defining arguments" on page 136.

6 Click OK.

FOR INFO See "Using a user event" next for information on using your user event.

Using a user event

Once you define a user event, you must write the script that PowerBuilder will execute when that user event is triggered. If it is an unmapped user event, you also write the code that will trigger the user event.

Writing the script

User events display in alphabetical order in the Select Event listbox in the PowerScript painter, along with the predefined events. As with predefined events, the script tells PowerBuilder what processing to perform when the user event occurs.

Triggering the event

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 INFO For information about calling events, see the *PowerScript Reference*.

Examples of user event scripts

This section includes two examples of using a mapped user event and one example of using an unmapped user event.

FOR INFO For more user event examples, see "Communicating between a window and a user object" on page 314.

Example 1: mapped user event for a control

Situation

You have several `SingleLineEdit` controls in a window and want the `ENTER` key to behave like the `TAB` key (if users press `ENTER`, you want them to tab to the next `SingleLineEdit`).

Solution

Define a user event for each `SingleLineEdit`. Give the event any name you want, such as `CheckKey`. Map the event to the event ID `pbm_keydown`.

Write a script for the user event that tests for the key that was pressed. If `ENTER` was pressed, set the focus to the `SingleLineEdit` that you want the user to go to.

For example, in the script for the user event for `sle_1`, you could code:

```
// Script for user event CheckKey,  
// which is mapped to pbm_keydown.
```

```

IF KeyDown(KeyEnter!) THEN // Go to sle_2
    ifsle_2.SetFocus( ) // Enter pressed.
END IF

```

Similarly, in the script for the user event for sle_2, you could code:

```

// Script for user event CheckKey,
// which is mapped to pbm_keydown.

IF KeyDown(KeyEnter!) THEN // Go to sle_3
    ifsle_3.SetFocus( ) // Enter pressed.
END IF

```

Example 2: mapped user event for an edit style

- Situation** You have a DataWindow control with a column that uses the RadioButton edit style and you want to allow users to scroll through the RadioButtons when they press DOWN ARROW or UP ARROW (normally, pressing DOWN ARROW or UP ARROW scrolls to the next or preceding row).
- Solution** Declare a user event for the DataWindow control that maps to the event ID pbm_dwnkey (dwn stands for *DataWindow notification*) and write a script like the following for it.

```

// Script is in a user event for a DataWindow control.
// It is mapped to pbm_dwnkey. If user is in column
// number 6, which uses the RadioButton edit style,
// and presses DownArrow, the cursor moves to the
// next item in the RadioButton list, instead of
// going to the next row in the DataWindow, which is
// the default behavior. Pressing UpArrow moves to
// preceding RadioButton.
//
// Note that the CHOOSE CASE below tests for data
// values, not display values, for the RadioButtons.

int colnum = 6 //Column number
long rownumrownum = dw_2.GetRow( ) // Current row
IF KeyDown(KeydownArrow!) AND &
This.GetColumn( ) = colnum THEN
    CHOOSE CASE dw_2.GetItemString(rownum, colnum)
        case "P" // First value in RB
            This.SetItem(rownum, colnum,"L") // Next

```

```
        case "L"                // second value in RB
            This.SetItem(rownum, colnum,"A") // Next
        case "A"                // last value in RB
            This.SetItem(rownum, colnum,"P") // First
    END CHOOSE
    This.SetActionCode(1)      // Ignore key press
END IF

// The following code does same thing for UpArrow.

IF KeyDown(KeyupArrow!) AND &
This.GetColumn( ) = colnum THEN
    CHOOSE CASE dw_2.GetItemString(rownum, colnum)
        case "P"                // First value in RB
            This.SetItem(rownum, colnum,"A") // Last
        case "L"                // Another value in RB
            This.SetItem(rownum, colnum,"P")case "A"
                                // Last value in RB
            This.SetItem(rownum, colnum,"L")
    END CHOOSE
    This.SetActionCode(1)
END IF
```

Example 3: unmapped user event for menu options

Situation Suppose you use the same menu in all your windows, but you want to enable or disable some menu items, in this case database update items, depending on which window the user is in.

Solution In the window that will be the ancestor of all the sheet windows that do not have database update capability, define an unmapped user event called `ct_menu_enable`. The event takes a boolean argument, `ab_state`, to set or clear the enabled property on various menus. This is the script for the `ct_menu_enable` user event in the ancestor window:

```
// Enable / Disable Menu Options
im_CurrMenu.m_maint.m_add.enabled = Not ab_state
im_CurrMenu.m_maint.m_delete.enabled = Not ab_state
im_CurrMenu.m_maint.m_undelete.enabled = Not ab_state
im_CurrMenu.m_maint.m_update.enabled = Not ab_state
im_CurrMenu.m_maint.m_close.enabled = ab_state
```

Then, in the script for the Activate event in the ancestor window, call the user event and pass the value TRUE for the boolean variable `ab_state`.

```
this.Event ct_menu_enable ( TRUE )
```

Write a similar script for the Deactivate event with the value FALSE for `ab_state`.

You can use this window as the ancestor of any sheet window in your application that does not have database update capability. When the window is active, the Add, Delete, Undelete, and Update menu items are greyed out. When it is not active, the Close item is greyed out.

For windows that have database update capability, you can create a second ancestor window that inherits from the ancestor window in which you defined `ct_menu_enable`. In the second ancestor window, you can override the `ct_menu_enable` event script so that the appropriate menu options are enabled.

FOR INFO These event scripts are adapted from the CashTrak sample application, included with the PowerBuilder Application Gallery.

Working with Databases

This part describes how to use PowerBuilder to manage your database, how to build DataWindow objects to retrieve, present, and manipulate data in your applications, and how to use the Data Pipeline painter to copy data from one database to another.

Managing the Database

About this chapter

Central to your PowerBuilder applications is the database. This chapter describes how you can manage the database from within PowerBuilder.

Contents

Topic	Page
About databases	336
About managing databases	339
Using the Database painter	342
Creating and deleting a SQL Anywhere database	351
Working with tables in the Database painter	353
Working with tables in the Table painter	360
Working with indexes	381
Working with keys	374
Working with views	384
Exporting table or view syntax	389
Manipulating data	390
Administering the database	398

Before you begin

You work with relational databases in PowerBuilder. If you are not familiar with relational databases, you may want to consult an introductory text, such as *A Database Primer* by C. J. Date.

About databases

A database is an electronic storage place for data. Databases are designed to ensure that data is valid and consistent, and that it can be accessed, modified, and shared.

A **database management system (DBMS)** governs the activities of a database and enforces rules that ensure data integrity. A *relational* DBMS stores and organizes data in tables.

How you work with databases in PowerBuilder

You can use PowerBuilder to work with the following database components:

- ◆ Tables and columns
- ◆ Keys
- ◆ Indexes
- ◆ Views
- ◆ Extended attributes

About 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 Powersoft Demo Database that is included with PowerBuilder, the Department table has a Dept_id column, and the Employee table also has a Dept_id column that identifies the department in which the employee works. When you work with the Department table and the Employee table, the relationship between them is specified by a **join** of the two tables.

About keys

Relational databases use keys to ensure database integrity.

Primary keys

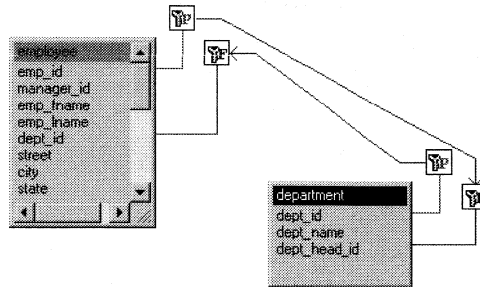
A **primary key** is a column or set of columns that uniquely identifies each row in a table. For example, two employees may have the same first and last names, but they have unique ID numbers. The Emp_id column in the Employee table is the primary key column.

Foreign keys

A **foreign key** is a column or set of columns that contains primary key values from another table. For example, the Dept_id column is the primary key column in the Department table, and a foreign key in the Employee table.

Key icons

In PowerBuilder, columns defined as keys are displayed with **key icons** that include a P for private or F for foreign. PowerBuilder automatically joins tables that have a primary/foreign key relationship, with the join on the key columns:



FOR INFO For more information, see "Working with keys" on page 374.

About indexes

An index is a column or set of columns you identify to improve database performance when searching for data specified by the index. You index a column that contains information you will need frequently. Primary and foreign keys are special examples of indexes.

You specify a column or set of columns with unique values as a **unique index**, represented by an icon with a single key.

You specify a column or set of columns that has values that are not unique as a **duplicate index**, represented by an icon with two keys.

FOR INFO For more information, see "Working with indexes" on page 381.

About views

If you often select data from the same tables and columns, you can create a **view** of the tables. You give the view a name, and each time you refer to it the associated SELECT command executes to find the data.

Views display in the Select Tables dialog box in the Database painter, but a 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 recreated whenever the definition is used.

Database administrators often create views for security purposes. For example, a view of an Employee table that is available to users who are not in Human Resources might show all columns except Salary.

FOR INFO For more information, see "Working with views" on page 384.

About extended attributes

Extended attributes enable you to store information about a table's columns in special system tables called the **Powersoft repository**. Unlike tables, keys, indexes, and views, which are DBMS-specific, extended attributes are Powersoft-specific. The most powerful extended attributes determine the edit style, display format, and validation rules for the column.

FOR INFO For more information about extended attributes, see "Specifying extended attributes" on page 364. For more information about the Powersoft repository, see the Appendix, "The Powersoft Repository".

About managing databases

PowerBuilder supports many database management systems (DBMSs). For the most part, you work in the same way in PowerBuilder for each DBMS. But because each DBMS provides some unique features (which PowerBuilder makes use of), there are some issues that are specific to a particular DBMS.

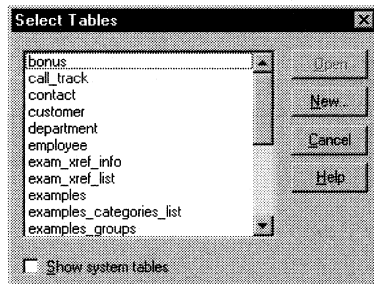
FOR INFO For complete information about using your DBMS, see *Connecting to Your Database*.

How you work with the database

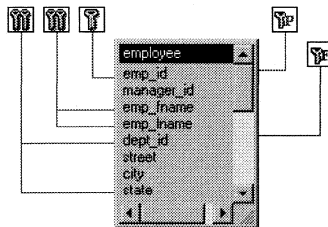
You do most of your database work using the Database painter and other painters you can launch from the Database painter.

How the Database painter works

When you open the Database painter, PowerBuilder displays the names of tables and views to which you have access in the current database:



You work with that database until you connect to another database. When you select tables and click Open, the Database painter graphically displays tables in the database. Indexes and keys display as icons:



In the Database painter, you can:

- ◆ Graphically display tables in the database
- ◆ Open and close tables, including tables related to a selected table
- ◆ Drop tables (remove tables from the database)
- ◆ Modify table and column properties
- ◆ Create, alter, and drop primary keys, foreign keys, and indexes
- ◆ Define and modify extended attributes for columns
- ◆ Create and drop views

Creating and deleting databases

When you are connected to SQL Anywhere, you can also create a new database or delete an existing database using the Database painter.

For all other DBMSs, creating and deleting a database is an administrative task that you cannot do within PowerBuilder.

Related painters

From the Database painter, you can also open five related painters:

- ◆ The *Table painter*, where you create and alter tables
- ◆ The *Data Manipulation painter*, where you retrieve and manipulate data from the database
- ◆ The *Database Administration painter*, where you control access to the database and execute SQL directly
- ◆ The *View painter*, where you create views
- ◆ The *Data Pipeline painter*, where you create pipelines

The following table summarizes what you can do in each painter:

Painter	What you can do
Database	Graphically display tables Open, close, and drop tables Modify table properties Modify column properties Create and drop keys and indexes Open, close, and drop views Define and modify extended attributes

Painter	What you can do
Table	Create new tables Modify table properties Create keys and indexes Define and modify extended attributes
Database Manipulation	Retrieve rows Insert, update, and delete rows Save data
Database Administration	Maintain users Define security Execute SQL Create stored procedures Create triggers Analyze performance
Data Pipeline	Create a data pipeline definition using a selected table Execute the pipeline
View	Create views

The Table painter, Data Pipeline painter, and Database Administration painter can all be accessed from the Database painter. But they are also standalone painters that can be opened by clicking a button in the PowerBar.

You can access the Data Manipulation painter and the View painter only from the Database painter.

Using the Database painter

❖ **To open the Database painter:**

- 1 Click the Database button in the PowerBar.

The Select Tables dialog box displays and lists the names of all tables and views in the current database.

Suppressing the table list

If you clear the Display Table List checkbox in the Database Preferences property sheet, PowerBuilder does not display the table list when you open the Database painter. You can then use menu items as needed and click the Open button in the PainterBar to open the Select Tables dialog box.

FOR INFO For information about changing Database preferences, see "Modifying database preferences" on page 347.

- 2 Select one or more tables and click Open to display them graphically.

or

Click New to create a new table in the Table painter.

or

Click Cancel to go to the painter workspace with no tables displayed.

FOR INFO For more information, see "Working with tables in the Database painter" on page 353.

Changing the database connection

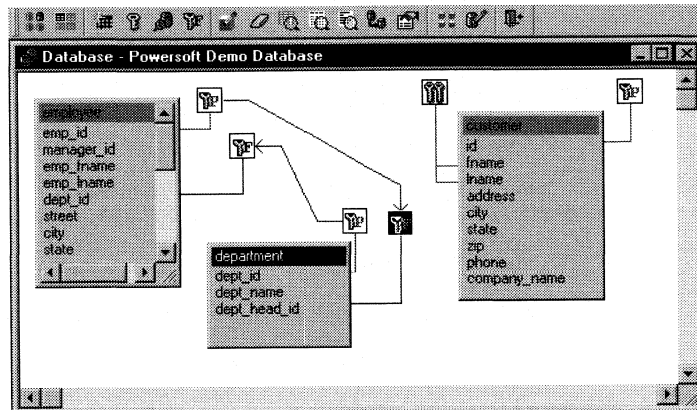
When you open a painter that communicates with the database (such as the Database painter or DataWindow painter), PowerBuilder connects you to the database you used last if you are not already connected. You can change to a different database anytime.

FOR INFO For more about changing the database you are connected to, see *Connecting to Your Database*.

About the painter

Like the other PowerBuilder painters, the Database painter contains a menu bar, a customizable PainterBar, and a workspace.

To work with database components, you open them in the workspace. Three Powersoft Demo Database tables are open in the workspace shown below: Employee, Department, and Customer:



PowerBuilder displays table columns and icons that mark a column or a set of columns as a primary key, a foreign key, or an index. Keys and indexes were defined for the tables earlier—by you, another user, or your database administrator.

Number of columns that display in tables

PowerBuilder displays eight columns in tables in the Database painter and adds a scrollbar for larger tables. You can change the number of columns that display by setting the Columns in Table Display property in the Database Preferences property sheet.

FOR INFO For information about changing Database preferences, see "Modifying database preferences" on page 347.

The menu bar and PainterBar

You can perform most common activities in a database from the Database painter's menu bar. The PainterBar provides buttons for performing all the activities listed in the following table, as well as two additional buttons for defining a database profile and opening the Database Administration painter. Use PowerTips to identify the buttons.

Menu item	Activity
Object>Select Tables	Open an existing table in the Database painter workspace or create a new table in the Table painter
Object>New>Table	Create a new table in the Table painter
Object>New>Index	Create an index for the table that's selected in the Database painter workspace
Object>New>View	Create a view in the current database
Object>New>Foreign Key	Create a foreign key for the table that's selected in the Database painter workspace
Object>Edit Object	Alter the table that's selected in the Database painter workspace using the Table painter
Object>Drop	Drop the table that's selected in the Database painter workspace
Object>Edit Data>Grid	Manipulate data in the table that's selected in the Database painter workspace using the grid format of the Data Manipulation painter
Object>Edit Data>Table	Manipulate data in the table that's selected in the Database painter workspace using the table format of the Data Manipulation painter
Object>Edit Data>Freeform	Manipulate data in the table that's selected in the Database painter workspace using the freeform format of the Data Manipulation painter
Object>Data Pipeline	Pipe data in the table that's selected in the Database painter workspace
Object>Properties	Modify the properties of the table that's selected in the Database painter workspace
Design>Arrange Tables	Arrange the tables in the Database painter workspace
File>Exit	Close the Database painter

How the Database painter and the Table painter work together

The Database painter and the Table painter are independent painters, but when they're open at the same time they work together. For example, when you create and save a table definition in the Table painter, the Table painter notifies the Database painter that a new table exists and the new table displays in all instances of the Database painter. If you add an index for a table in the Database painter, you can see the change if you open the Table Properties property sheet for that table in the Table painter.

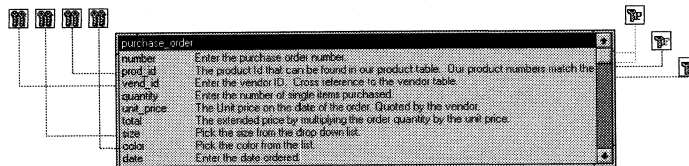
Working with objects in the workspace

Moving objects You can move an object around the workspace by dragging it with the mouse. For example, to move a table, press the left mouse button on the title bar for the table and drag the table.

Resizing objects You can resize an object in the workspace by dragging one of its corners.

Displaying comments, indexes, and keys When a table is created or modified in PowerBuilder, you can assign comments to the table and to each of its columns. These comments can be useful to others who will be using the tables. You can choose to display comments for tables displayed in the workspace by selecting Show Comments from the popup menu for the workspace.

Similarly, you can choose whether to display indexes and, if your DBMS supports them, primary and foreign keys:



Using a table or column popup menu A table open in the workspace has a table name at the top and column names below. In the illustration above, the table name is purchase_order.

Table popup menu

When you point at the table name and press the right mouse button, you display the popup menu for the table.

On Macintosh

On the Macintosh, hold down the CONTROL key and press the mouse button to display popup menus.

From the popup menu, you can select menu items to work with the table in the Database painter or in one of the related painters:

Select this	To do this
Close	Close the table
Alter Table	Open the Table painter so that you can modify the table
Properties	Open the property sheet for the table
New	Create a new index or foreign key
Drop Table	Drop the table
Edit Data	Open the Data Manipulation painter so that you can modify the data
Data Pipeline	Open the Data Pipeline painter so that you can create a data pipeline definition with the table as the source
Print Definition	Print the table definition

FOR INFO For information about modifying table properties, see "Modifying table properties in the Database painter" on page 355. The Table painter and Data Manipulation painter are described later in this chapter. The Data Pipeline painter is described in Chapter 24, "Working with Data Pipelines".

Column popup menu

When you point at a column name and press the right mouse button, you display the popup menu for the column.

On Macintosh

On the Macintosh, hold down the CONTROL key and press the mouse button to display popup menus.

Select Definition to display and modify the table definition in the Table painter, and select Properties to display the column properties and modify them.

FOR INFO For information about using the Table painter, see "Working with tables in the Table painter" on page 360. For information about modifying column properties, see "Modifying column properties in the Database painter" on page 356.

Keyboard alternatives

PowerBuilder provides keyboard alternatives to common workspace activities:

To	Do this
Select a table or view	Press TAB to move left to right among the opened tables and views, or press SHIFT+TAB to move right to left
Display the Table painter	Select the table for which you want to display the information, then press ENTER
Browse an index or key	Press TAB or SHIFT+TAB to select the index or key to browse and then press ENTER
Scroll in the painter workspace	If you open more tables and views than can be displayed in the Database painter workspace at one time, you can scroll up or down to view all the tables and views To scroll up, press UP ARROW . To scroll down, press DOWN ARROW

Modifying database preferences

To modify database preferences, select **Design>Options** from the menu bar. Some of the preferences are specific to database connection. Other preferences are specific to the Database painter.

Preferences on the General property page

The Shared Database Profiles, Use Powersoft Repository, Read Only, and Keep Connection Open preferences are database-connection specific preferences.

FOR INFO For information about modifying these preferences, see *Connecting to Your Database*.

The remaining preferences are specific to the Database painter.

Database preference	What PowerBuilder does with the specified preference
Display Table List	PowerBuilder displays the table list when you open the Database painter unless you clear the checkbox
SQL Terminator Character	PowerBuilder uses the semicolon as the SQL statement terminator unless you enter a different terminator character in the textbox

Database preference	What PowerBuilder does with the specified preference
Refresh Table List	When PowerBuilder first displays a table list, PowerBuilder retrieves the table list from the database and displays it. To save time, PowerBuilder saves this list internally to use again so very large table lists don't have to be regenerated. The table list is refreshed every 30 minutes (1800 seconds) unless you specify a different refresh rate
Columns in the Table List	When PowerBuilder displays tables graphically, eight table columns display unless you change the number of columns

Preferences on the Color 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, and foreign keys. Set a color preference by selecting a color from a dropdown listbox.

You can design custom colors that you can use when you select color preferences. To design custom colors, select Design>Custom Colors from the menu bar and work in the Custom Colors dialog box.

Logging your work

As you work with your database, you will probably generate SQL statements. For example, as you define a new table in the Table painter, PowerBuilder is building a SQL CREATE TABLE statement internally. When you click the Create button, PowerBuilder sends the SQL statement to the DBMS to create the table. Similarly, when you add an index, PowerBuilder is building a CREATE INDEX statement.

You can record all SQL generated in a Database painter session in a log file. This allows you to have a record of your work and also makes it easy to duplicate the work if you need to create the same or similar tables in another database.

Logging work in the Table painter

When you create or alter a table, you are working in the Table painter and the SQL generated is *not* recorded in the Database painter log file. If you want to change the extended attributes for a table and you want the generated SQL to be added to the Database painter log file, use the table's property sheet in the Database painter to make the changes.

FOR INFO For how to create a log file in the Table painter, see "Logging applied SQL syntax changes" on page 372.

❖ To start logging your work in a text file:

- 1 Open the Database painter.
- 2 Select Design>Start Log from the menu bar.

PowerBuilder opens a log file and displays the Activity Log icon at the bottom of the screen.

From now on, all your work will be saved in a temporary file.

❖ To view the log:

- ◆ Double-click the Activity Log icon.

PowerBuilder opens a window showing the log. You can leave the window open as you work.

❖ To save the log to a permanent text file:

- 1 Select Design>Save Log As from the menu bar.
The Save the Log dialog box displays.
- 2 Name the file and click OK. The default file extension is SQL, but you can change that if you want.

PowerBuilder saves the log as a text file with the specified name.

Submitting the log to your DBMS

You can open a saved log file later and submit it to your DBMS in the Database Administration painter.

FOR INFO For more information, see "Building and executing SQL statements" on page 399.

❖ **To clear the log:**

- ◆ Select Design>Clear Log from the menu bar.

PowerBuilder deletes all statements in the log, but leaves the log open.

❖ **To stop the log:**

- ◆ Select Design>Stop Log from the menu bar.

or

Close the Activity Log window.

PowerBuilder closes the log. Your work will no longer be logged. However, the log file still exists. You can open it again and continue logging from where you left off.

Creating and deleting a SQL Anywhere database

In PowerBuilder you work within an existing database. With one exception, creating or deleting a database is an administrative task that is not performed directly in PowerBuilder. The one exception is that you can create and delete a local SQL Anywhere database from within PowerBuilder.

FOR INFO For information about creating and deleting databases, see your DBMS documentation.

On UNIX

PowerBuilder does not support SQL Anywhere on UNIX systems.

❖ To create a local SQL Anywhere database:

- 1 Open the Database painter.
- 2 Select File>Create Database from the menu bar.

The Create Local Database dialog box displays.

- 3 In the Database Name box, specify the filename and path of the database you are creating.

If you do not provide a path, the database will be created in the current directory or folder. If you do not provide a file extension, the database filename will be given the extension DB.

- 4 Define other properties of the database as needed. Click the More button to define additional properties.

If you are using a non-English database, you can specify a code page in the Collation Sequence box.

FOR INFO For complete information about filling in the dialog box, click the Help button in the dialog box.

- 5 Click OK.

What happens

When you click OK, PowerBuilder does the following:

- ◆ Creates a database with the specified name in the specified directory or folder. If a database with the same name exists, you are asked whether you want to replace it.
 - ◆ Adds a data source to the ODBC preferences file (ODBC.INI on Windows). 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 PowerBuilder initialization file. The profile has the same name as the database unless one with the same name already exists, in which case a suffix is appended.
 - ◆ Connects to the new database.
- ❖ **To delete a local SQL Anywhere database:**
- 1 Open the Database painter.
 - 2 Select File>Delete Database from the menu bar.
 - 3 Select the database you want to delete.
You are prompted to confirm your action.
 - 4 Click Yes to delete the database.

What happens

When you click Yes, PowerBuilder:

- ◆ Deletes the specified database
- ◆ Removes the data source from the ODBC preferences file
- ◆ Deletes the database profile from the PowerBuilder initialization file

Working with tables in the Database painter

When you open the Database painter, the Select Tables dialog box displays listing all the tables and views in the current database that you have access to (including tables and views that were not created using PowerBuilder). You can create a new table or open existing tables.

Opening a table

❖ **To open a database table:**

- 1 Click the Open button in the PainterBar.

or

Select Object>Select Tables from the menu bar.

The Select Tables dialog box displays.

- 2 (Optional) To display system tables, select the Show System Tables checkbox.

FOR INFO For information about system tables, see "About system tables" on page 354.

PowerBuilder caches the table list

The first time PowerBuilder has to build a list of tables in the database, it retrieves the list of tables from the database and displays it.

PowerBuilder saves this list internally and uses it the next time a list of tables needs to be displayed; this saves time, because PowerBuilder doesn't have to regenerate what could be a very large list.

By default, the list is refreshed every 30 minutes (1800 seconds). You can specify a different refresh rate by setting the Refresh Table List property in the Database Preferences property sheet.

FOR INFO For information about changing Database preferences, see "Modifying database preferences" on page 347.

- 3 Select the tables to open by doing one of the following:
 - ◆ Click the name of each table you want to open in the list displayed in the Select Tables dialog box, then click Open to open the selected tables.

- ◆ Double-click the name of each table you want to open. Each table is opened immediately. Then click Cancel to close the Select Tables dialog box.

Representations of the selected tables display in the Database painter workspace.

About system tables

By default, PowerBuilder shows only user-created tables in the Select Tables dialog box. If you select Show System Tables in the dialog box, PowerBuilder also shows system tables.

Two kinds of system tables exist in the database:

- ◆ System tables provided by your DBMS (for more information, see your DBMS documentation)
- ◆ PowerBuilder system tables

About PowerBuilder system tables

PowerBuilder stores extended attribute information you provide when you create or modify a table (such as the text to use for labels and headings for the columns, validation rules, display formats, and edit styles) in the Powersoft repository.

For example, in the Employee table, one column name is Emp_lname. A label and a heading for the column is defined for PowerBuilder to use in DataWindow objects. The column label is defined as Last Name:. The column heading is defined as Last Name. The label and heading are stored in the PBCatCol table in the Powersoft repository.

About the Powersoft repository

The Powersoft repository system tables are maintained by PowerBuilder. The repository contains information about database tables and columns. Repository information extends database definitions. Only PowerBuilder users can enter information into the repository.

The Powersoft repository has five system tables, each of which stores Powersoft-specific information about extended attributes:

This system table	Stores this Powersoft-specific 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

This system table	Stores this Powersoft-specific information
PBCatFmt	Display format names and definitions
PBCatTbl	Table data such as name, fonts, and comments
PBCatVld	Validation rule names and definitions

You can open system tables in the Database painter just like other tables.

FOR INFO For more about the tables in the Powersoft repository, see the Appendix, "The Powersoft Repository".

Modifying properties in the Database painter

You can modify the properties of any table in the workspace, 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, and a comment you can associate with the column.

Modifying table properties in the Database painter

❖ To modify the properties of the table selected in the Database painter:

- 1 In the Database painter, click the Table Properties button in the PainterBar.
The Table property sheet displays.
- 2 Select a tab and specify properties.

The Table property sheet is DBMS-dependent

If you are connected to a text-based ODBC data source, only the General and Font property pages display. The Primary Key property page does not display.

Select this tab	To modify this property
General	Comments associated with the table
Data Font	Font for data displayed in the Data Manipulation painter
Heading Font	Font for headers displayed in the Data manipulation painter

Select this tab	To modify this property
Label Font	Font for labels displayed in the Freeform format of the Data Manipulation painter
Primary Key	The table's primary key

FOR INFO For information about modifying primary keys, see "Defining primary keys" on page 376.

Setting table properties in the Database or Table painter

The properties that you can set on the Table property sheet in the Database painter are the same as those you can set in the Table painter. However, the Table Properties property sheet in the Table painter has two additional tab pages, for foreign keys and indexes.

FOR INFO For more information, see "Modifying table properties in the Table painter" on page 366.

Modifying column properties in the Database painter

❖ To modify the column properties of a table in the Database painter:

- 1 Display the column's popup menu and select Properties.

The Column property sheet displays.

- 2 Select a tab and specify properties:

Select this tab	To modify this property
General	Comments associated with the column
Headers	The header text used in the Data Manipulation painter, the label text used in the freeform format of the Data Manipulation painter, and their positions
Display	Display format for the column and its justification, display height and width, and for character columns, the case and whether the column is a picture column
Validation	Validation rule for the column
Edit Style	Edit style for the column

Setting column properties in the Database or Table painter

Most of the properties that you can set on the property sheet for a column correspond to the extended attributes you can set when creating or altering a table in the Table painter. However, there are some properties you can set in the Column property sheet that you cannot set in the Table painter.

For example, in the Table painter you have to select an existing display format, edit style, and validation rule, but in the Database painter you can specify new ones.

FOR INFO For more information about the column properties you can set in the Table painter, see "Specifying extended attributes" on page 364. For information about working with display formats, edit styles, and validation rules, see Chapter 16, "Displaying and Validating Data".

Specifying additional properties for character columns

You can also set two additional properties for character columns on the Display property page: Case and Picture.

Specifying the displayed case

You can specify whether PowerBuilder converts the case of characters for a column in a report or form.

❖ **To specify how character data should be displayed:**

- ◆ On the Display property page, select a value in the Case dropdown listbox:

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 or, on supported platforms, WMF or PICT files).

❖ **To specify that column values are names of picture files:**

- 1 On the Display property page, select the Picture checkbox.

When the Picture checkbox is selected, PowerBuilder expects to find bitmap (BMP), Windows metafile (WMF), or Macintosh PICT filenames in the column and displays the contents of the picture file—not the name of the file—in reports and forms.

Because PowerBuilder cannot determine the size of the image until execution time, it sets both display height and display width to 0 when you check the Picture checkbox.

- 2 Enter the size and optionally the justification for the picture.

Closing a table

You can remove a table from the workspace by selecting Close from its popup menu. This action only removes the table from the Database painter workspace. It does not drop (remove) the table from the database.

Dropping a table

You can drop a table in the Database painter workspace. Dropping removes the table from the database.

❖ **To drop a table:**

- 1 Click the Drop button.

or

Select Drop Table from the table's popup menu.

or

Select Object>Drop from the menu bar.

PowerBuilder prompts you to confirm the deletion.

- 2 Click Yes.

Deleting orphaned table information

If you drop a table outside PowerBuilder, information will remain in the repository about the table, including extended attributes for the columns.

❖ **To delete orphaned table information from the repository:**

- 1 Select Design>Synchronize PB Attributes from the menu bar.

PowerBuilder prompts you to confirm your action.

- 2 Click Yes.

PowerBuilder deletes information about the table's attributes from the repository.

If you try to delete orphaned table information and there is none, a message tells you that synchronization is not necessary.

Working with tables in the Table painter

In the Table painter, you can create a new table definition or alter an existing table definition. You can also modify table properties and work with indexes and keys.

About the Table painter

Although the Table painter is not an editor, it has some features that are similar to an editor. Instead of working with text, you work with table columns. For example, you can copy a column and paste it in the current table definition or in a different table definition.

Working with more than one table

Each time you open the Table painter, you can select one table definition to change or create one new table definition. If you want to work on more than one table at a time, open a new instance of the Table painter for each table.

Two views in the Table painter

As you work in the Table painter, most of the time you'll be working in the workspace creating or altering a table definition. But you can leave the workspace and enter SQL Syntax view to see pending changes (in SQL syntax) to the table definition as you work.

FOR INFO For more information, see "Working in SQL Syntax view" on page 371.

How the Table painter and the Database painter work together

The Table painter and the Database painter are independent painters, but when they're open at the same time, they work together.

For example, when you create and save a table definition in the Table painter, the Table painter notifies the Database painter that a new table exists and the new table displays in all instances of the Database painter. If you change a column property in the Table painter—say you add an index—you can see the change in the Database painter.

Creating a new table from scratch in the Table painter

You can create a new table in PowerBuilder in the current database. The current database is the database to which PowerBuilder is connected.

❖ **To create a table in the current database:**

- 1 Click the Table button in the PowerBar.

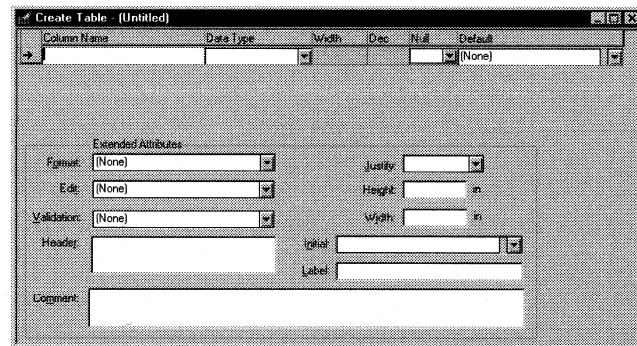
The Open Table dialog box displays.

Displaying system tables

You can open system tables (including the five tables in the Powersoft repository) in the Table painter. To see system tables in the Open Table list, select the Show System Tables checkbox. Don't change the definition of system tables. To guard against doing this, specify them as read-only by selecting the Read Only checkbox.

- 2 Click New.

The Table painter workspace displays with the insertion point in the Column Name box for the first column. The workspace is a template for specifying each column in the table. What you see in the workspace is DBMS-dependent:



- 3 Enter required information for the first column.

FOR INFO For what to enter in each field, see "Specifying the column definition" on page 363.

As you enter information, you can move from place to place in the column definition by pressing the TAB key. After defining the last item in the column definition (in this case, for SQL Anywhere, the Default definition), press the TAB key to display the work area for the next column.

- 4 (Optional) Specify extended attributes for the column.

You can do this now or later when you modify the table.

FOR INFO For what to enter in each field, see "Specifying extended attributes" on page 364.

- 5 Repeat steps 3 and 4 for each additional column in your table.
- 6 (Optional) Click the SQL Syntax button to see pending SQL syntax.
The SQL syntax will be submitted to the database when you save or close the table. When you click the SQL Syntax button, you will be asked to enter a name for the table in the Create New Table dialog box.
To return from SQL Syntax view to the Table painter workspace, click the SQL Syntax button again.
- 7 Click the Save button.
If you did not do step 6, you will be asked to name the table. PowerBuilder submits the pending SQL syntax statements it generated to the DBMS, and the table is created. Then control is returned to the Table painter with the new table open.
- 8 Click the Close button to close the table.
If you have made no changes since you saved the table, the table closes. Otherwise, you are prompted to save the table.
If you click Yes, PowerBuilder applies all changes that are pending, and the Table painter closes.

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 look at SQL Syntax view again. When you click Save again, PowerBuilder submits a DROP TABLE statement to the DBMS, recreates the table, and applies all changes that are pending.

Clicking Save many times can be costly when you are working with large tables. So waiting to save a table until you have finished defining it can be a good practice.

Creating a new table from an existing table in the Table painter

Because you can open a table as read-only, you can create a new table that's similar to an existing table very quickly by using the existing table in read-only mode as a template for the new table.

❖ To create a new table using an existing table as a template:

- 1 Click the Table button in the PowerBar.
The Open Table dialog box displays.

- 2 Select the table you want to use as a template, select the Read Only checkbox, and then click Open.

The Table painter workspace displays the read-only table.

- 3 Make whatever changes you want to the table definition.
- 4 Click the Properties button to display the Table Properties property sheet.
- 5 On the General property page, change the table name and add comments to document what you've done.

FOR INFO For more information about modifying table properties, see "Modifying table properties in the Table painter" on page 366.

- 6 In the property sheet, click Apply to apply the changes you made to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.

or

Click OK to apply the request for changes and close the property sheet.

- 7 Click Save to save the table and then click Close to close the Table painter.

Specifying column definitions and extended attributes

When you create a new table, you must specify a definition for the column. You can specify extended attributes for the column when you define it or you can do it later in the Table painter or the Database painter.

Specifying the column definition

The fields that display for each column at the top of the Table painter depend on your DBMS. You may not see all of the following fields, and the values that you can enter are dependent on the DBMS.

FOR INFO For more information, see your DBMS documentation.

Field	What you enter
Column name	(Required) The name by which the column will be identified
Data Type	(Required) Select a data type from the dropdown listbox. All data types supported by the current DBMS are displayed in the listbox

Field	What you enter
Width	For data types with variable widths, the number of characters in the field
Dec	For numeric data types, the number of decimal places to display
Null	Select Yes or No from the Null dropdown listbox 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 dropdown listbox has built-in choices, but you can type any other value. For an explanation of the built-in choices, see your DBMS documentation

Specifying extended attributes

In addition to providing the column information that is required to create a table, you can specify extended attributes for each column. An extended attribute is PowerBuilder-specific information that enhances the definition of the column.

❖ **To specify extended attributes in the Table painter workspace:**

- 1 Display or create the table definition in the workspace.
- 2 Select the column to modify and specify the extended attributes.

The Extended Attributes group box at the bottom of the workspace is the area used for specifying the extended attributes. The following table shows the extended attributes that you can set in the Table painter:

Extended attribute	Meaning
Format	How the data is formatted in a DataWindow object. 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
Justify, Height, and Width	How data is aligned and how much space to allocate to the data in a DataWindow object
Edit	How the column is presented in a DataWindow object. For example, you can display column values as radio buttons or in a dropdown listbox

Extended attribute	Meaning
Validation	Criteria that a value must pass to be accepted in a DataWindow object. For example, you can associate a validation rule with a Salary column so that you can only enter a value within a particular range
Initial	The initial value for the column. You can select a value from the list or type in a value. The initial value must be the same data type as the column, must pass validation, and can be NULL only if NULL is allowed for the column
Header	Header text used in tabular, grid, or n-up DataWindow objects
Label	Label text used in freeform DataWindow objects
Comment	Description of the column. Whenever the table is opened in the Database painter, you can display column comments to understand the purpose of the columns

- 3 Click Save to save your changes.

Specifying extended attributes in the Database painter

You can also specify extended attributes on the property sheet for the column in the Database painter.

FOR INFO For more information, see "Modifying column properties in the Database painter" on page 356.

Overriding definitions

In the DataWindow painter, you can override the extended attributes specified in the Database or Table painter for a particular DataWindow object.

How the information is stored

Extended attributes are stored in the Powersoft repository (in the PowerBuilder system tables in the database). PowerBuilder uses the information to display, present, and validate data in the Database painter, the Data Manipulation 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 Table painter, you can select predefined display formats, edit styles, and validation rules from dropdown listboxes.

In the Database painter, you can create and name your own display formats, edit styles, and validation rules.

FOR INFO For more information about these extended attributes, see Chapter 16, "Displaying and Validating Data".

About headings and labels

By default, PowerBuilder uses the column names as labels and headings, replacing any underscore characters with spaces and capping each word in the name. For example, the default label or heading for the column Dept_name is Dept Name. To define multiple-line headings, press CTRL+ENTER to begin a new line.

On Macintosh

On the Macintosh, use COMMAND+RETURN to define multiple-line headings.

Modifying table properties in the Table painter

❖ **To modify the properties of the table selected in the Database painter:**

- 1 In the Table painter, click the Properties button in the PainterBar. The Table Properties property sheet displays.
- 2 Select a tab and specify properties.

The Table property sheet is DBMS-dependent

If you are connected to a text-based ODBC data source, only the General and Font property pages display. The Indexes and Key property pages do not display.

Select this tab	To modify this property
General	Comments associated with the table
Data Font	Font for data displayed in the Data Manipulation painter
Heading Font	Font for headers displayed in the Data manipulation painter
Label Font	Font for labels displayed in the freeform format of the Data Manipulation painter

Select this tab	To modify this property
Primary Key	The table's primary key
Foreign Keys	The table's foreign keys
Indexes	The table's indexes

3 Click Apply.

Any changes you've made in the property sheet are applied to the pending SQL Syntax changes that PowerBuilder generates, not to the table definition.

You can view these pending changes by clicking the SQL Syntax button in the PainterBar. If you then click the Cancel button, the property sheet closes, but the specified pending changes already made are still pending.

4 Click Save.

The SQL executes. There are now no pending changes until you make additional changes.

Specifying table properties in the Database painter

You can also specify table properties on the property sheet for the table in the Database painter.

FOR INFO For more information, see "Modifying table properties in the Database painter" on page 355.

Altering a table in the Table painter

After a table is created, what you can do to alter the table depends on your DBMS.

In an existing table, you can always:

- ◆ Add or modify PowerBuilder-specific extended attributes for columns
- ◆ Delete an index and create a new index

In an existing table, you can never:

- ◆ Insert a column between two existing columns
- ◆ Prohibit NULL values for an appended column
- ◆ Alter an existing index

In an existing table, some DBMSs let you do the following, but others do not:

- ◆ Append columns that allow NULLs
- ◆ Increase or decrease the number of characters allowed for data in an existing column
- ◆ Allow NULLs
- ◆ Prohibit NULLs in a column that allowed NULLs

Table painter is DBMS-dependent

The Table painter knows your DBMS and grays out actions or notifies you about actions that your DBMS prohibits.

FOR INFO If you need complete information about what you can and cannot do when you modify a table in your DBMS, see your DBMS documentation.

❖ **To alter a table:**

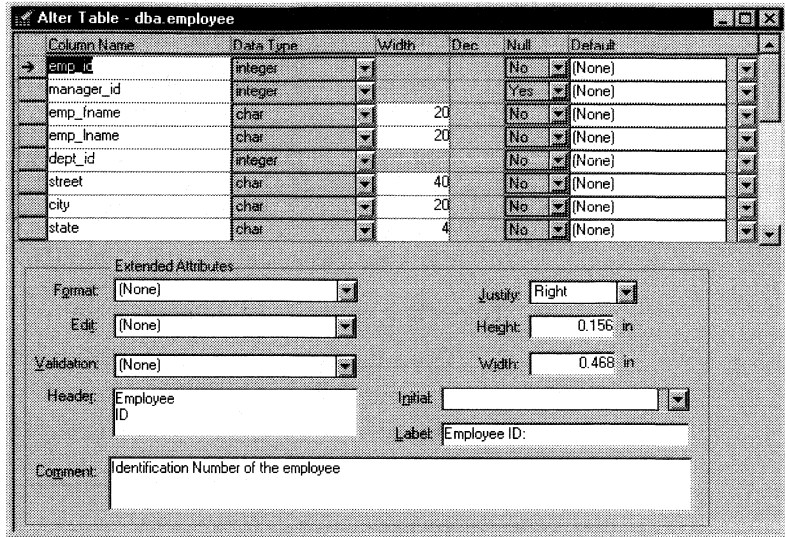
- 1 Open the Table painter and, in the Open Table dialog box, select the table you want to alter.

Opening multiple instances of the Table painter

In the Open Table dialog box, you can select only one table to open. However, you can click the Table button in the PowerBar at any time, select another table, and open another instance of the Table painter. Doing this is helpful when you want to use the Table painter's cut, copy, and paste features to cut or copy and paste between tables.

- Click Open.

The table definition displays in the Table painter workspace. This screen shows the Employee table:



- Make the changes you want in the workspace or in the Table Properties property sheet.

FOR INFO For information about settings you can change in the Table painter, see "Specifying extended attributes" on page 364. For information about the Table Properties property sheet, see "Modifying table properties in the Table painter" on page 366.

- Click the Save button.

PowerBuilder submits the pending SQL syntax statements it generated to the DBMS, and the table is modified.

- Click the Close button.

Cutting, copying, and pasting columns in the Table painter

In the Table painter, you can use the Cut, Copy, and Paste buttons in the PainterBar (or Edit>Cut Column, Edit>Copy Column, Edit>Paste Column in the menu bar) to cut, copy, and paste one column at a time within a table or between multiple 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 the table you want to paste a column to is 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'll get an error message. You can only append a column to an existing table.

- 2 Click the Paste button in the PainterBar.

❖ **To paste a column to a different table:**

- 1 Open another instance of the Table painter and open a 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.

Specifying fonts for the table in the Table painter

When you create or alter a table, you can choose the fonts that will be used to display information from the table in a DataWindow object.

❖ **To specify fonts for a table:**

- 1 In the Table painter workspace, click the Properties button.
- 2 On the Data Font, Heading Font, and Label Font property pages, specify the fonts as needed.

You can specify the font, point size, color, and style for the following:

- ◆ **Data** The values retrieved from the database
- ◆ **Headings** The column identifiers used in grid, tabular, and n-up DataWindow objects
- ◆ **Labels** The column identifiers used in freeform DataWindow objects

- 3 In the property sheet, click Apply to apply the changes you made to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.

or

Click OK to apply the request for changes and close the property sheet.

Working in SQL Syntax view

Viewing pending SQL changes

As you create or alter a table definition in the Table painter, you can view the pending SQL syntax changes that will be made when you save the table definition.

❖ To view pending SQL syntax changes:

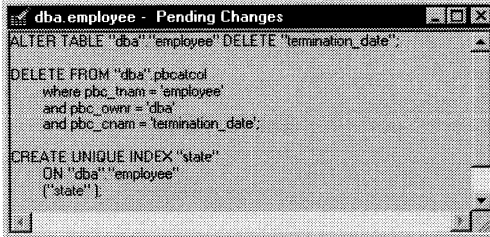
- ◆ Click the SQL Syntax button in the PainterBar.

or

Select Design>Syntax from the menu bar.

If you are working in a new table that hasn't been saved, PowerBuilder asks you to name the table before displaying SQL syntax.

PowerBuilder displays in SQL syntax the pending changes to the table definition:



```

ALTER TABLE "dba"."employee" DELETE "termination_date";

DELETE FROM "dba".pbcatal
  where pbc_tnam = 'employee'
  and pbc_ownr = 'dba'
  and pbc_cnam = 'termination_date';

CREATE UNIQUE INDEX "state"
  ON "dba"."employee"
  ("state");

```

The SQL statements execute only when you click the Save button to save the table definition or click the Close button and then tell PowerBuilder to save changes.

Copying, saving, and printing pending SQL changes

When you are viewing pending SQL changes, you can:

- ◆ Copy pending changes to the clipboard
- ◆ Save pending changes to a file
- ◆ Print pending changes

To copy, save, or print only part of the SQL syntax

Select the part of the SQL syntax you want before you copy, save, or print.

❖ **To copy the SQL syntax to the clipboard:**

- ◆ In SQL Syntax view, click the Copy button.
or
Press CTRL+C.

On Macintosh

On the Macintosh, press the COMMAND key instead of the CTRL key.

The SQL syntax or the selected part is copied to the clipboard.

❖ **To save SQL syntax for execution at a later time:**

- 1 In SQL Syntax view, Select File>Save As.
The Save Syntax to File dialog box displays.
- 2 Navigate to the folder in which you want to save SQL, name the file, and then click the Save button (on Windows and Macintosh) or the OK button (on UNIX).

The SQL syntax or the selected part is saved to the file.

At a later time, you can import the SQL file into the Database Administration painter and execute it.

❖ **To print pending table changes:**

- ◆ While viewing SQL syntax, select File>Print from the menu bar.
The SQL syntax or the selected part is printed.

Logging applied SQL syntax changes

You can choose to save to a log file the SQL syntax changes that have been applied to tables.

❖ **To save applied table changes to a log file:**

- 1 In the Table painter workspace, click the Properties button to display the Table Properties property sheet.

- 2 On the General property page, enter the name and location of the log file.

Now when you save changes to any table and close the Table painter, PowerBuilder will ask if you want to save applied changes to the log file.

Changes to other tables will be saved to the same log file

All instances of the Table painter will continue to ask about saving applied table changes to the log file until you delete or change the log file specification in the Table Properties property sheet. If you don't delete or change the file specification, changes to many tables can be saved to the same log. A banner that displays the date and time precedes the applied changes made for each Table painter instance.

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:**
 - ◆ In the Table painter workspace, select File>Print from the menu bar.

Working with keys

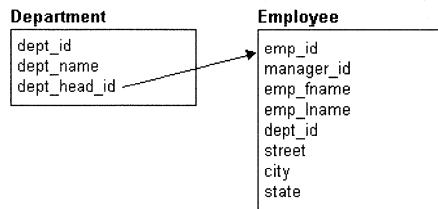
If your DBMS supports primary and foreign keys, you can work with the keys in PowerBuilder. When you open a table with keys in the Database painter, PowerBuilder gets the information from the DBMS and displays it in the painter workspace.

Why you should use keys

If your DBMS supports them, you should use primary and foreign keys to enforce the referential integrity of your database. If you use keys, you can rely on the DBMS to make sure that only valid values are entered for certain columns instead of having to write code to enforce valid values.

For example, say you have two tables, Department and Employee. The Department table contains the column Dept_Head_ID, which holds the ID of the department's manager. You want to make sure that only valid employee IDs are entered in this column. That is, the only valid values for Dept_Head_ID in the Department table are values for Emp_ID in the Employee table.

To enforce this kind of relationship, you define a foreign key for Dept_Head_ID that points to the Employee table. With this key in place, the DBMS disallows any value for Dept_Head_ID that does not match an Emp_ID in the Employee table:



FOR INFO For more about primary and foreign keys, consult a book about relational database design or your DBMS documentation.

What you can do in the painters

In this painter	You can
Database	Look at existing primary and foreign keys
Database	Open all tables that depend on a particular primary key
Database	Open the table containing the primary key used by a particular foreign key
Database and Table	Create, alter, and drop keys

For the most part, you work with keys the same way for each DBMS that supports keys. But there are some DBMS-specific issues.

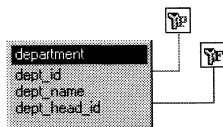
FOR INFO For complete information about using keys with your DBMS, see your DBMS documentation.

Viewing keys

In the Database painter, when you open and expand a table containing primary and/or foreign keys, PowerBuilder displays the keys in the workspace. The keys are shown as icons with lines connected to the table.

In the following picture, the Department table has two keys:

- ◆ A primary key (on dept_id)
- ◆ A foreign key (on dept_head_id)



If you can't see the lines

If the color of your window background makes it hard to see the lines for the keys and indexes, you can set the colors for each component of the Database painter's graphical table representation, including keys and indexes.

FOR INFO For information, see "Modifying database preferences" on page 347.

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 In the Database painter, open and expand the table containing the foreign key.
- 2 Display the popup menu for the button representing the foreign key.
- 3 Select Open Referenced Table from the popup menu.

❖ **To open all tables referencing a particular primary key:**

- 1 In the Database painter, open and expand the table containing the primary key.
- 2 Display the popup menu for the button representing the primary key.
- 3 Select Open Dependent Table(s) from the popup menu.

PowerBuilder opens and expands all tables in the database containing foreign keys that reference the selected primary key.

Defining primary keys

If your DBMS supports primary keys, you can define them in PowerBuilder.

❖ **To create or modify a primary key:**

- 1 In the Database painter, click the Table Properties button.
or
In the Table painter, click the Properties button.
- 2 Select the Primary Key tab.
- 3 In the Table Columns box, select one or more columns for the primary key (or select and deselect as needed to modify an existing primary key).

Each time you select a key column, the column displays in the Key Columns box. Each time you deselect a key column, the column is removed from the Key Columns box.

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 will get a DBMS error when you save the table. In DBMSs that allow rollback for Data Definition Language (DDL), the table definition will be rolled back. In DBMSs that don't allow rollback for DDL, the painter is refreshed with the current definition of the table.

You can reorder the columns in the key by dragging them with the mouse.

- 4 Specify any information required by your DBMS.

Naming a primary key

Some DBMSs allow you to name a primary key and specify whether it is clustered or not clustered. For these DBMSs, the Primary Key property page has a way to specify these properties.

FOR INFO For DBMS-specific information, see your DBMS documentation.

- 5 In the property sheet, click Apply to apply the create- or drop-create-primary-key request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
- 6 Click Save.

Completing the primary key

Some DBMSs automatically create a unique index when you define a primary key so that you can immediately begin to add data to the table. Others require that you separately create a unique index to support the primary key before populating the table with data.

FOR INFO To find out what your DBMS does, see your DBMS documentation.

Defining foreign keys

If your DBMS supports foreign keys, you can define them in PowerBuilder.

❖ **To create a foreign key:**

- 1 In the Database painter, click the Create FK button.

or

In the Table painter, click the Properties button, select the Foreign Keys tab, and click New.

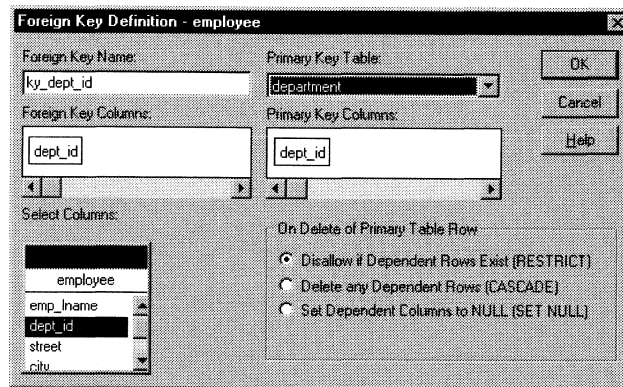
The Foreign Key Definition dialog box displays. Some of the information in the dialog box is DBMS-specific.

- 2 Name the foreign key in the Foreign Key Name box.
- 3 Select the columns for the foreign key in the Select Columns listbox.

The selected columns display in the Foreign Key Columns box. You can reorder the columns by dragging them with the mouse.

- 4 In the Primary Key Table listbox, select the table containing the Primary key referenced by the foreign key you are defining.

PowerBuilder displays the selected table's primary key in the Primary Key Columns box:



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 Specify any information required by your DBMS.
For example, you may need to specify a delete rule by clicking a radio button in the On Delete of Primary Table Row groupbox.
FOR INFO For DBMS-specific information, see your DBMS documentation.
- 6 Click OK to close the dialog box.
If you are working in the Database painter, you return to the workspace.
If you are working in the Table painter, you return to the Table Properties property sheet.
- 7 Click Apply to apply the create-foreign-key request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
- 8 Click Save to save your changes.

Dropping a key

You can drop keys (remove them from the database) from within PowerBuilder.

❖ To drop a key in the Database painter:

- 1 In the Database painter, open the table containing the key.
- 2 Select Drop Primary Key or Drop Foreign Key from the key's popup menu.
- 3 Click Yes to confirm the deletion.

❖ To drop a primary key in the Table painter:

- 1 In the Table painter workspace, click the Properties button and then select the Primary Key tab.
- 2 In the Table Columns box, deselect all the columns in the primary key.

- 3 In the property sheet, click Apply to apply the drop-primary-key request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
 - 4 Click Save.
- ❖ **To drop a foreign key in the Table painter:**
- 1 In the Table painter workspace, click the Properties button and then select the Foreign Keys tab.
 - 2 Select a key from the Foreign Keys box and then click Delete.
 - 3 In the property sheet, click Apply to apply the drop-foreign-key request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
 - 4 Click Save.

Working with indexes

You can work with indexes in the Database painter or the Table painter. You can create as many single- or multi-valued indexes for a database table as you need, and you can drop indexes that are no longer needed.

Update limitation

You can update a table in a DataWindow object only if it has a unique index or primary key.

Creating an index

In SQL Anywhere databases

You shouldn't define an index on a column that is defined as a foreign key, because foreign keys are already optimized for quick reference.

❖ To create an index:

- 1 In the Database painter, click the Index button.
or
In the Table painter, click the Properties button, select the Indexes tab, and click New.

The Create Index dialog box displays. Some of the information is DBMS-specific, so the dialog box may look different for different DBMSs.

- 2 Enter a name for the index in the Index Name box.
- 3 Select whether or not to allow duplicate values for the index.
- 4 Specify any other information required for your database (for example, in SQL Server specify whether the index is clustered, and in SQL Anywhere specify the order of the index).
- 5 Click the names of the columns that make up the index.
The selected column names display in the Index Columns box.
- 6 Click OK to close the dialog box.
If you are working in the Database painter, you return to the workspace.
If you are working in the Table painter, you return to the Table Properties property sheet.

- 7 Click Apply to apply the create-index request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
- 8 Click Save to save your changes.

Modifying an index

Modifying an index is similar to creating an index, but you can only modify an index in the Table painter.

In the Database painter, you can delete the index and create a new one. To see the definition of an index in the Database painter, double-click its icon.

❖ To modify an index:

- 1 In the Table painter, click the Properties button and then select the Indexes tab.
- 2 Select an index from the Indexes box and click Edit.
- 3 In the Create Index dialog box, select or deselect columns as needed.
- 4 In the property sheet, click Apply to apply the drop-create-index request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
- 5 Click Save.

Dropping an index

Dropping an index removes it from the database.

❖ To drop an index from a table in the Database painter:

- 1 In the Database painter workspace, display the popup menu for the icon of the index you want to drop.
- 2 Select Drop Index.
- 3 Click Yes to confirm the drop.

❖ **To drop an index from a table in the Table painter:**

- 1 In the Table painter workspace, click the Properties button and then select the Indexes tab.
- 2 Select the name of the index you want to drop.
- 3 Click Delete.
- 4 In the property sheet, click Apply to apply the delete-index request to the list of pending SQL statements to be submitted to the DBMS and then work on another property page.
or
Click OK to apply the request for changes and close the property sheet.
- 5 Click Save.

Working with views

A view gives you a different (and usually limited) perspective of the data in one or more tables. Although you see existing views listed in the Select Tables dialog box, a view does not physically exist in the database as a table does. Each time you select a view and use the view's data, PowerBuilder executes a SQL SELECT statement to retrieve the data and creates the view.

FOR INFO For more information about using views, see your DBMS documentation.

You can define and manipulate views in PowerBuilder. Typically you use views for the following reasons:

- ◆ To give names to frequently executed SELECT statements.
- ◆ To limit access to data in a table. For example, you can create a view of all the columns in the Employee table except Salary. Users of the view can see and update all information except the employee's salary.
- ◆ To combine information from multiple tables for easy access.

In PowerBuilder, you can create single- or multiple-table views. You can also use a view when you define data to create a new view.

You open and manipulate existing views in the Database painter. You define views in the View painter, which is similar to the Select painter.

FOR INFO For more information about the Select painter, see "Using SQL Select" on page 441.

Updating views

Some views are logically updatable; some are not. Some DBMSs don't allow any updating of views. For the rules your DBMS follows, see your DBMS documentation.

Opening a view

❖ To open a view:

- 1 From the Database painter, click the Open button.

or

Select Object>Select Tables from the menu bar.

The Select Tables dialog box displays listing all tables and views defined in the database.

- 2 Select the view.
- 3 Click Open.

Creating a view

❖ To create a view:

- 1 From the Database painter, click the View button.

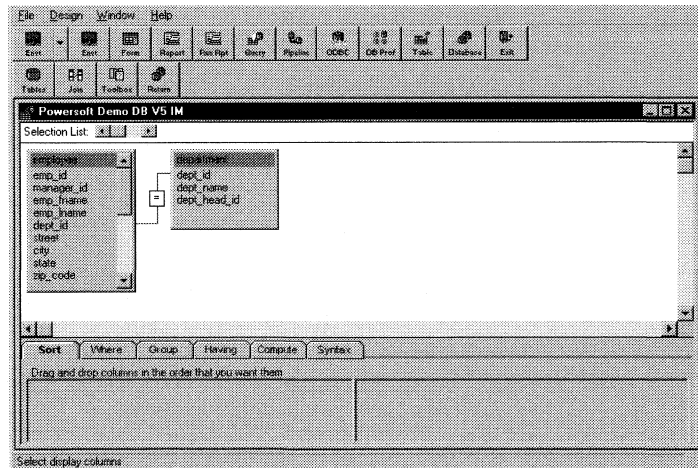
or

Select Object>New>View from the menu bar.

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:



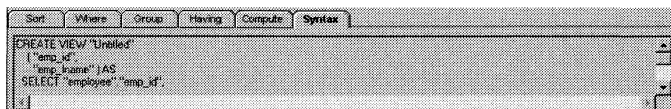
- 3 Select the columns to include in the view and include computed columns as needed.
- 4 Join the tables if there is more than one table in the view.
FOR INFO For information, see "Joining tables" on page 387.
- 5 Specify criteria to limit (Where tab), group retrieved rows (Group tab), and limit the retrieved groups (Having tab) if appropriate.
FOR INFO For information, see "Using SQL Select" on page 441. The View painter and the SQL Select painter are similar.
- 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.

PowerBuilder generates a CREATE VIEW statement and submits it to the DBMS. The view definition is created in the database. You return to the Database painter workspace with the new view displayed in the workspace.

Displaying a view's SQL statement

You can display the SQL statement that defines a view. How you do it depends on whether you are in the View painter creating a new view, or in the Database painter and want to look at the definition of an existing view.

- ❖ **To display the SQL statement from the View painter:**
 - ◆ Select the Syntax tab in the View painter.

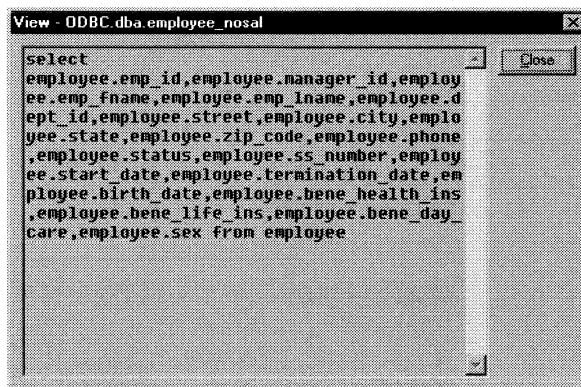


PowerBuilder displays the SQL it is generating. The display is updated each time you change the view.

❖ **To display the SQL statement from the Database painter:**

- 1 Open the view.
- 2 Double-click the name of the view in the painter workspace.
or
Open the popup menu and select Definition.

PowerBuilder displays the View dialog box showing the completed SELECT statement used to create the view:



View dialog box is read-only

You cannot alter the view definition in this dialog box. To alter a view, drop it and then create another view.

Joining tables

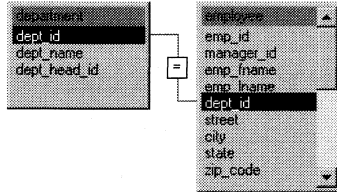
If the view contains more than one table, you should join the tables on their common columns. When the View painter is first opened for a view containing more than one table, PowerBuilder makes its best guess as to the join columns, as follows:

- ◆ If there is a primary/foreign key relationship between the tables, PowerBuilder automatically joins them.
- ◆ If there are no keys, PowerBuilder tries to join tables based on common column names and types.

❖ **To join tables:**

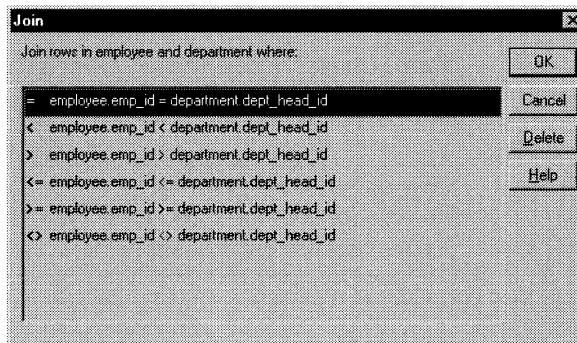
- 1 Click the Join button.
- 2 Click the columns on which you want to join the tables.

In the following screen, the Employee and Department tables are joined on the dept_id column:



- 3 To create a join other than the equality join, click the join representation in the workspace.

The Join dialog box displays:



- 4 Select the join operator you want from the Join dialog box.

If your DBMS supports outer joins, outer join options also display in the Join dialog box (for example, in the preceding dialog box, which uses the Employee and Department tables, you can choose to include rows from the Employee table where there are no matching departments, or rows from the Department table where there are no matching employees).

FOR INFO For more about outer joins, see your DBMS documentation.

Dropping a view

Dropping a view removes its definition from the database.

❖ **To drop a view:**

- 1 Select the view you want to drop in the Database painter workspace.
- 2 Click the Drop View button in the PainterBar.

PowerBuilder prompts you to confirm the drop, then generates a DROP VIEW statement and submits it to the DBMS.

Exporting table or view syntax

You can export the syntax for a table or view to the log. This feature is useful when you want to create a backup definition of the table or view before you alter it or when you want to create the same table or view in another DBMS.

To export to another DBMS, you must have the PowerBuilder interface for that DBMS.

❖ **To export the syntax of an existing table or view to a log:**

- 1 Select the table or view in the painter workspace.
- 2 Select Object>Export Syntax To Log from the menu bar.
If you selected a table and have more than one DBMS interface installed, the DBMS dialog box displays. If you selected a view, PowerBuilder immediately exports the syntax to the log.
- 3 Select the DBMS to which you want to export the syntax.
- 4 If you selected ODBC, specify a data source in the Data Sources dialog box.
- 5 Supply any information you are prompted for.

PowerBuilder exports the syntax to the log. Powersoft repository information (such as validation rules used) for the selected table is also exported. The syntax is in the format required by the DBMS you selected.

FOR INFO For more information about the log, see "Logging your work" on page 348.

Manipulating data

As you work on the database, you will often want to look at existing data or create some data for testing purposes. You will also want to test display formats, validation rules, and edit styles on real data.

PowerBuilder provides the Data Manipulation painter for such purposes. In this painter, you can:

- ◆ Retrieve and manipulate database information
- ◆ Save the contents of the database in a variety of formats (such as Excel, dBASE, or Lotus 1-2-3)

Opening the Data Manipulation painter

❖ **To open the Data Manipulation painter:**

- 1 In the Database painter, select the table or view whose data you want to manipulate.
- 2 Click the Grid, Table, or Freeform button in the PainterBar.

or

Select Object>Edit Data from the menu bar and choose one of the edit options from the cascading menu that displays.

The Data Manipulation painter opens and all rows are retrieved. As the rows are being retrieved, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval.

Exactly what you see depends on the formatting style you picked. The Data Manipulation painter is actually a DataWindow object. The formatting style you picked corresponds to a type of DataWindow object (grid, tabular, or freeform). In a grid display, you can drag the mouse on a column's border to resize the column.

The window shown below is in the grid format:

Customer ID	First Name	Last Name	Address	City
101	Michaels	Devlin	3114 Pioneer Avenue	Rutherford
102	Beth	Reiser	1033 Whippary Road	New York
103	Erin	Niedringhaus	1990 Windsor Street	Paoli
104	Meghan	Mason	550 Dundas Street East	Knoxville
105	Laura	McCarthy	1210 Highway 36	Carmel
106	Paul	Phillips	2000 Cherry Creek N. Dr.	Middletown

Only a few rows of data display at a time. You can use the First, Prior, Next, and Last buttons to move from page to page.

Retrieving data

- ❖ **To retrieve rows from the database:**
 - ◆ Click the Retrieve button in the PainterBar.
or
 - Select Rows>Retrieve from the menu bar.

PowerBuilder retrieves all the rows in the current table or view. As the rows are being retrieved, the button changes to Cancel. You can click it to stop the retrieval.

Modifying data

You can add, modify, or delete rows. When you 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; some are not. Some DBMSs don't allow any updating of views.

FOR INFO For the rules your DBMS follows regarding updating of views, see your DBMS documentation.

❖ **To modify data:**

- 1 Take one of the following actions:

To do this	Take this action
Modify existing data	Tab to a field and enter a new value
Add a row	Click the Insert Row button and enter data in the new row
Delete a row	Click the Delete Row button

The Data Manipulation painter uses validation rules, display formats, and edit styles that you or others have defined for the table in the Database painter when you add or modify data.

- 2 Click the Update Database button to apply changes to the database.

Sorting and filtering data

You can define and use sort criteria and filters for the rows.

The sort criteria and filters you define in the Data Manipulation painter are for testing only and are not saved with the table or passed to the DataWindow painter.

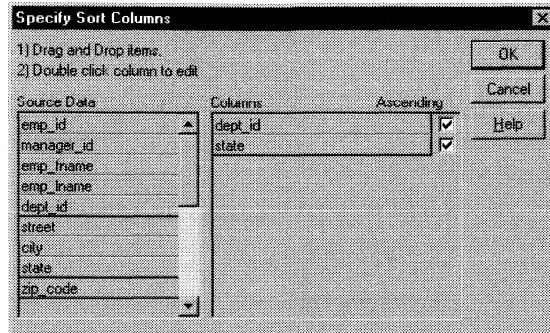
Sorting rows

❖ **To sort the rows:**

- 1 Select Rows>Sort from the menu bar.

The Specify Sort Columns dialog box displays.

- 2 Drag the columns you want to sort on from the Source Data box to the Columns box:



A checkbox with a check 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 checkbox.

Precedence of sorting

The order in which the columns display in the Columns box determines the precedence of the sorting. For example, in the preceding dialog box, rows will be sorted by department ID. Within department ID, rows will be sorted by state.

To change the precedence order, drag the column names in the Column box to the order you want.

- 3 (Optional) Double-click an item in the Columns box to specify an expression to sort on.

The Modify Expression dialog box displays.

- 4 Specify the expression.

For example, if you have two columns, Revenues and Expenses, you can sort on the expression *Revenues – Expenses*.

- 5 Click OK.

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.

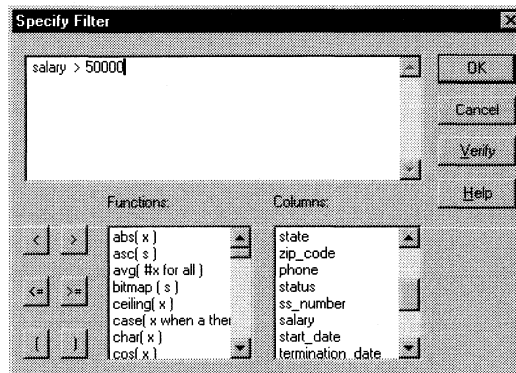
- 6 When you have specified all the sort columns and expressions, click OK.

Filtering rows

You can limit which rows are displayed by defining a filter.

❖ **To filter the rows:**

- 1 Select Rows>Filter from the menu bar.
The Specify Filter dialog box displays.
- 2 Enter a boolean expression that PowerBuilder will test against each row:



If the expression evaluates to TRUE, the row will be displayed. You can paste functions, columns, and operators in the expression.

- 3 Click OK.
PowerBuilder filters the data. Only rows meeting the filter criteria are displayed.

❖ **To remove the filter:**

- 1 Select Rows>Filter from the menu bar.
The Specify Filter dialog box displays showing the current filter.
- 2 Delete the filter expression, then click OK.

Filtered rows and updates

Filtered rows are updated when you update the database.

Viewing row information

You can display information about the data you have retrieved.

❖ **To display row information:**

- ◆ Select Rows>Described from the menu bar.

The Describe Rows dialog box displays showing the number of:

- ◆ Rows that have been deleted in the painter *but not yet deleted from the database*
- ◆ Rows displayed in Preview
- ◆ Rows that have been filtered
- ◆ Rows that have been modified in the painter *but not yet modified in the database*

All row counts are zero until you retrieve the data from the database or add a new row. The count changes when you modify the displayed data or test filter criteria.

Importing data

You can import data from an external source and display it in the Data Manipulation painter, then save the imported data in the database.

❖ **To import data:**

- 1 Select Rows>Import from the menu bar.

The Select Import File dialog box displays.

- 2 Specify the file from which you want to import the data.

The types of files that you can import into the painter are shown in the Files of Type dropdown listbox.

- 3 Click Open.

PowerBuilder reads the data from the file into the painter. You can then click the Update Database button in the PainterBar to add the new rows to the database.

Printing data

You can print the data displayed in the Data Manipulation painter by selecting File>Print from the menu bar. Before printing, you can also preview the output on the screen.

❖ **To preview printed output before printing:**

- 1 Select File>Print Preview from the menu bar.

Preview displays the data as it will print. To display rulers around the page borders in Print Preview, select File>Print Preview Rulers.

- 2 To change the magnification used in Print Preview, select File>Print Preview Zoom from the menu bar.

The Zoom dialog box displays.

- 3 Select the magnification you want and click OK.

Preview zooms in or out as appropriate.

- 4 When you have finished looking at the print layout, select File>Print Preview from the menu bar again.

Saving data

You can save the displayed data in an external file.

❖ **To save the data in an external file:**

- 1 Select File>Save Rows As from the menu bar.

The Save Rows As dialog box displays.

- 2 Choose a format for the file.

You can select from several formats, including Powersoft report (PSR) or HTML.

If you want the column headers saved in the file, select a file format that includes headers, such as Excel With Headers. When you select a *with headers* format, the names of the database columns (not the column labels) will also be saved in the file.

FOR INFO For more information, see "Saving data in an external file" on page 488.

- 3 Name the file and save it.

PowerBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Returning to the Database painter workspace

- ❖ **To leave the Data Manipulation painter and return to the Database painter workspace:**

- 1 Select File>Close from the menu bar.

If you have made changes to the database but not yet saved them, PowerBuilder asks you whether you want to update the database.

- 2 If you have unsaved changes, respond to the prompt.

You return to the Database painter.

Administering the database

You can use the Database Administration painter to control access to the database and create SQL for immediate execution.

About administering databases

Administering your databases with the Database Administration painter means having the ability to:

- ◆ Create SQL statements to send to the DBMS for immediate execution
- ◆ Control access to the current database

Building SQL statements

The Database Administration painter's workspace is a SQL editor in which you can enter SQL statements and execute them. The painter 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.

Controlling database access

The Database Administration painter's Design menu provides access to a series of dialog boxes you can use to control access to the current database. For example, in some DBMSs you can assign table access privileges to users and groups.

Which menu items display on the Design menu and which dialog boxes display depend on your DBMS.

FOR INFO For information about support for security options in your DBMS, see *Connecting to Your Database* and your DBMS documentation.

Opening the Database Administration painter

❖ To open the Database Administration painter:

- ◆ Click the Database Administration button in the Database painter's PainterBar.
or
Select Design>Database Administration from the menu bar in the Database painter.

Using the editor

The Database Administration painter provides the same editing capabilities as the file editor. It also has General, Font, and Coloring properties that you can change to make SQL files easier to read. With no change in properties, SQL files will have black text on a white background and a tab stop setting of 3 for indentation.

Select Design>Options from the menu bar to open the editor's property sheet. The General and Font properties are the same as those you can set for the File Editor.

FOR INFO For more information, see "Using the file editor" on page 27.

Editor properties apply elsewhere

When you set General and Font properties for the Database Administration painter, the settings also apply to the Function and Script painters, the file editor, and the Debug window.

Setting coloring properties

In addition to setting General and Font properties, you can set the text color and background color for SQL styles, such as data types 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 of the Properties for Editor property sheet.

Enabling syntax coloring

Be sure the Enable Syntax Coloring checkbox is selected before you set colors for SQL styles. You can turn off all Coloring properties by clearing the checkbox.

Building and executing SQL statements

You can use the Database Administration painter to build SQL statements and execute them immediately. The painter's workspace acts as a notepad in which you can enter SQL statements.

Creating stored procedures

You can use the Database Administration painter to create stored procedures or triggers, but make sure that the painter's SQL statement terminator character is not the same as the terminator character used in the stored procedure language of your DBMS.

About the statement terminator

By default, PowerBuilder uses the semicolon as the SQL statement terminator. You can override the semicolon by specifying a different terminator character in the Database painter. To change the terminator character, select **Design>Options** from the Database painter's menu bar.

About comments

By default, PowerBuilder strips off comments when it sends SQL to the DBMS. You can have comments included by clearing the check next to **Design>Strip Comments** from the menu bar.

Entering SQL

You can enter a SQL statement in three ways:

- ◆ Pasting the statement
- ◆ Typing the statement in the workspace
- ◆ Opening a text file containing the SQL

Pasting SQL

You can paste **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements to the workspace. Depending on which kind of statement you want to paste, PowerBuilder displays dialog boxes that guide you through pasting the full statement.

❖ **To paste a SQL statement to the workspace:**

- 1 Click the **Paste SQL** button in the PainterBar.

or

Select **Edit>Paste SQL** from the menu bar.

The **SQL Statement Type** dialog box displays listing the types of SQL statements you can use.

- 2 Double-click the appropriate icon to select the statement type.

The **Select Table** dialog box displays.

- 3 Select the table(s) you will reference in the SQL statement.

You go to the Select, Insert, Update, or Delete painter, depending on the type of SQL statement you are pasting. The Insert, Update, and Delete painters are similar to the Select painter, but only the appropriate tabs display in the SQL toolbox at the bottom of the workspace.

FOR INFO For more information about the Select painter, see "Using SQL Select" on page 441.

- 4 Follow the procedure for the statement you are pasting:

Type of statement	What you do
SELECT	Define the statement exactly as in the Select painter when building a view. You choose the columns to select. If you want, you can define computed columns, specify sorting and joining criteria, and WHERE, GROUP BY, and HAVING criteria FOR INFO For more information, see "Working with views" on page 384
INSERT	Type the values to insert into each column. You can insert as many rows as you want
UPDATE	First specify the new values for the columns in the Update Column Values dialog box. Then specify the WHERE criteria to indicate which rows to update
DELETE	Specify the WHERE criteria to indicate which rows to delete

- 5 When you have completed painting the SQL statement, click the Return button in the PainterBar in the Select, Insert, Update, or Delete painter.

You return to the Database Administration painter with the SQL statement pasted into the workspace.

Typing SQL

If you want, you can simply type one or more SQL statements directly in the workspace.

You can enter 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 CREATE TRIGGER statement). You cannot enter certain statements that could destabilize the PowerBuilder development environment. These include the SET statement and the USE *database* statement.

Sybase SQL server stored procedures

When you use the Database Administration painter to execute a Sybase SQL Server system stored procedure, you *must* start the syntax with the keyword EXEC or EXECUTE. You cannot execute the stored procedure simply by entering its name. For example, enter:

```
EXEC SP_LOCK
```

Importing SQL from a text file

You can import SQL that has been saved in a text file into the Database Administration painter.

❖ To read SQL from a file:

- 1 Position the insertion point where you want to insert the SQL.
- 2 Select File>Import from the menu bar.
The File Import dialog box displays.
- 3 Select the file containing the SQL and click OK.

Explaining SQL

Sometimes there is more than one way to code SQL statements to obtain the results you want. When this is the case, you can use Explain SQL on the Design menu to help you select the most efficient method. Explain SQL displays information about the path that PowerBuilder will use to execute the statements in the SQL Statement Execution Plan dialog box. This is most useful when you are retrieving or updating data in an indexed column or using multiple tables.

DBMS-specific information

The information displayed in the SQL Statement Execution Plan dialog box depends on your DBMS.

FOR INFO For more about the SQL execution plan, see your DBMS documentation.

Executing SQL

When you have the SQL statements you want in the workspace, you can submit them to the DBMS.

❖ **To execute the SQL:**

- ◆ Click the Execute button.

or

Select Design>Execute SQL from the menu bar.

What happens next

If the SQL retrieves data, the data appears in a window identical to a grid Data Manipulation painter.

FOR INFO For a description of what you can do with the data, see "Manipulating data" on page 390.

If there is a database error, you see a message box describing the problem.

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

Topic	Page
Introducing DataWindow objects	406
Introducing reports	409
Building a DataWindow object	417
Choosing a presentation style	420
Choosing DataWindow object-wide options	428
Defining the data source	430
Generating and saving a DataWindow object	460
Defining queries	462
What's next	465

Introducing 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).

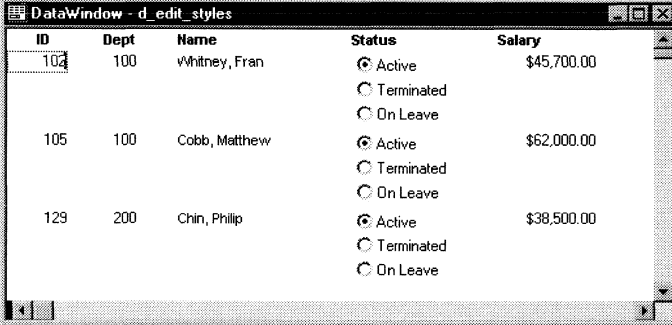
DataWindow objects have knowledge about the data they are retrieving. You can specify display formats, presentation styles, and other data properties so the data is used in the most meaningful way by users.

DataWindow object examples

You can display the data in the format that will best present the data to your users:

Edit styles

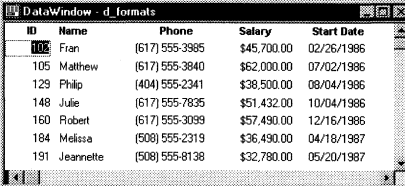
If a column can take only a small number of values, you can have the data appear as radio buttons in a DataWindow object so users know what their choices are.



ID	Dept	Name	Status	Salary
104	100	Whitney, Fran	<input checked="" type="radio"/> Active <input type="radio"/> Terminated <input type="radio"/> On Leave	\$45,700.00
105	100	Cobb, Matthew	<input checked="" type="radio"/> Active <input type="radio"/> Terminated <input type="radio"/> On Leave	\$62,000.00
129	200	Chin, Philip	<input checked="" type="radio"/> Active <input type="radio"/> Terminated <input type="radio"/> On Leave	\$38,500.00

Display formats

If a column displays phone numbers, salaries, or dates, you can specify the format appropriate to the data.



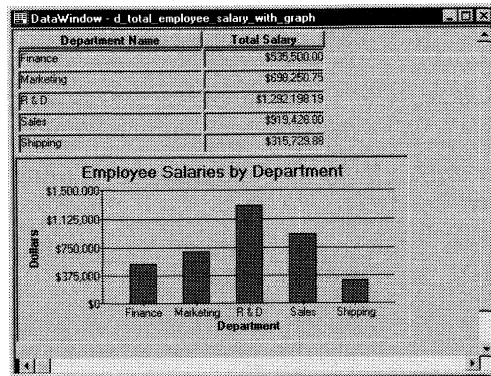
ID	Name	Phone	Salary	Start Date
104	Fran	(617) 555-3985	\$45,700.00	02/26/1986
105	Matthew	(617) 555-3940	\$62,000.00	07/02/1986
129	Philip	(404) 555-2341	\$38,500.00	08/04/1986
148	Julie	(617) 555-7835	\$51,432.00	10/04/1986
160	Robert	(617) 555-3099	\$57,490.00	12/16/1986
184	Melissa	(508) 555-2319	\$36,490.00	04/18/1987
191	Jeannette	(508) 555-8138	\$32,780.00	05/20/1987

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 in an application, you need to build the object. To do that you can go to the DataWindow painter, which lets you create and edit DataWindow objects. It also lets you make **PSR** (Powersoft Report) files, which you may want to use in applications too. A PSR file contains a report definition—essentially a nonupdatable DataWindow object—as well as the data contained in that report when the PSR file was created.

This section describes the overall process for creating and using DataWindow objects.

❖ To use DataWindow objects:

- 1 Create the DataWindow *object* (or PSR file) in the DataWindow painter.

In this painter, you define the data source, presentation style, and all other properties of the object, such as display formats, validation rules, sorting and filtering criteria, and graphs.

FOR INFO For more information about creating a DataWindow object, see "Building a DataWindow object" on page 417.

- 2 Place a DataWindow *control* in a window (or user object).

It is through this control that your application communicates with the DataWindow object you created in the DataWindow painter.

- 3 Associate the DataWindow control with the DataWindow object.

- 4 Write scripts in the Window painter to manipulate the DataWindow control and its contents.

For example, you use the PowerScript Retrieve function to retrieve data into the DataWindow control.

You can write scripts for the DataWindow control to deal with error handling, sharing data between DataWindow controls, and so on.

FOR INFO For more information about using DataWindow objects, see *Application Techniques*.

- 5 Write code to control the processing that is initiated when events occur in the DataWindow control.

You can write scripts for the DataWindow control to deal with error handling, sharing data between DataWindow controls, and so on.

Introducing reports

Reports present data. The Report painter in PowerBuilder provides many ways for you to present data. You may want a tabular report with rows and columns of information. Sometimes a graph or a crosstab is a better way to present the data.

A PowerBuilder report can also be mailing labels or many reports nested together on the same page. Freeform PowerBuilder reports let you place text, data, lines, boxes, and pictures anywhere you want. This means you can be very creative.

Report examples

Following are sample PowerBuilder reports that use data from the Powersoft Demo Database.

Tabular report

The tabular report is the most common kind of report. You use it for presenting information in rows and columns. This is a basic tabular report:

List of Employees						
Employee Last Name	Employee First Name	Street	City	State	Zip Code	Phone
Evans	Scott	10-A Sunrise Circle	Concord	MA	01742-	(508) 555-0096
Kelly	Moira	12 Fountain Road	Gloucester	MA	01930-	(508) 555-3769
Leticq	John	149 Vista Drive	Burlington	MA	01803-	(617) 555-1167
Lull	Kim	199 Lincoln Street	Concord	MA	01742-	(508) 555-4444
Pastor	Lynn	14 Cricklewood Drive	Burlington	MA	01803-	(617) 555-2001
Rabkin	Andrew	44 Birds Hill Way	Burlington	MA	01803-	(617) 555-4444
Savarino	Pamela	112 Beach Street	Long Beach	CA	90806-	(310) 555-1857
Scott	David	21 Riverdale Drive	Belmont	MA	02178-	(617) 555-3246
Shea	Mary Anne	197 Camden Road	Lexington	MA	02173-	(617) 555-4616
Sheffield	John	45 Belleview Drive	Houston	TX	77079-	(713) 555-3877
Shishov	Natasha	15 Milk Street	Waltham	MA	02154-	(617) 555-2755
Sterling	Paul	112 Endicott Street	Concord	MA	01742-	(508) 555-0295
Sullivan	Dorothy	124 Minuteman Drive	Lexington	MA	02173-	(617) 555-3947
Wang	Albert	48 Edwin Street	Waltham	MA	02154-	(617) 555-8741

A more advanced tabular report

This tabular report includes a column computed from other columns (Salary Plus Benefits) and special enhancements such as the *Confidential* watermark:

Total Compensation Report								Page 1 of 4	
Salary Plus Benefits								3/22/97	
								Value of health ins. = \$4,800 Value of life insurance = \$(5.43 x salary)/1,000 Value of day care = \$5,200	
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits	
100	102	Fran	Whitney	\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748	
	105	Matthew	Cobb	\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137	
	160	Robert	Bregault	\$67,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802	
	243	Nafasha	Shishov	\$72,995	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191	
	247	Kurt	Driscoll	\$48,024	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284	
	249	Rodrigo	Guevara	\$42,998	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031	
	266	Ram	Gowda	\$59,840	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165	
	278	Terry	Melkisetian	\$48,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763	
	316	Lynn	Pastor	\$74,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905	
	445	Kim	Lull	\$67,900	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$93,177	
	453	Andrew	Rabkin	\$64,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$69,650	

Grouped report

This Group style report groups by department and lists employees and salaries. It also includes a subtotal and a grand total for the salary column:

Employee Report				
4/8/97				
Department ID	Employee ID	First Name	Last Name	Salary
500	191	Jeannette	Bertrand	\$29,800
	703	Jose	Martinez	\$55,501
	750	Jane	Braun	\$34,300
	868	Felicia	Kuo	\$28,200
	921	Charles	Crowley	\$41,700
	1013	Joseph	Barker	\$27,290
	1570	Anthony	Rebeiro	\$34,576
	1815	Sheila	Romero	\$27,500
	1658	Michael	Lynch	\$24,903
Total for department:				\$303,770
Grand Total:				\$3,749,147

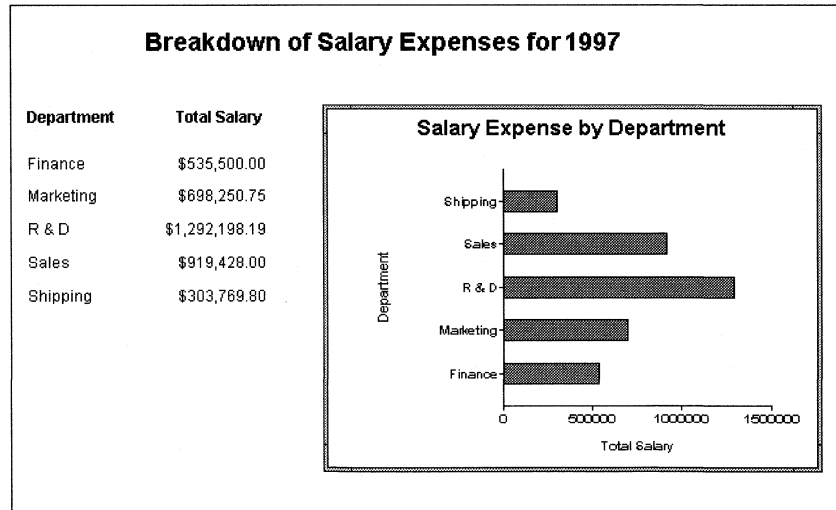
Grid report

This grid report looks very much like a tabular report. However, the grid is a rigid structure of rows and columns. You can change the column width and reorder the columns while you are viewing retrieved data. The grid report is useful for ad hoc reporting:

Employee Last Name	Employee First Name	Street	City	State	Zip Code	Phone
Ahmed	Alex	114 Cushing Street	Needham	MA	02192-	(617) 555-8748
Barker	Joseph	58 West Drive	Bedford	MA	01730-	(617) 555-8021
Barletta	Irene	37 Gleason Street	Bedford	MA	01730-	(617) 555-8345
Bertrand	Jeannette	209 Concord Street	Acton	MA	01720-	(508) 555-8138
Bigelow	Janet	84 Lunda Street	Waltham	MA	02154-	(617) 555-1493
Blaikie	Barbara	66 Beaumont Terrace	Needham	MA	02192-	(617) 555-9345
Braun	Jane	45 Wood Street	Cambridge	MA	02140-	(617) 555-7857
Breault	Robert	58 Cherry Street	Milton	MA	02186-	(617) 555-3099
Bucceri	Matthew	57 Taylor Place	Lexington	MA	02173-	(617) 555-5336
Butterfield	Joyce	119 Adams Street	Cambridge	MA	02140-	(617) 555-2232
Chao	Shih Lin	59 Holyoke Street	Lexington	MA	02173-	(617) 555-5921
Charlton	Doug	57 Webster Street	Concord	MA	01742-	(508) 555-9246
Chin	Phillip	59 Pond Street	Atlanta	GA	30339-	(404) 555-2341
Clark	Alison	56 Carver Street	Emeryville	CA	94608-	(510) 555-9437

Report and graph

This report lists the salary totals by department. The graph presents the same data in a visual way, which makes it easier to see the relative cost of personnel in the five departments:



Freeform report













This freeform report presents all of the information about employees, one at a time. You can move the information around easily until you get what you want. The bitmap in the background marks the information as confidential:

Employee Information

Employee ID: 102	Status: <input checked="" type="radio"/> Active	
Manager ID: 501	<input type="radio"/> Terminated	
Emp. First Name: Fran	<input type="radio"/> On Leave	
Emp. Last Name: Whitney	Soc. Sec. No.: 017-34-9033	
Department ID: 100	Salary: \$45,700.00	
Street: 49 East Washington Street	Start Date: 8/28/1984	
City: Needham	Termination Date: 0/00/0000	
State: MA	Birth Date: 6/05/1958	
Zip Code: 02192-	Health Insurance: <input checked="" type="checkbox"/>	
Phone: (617) 555-3985	Life Insurance: <input checked="" type="checkbox"/>	
	Day Care: <input type="checkbox"/>	

Mailing labels

These mailing labels use the name and address information from the employee table and a bitmap to mark them with a logo:

 Alex Ahmed 114 Cushing Street Needham, MA 02192	 Joseph Barker 58 West Drive Bedford, MA 01730	 Irene B 37 Gle Bedford
 Jeannette Bertrand 209 Concord Street Acton, MA 01720	 Janet Bigelow 84 Lunda Street Waltham, MA 02154	 Barbar 66 Bea Needh
 Jane Braun 45 Wood Street Cambridge, MA 02140	 Robert Breaull 58 Chery Street Milton, MA 02186	 Matthe 57 Tay Lexing
 Joyce Butterfield 119 Adams Street Cambridge, MA 02140	 Shih Lin Chao 59 Holyoke Street Lexington, MA 02173	 Doug 57 Wel Conco

N-up report

This n-up report shows four rows of information next to each other. Similar to the freeform report, n-up is useful for fitting more information on the page. N-up is also useful for presenting periodic information, such as data that repeats for Monday through Friday (five blocks):

Contacts				
Last First Phone: Fax:	Bertrand Coleman (706) 555-2886 (704) 555-4532	Brier Michael (617) 555-2398 (617) 555-3337	Burrill Dana (617) 555-7956 (617) 555-2398	Caruso William (617) 555-2144 (617) 555-1656
Last First Phone: Fax:	Chin David (617) 555-3378 (617) 555-4453	Clarke Molly (617) 555-4325 (617) 555-7638	Cobb Paul (404) 555-2239 (404) 555-8111	Cohen Paul (617) 555-8883 (617) 555-4499
Last First Phone: Fax:	Collins MargBeth (617) 555-1199 (617) 555-9586	Critch Susan (508) 555-4829 (508) 555-3025	Crossland Ellen (617) 555-0004 (617) 555-8005	Crowley Charles (617) 555-1344 (617) 555-9877
Last First Phone: Fax:	Davidson Joann (510) 555-7363 (510) 555-9278	DeMarco Michael (617) 555-4400 (617) 555-7876	Dewey Michael (617) 555-9877 (617) 555-2322	Elkins John (603) 555-1200 (603) 555-0078
Last First Phone: Fax:	Evans Carrie (404) 555-1163 (404) 555-8244	Fish Jeffrey (617) 555-3528 (617) 555-9563	Galvin Liz (617) 555-9312 (617) 555-9870	Glassmann Beth (617) 555-0273 (617) 555-9933

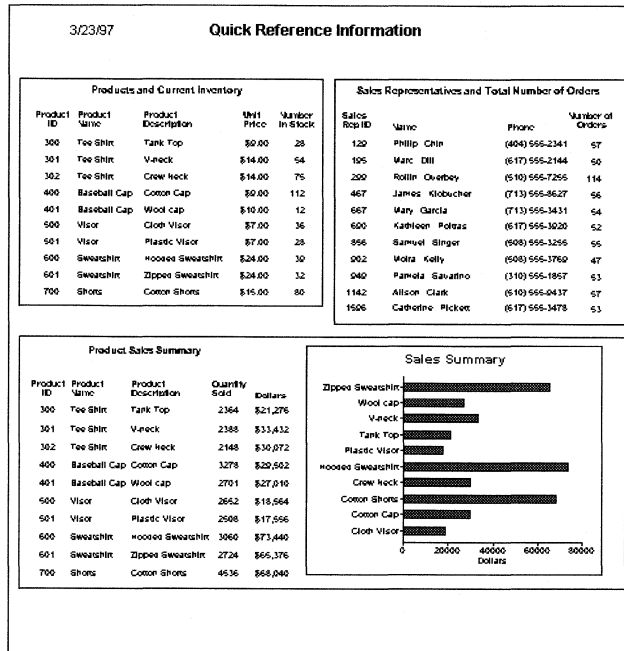
Crosstab

This crosstab report counts the number of employees that fit into each cell. For example, there are three employees in department 100 who make between \$30,000 and \$39,999:

Number of employees by department and salary	Dept Id					Total number of employees making the salary
	100	200	300	400	500	
30,000 includes up to 39,999						
Salary						
20000				2	5	7
30000	3	8	2	5	2	20
40000	6	5	2	5	1	19
50000	4	3	3	2	1	13
60000	4	1		2		7
70000	2	1	1			4
80000	2	1				3
90000	1					1
130000			1			1
Total number of employees in the department	22	19	9	16	9	

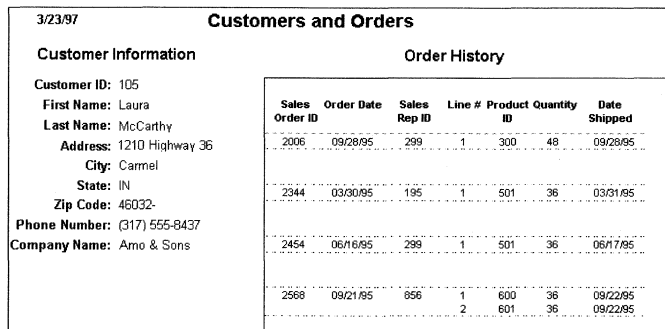
Composite report with nested reports

This composite report consists of three nested tabular reports. One of the tabular reports includes a graph. Composite reports are a way to show different reports together on the same page:



Freeform report with a nested report

This freeform report lists all information about a customer and includes a related nested report that lists all the orders that belong to the customer. This is an example of a master/detail relationship—one customer has many orders:



Reports versus DataWindow objects

Report and DataWindow objects are the same objects. When you create a report in the Report painter, you are actually creating a DataWindow object that you can open and modify in the DataWindow painter.

How they differ

The only difference between DataWindow objects and reports is that reports are by definition not updatable and DataWindow objects can be updatable. That means that there are several differences between the DataWindow painter and Report painter:

- ◆ **Update Properties** The DataWindow painter has a Rows>Update Properties menu item, which displays a Specify Update Properties dialog box where you can specify which columns in the DataWindow object are updatable. The Report painter does not have an Update Properties menu item because reports are not updatable.
- ◆ **Validation Rules** In the DataWindow painter you can specify validation rules for columns. You can't specify validation rules in the Report painter, because users don't enter values in reports.
- ◆ **Previewing** In the DataWindow painter, when you preview an updatable DataWindow object, you can manipulate the data. In the Report painter, when you preview a report, you go to Print Preview, where you cannot manipulate the data.
- ◆ **Mailing** In the DataWindow painter, you cannot open and mail Powersoft report files (PSR files). In the Report painter, you can.

The DataWindow painter functions

Because DataWindow objects and reports are fundamentally the same, the functions described in the *DataWindow Reference* also apply to reports.

Why use the Report painter

The Report painter is provided with PowerBuilder as a convenience. You need never use the Report painter—you can do all your development work in the DataWindow painter. However, if you are creating a DataWindow object that is not updatable (that is, a *report*), you might want to use the Report painter so you won't have to make sure the object is specified as not updatable.

Two versions of the Report painter

There are actually two versions of the Report painter that come with PowerBuilder. The first Report painter button lets you design and run reports (that is, DataWindow objects that are not updatable); the second button lets you only run reports.

Building a DataWindow object

You use the DataWindow painter to work with DataWindow objects.

Connecting to a database

To use the DataWindow painter, you must be connected to the database whose data you will be accessing.

When you open the DataWindow painter, PowerBuilder connects you to the DBMS and database you used last. If you need to connect to a different database, do so before working with a DataWindow object.

FOR INFO For information about changing your database connection, see *Connecting to Your Database*.

Modifying an existing DataWindow object

❖ **To modify an existing DataWindow object:**

- 1 Click the DataWindow button in the PowerBar.

The Select DataWindow dialog box displays listing the DataWindow objects in the current library.

- 2 Select the existing DataWindow object and click OK.

PowerBuilder opens the DataWindow painter and displays the DataWindow object in the workspace.

FOR INFO To learn how you can modify an existing DataWindow object, see Chapter 15, "Enhancing DataWindow Objects".

Creating a new DataWindow object

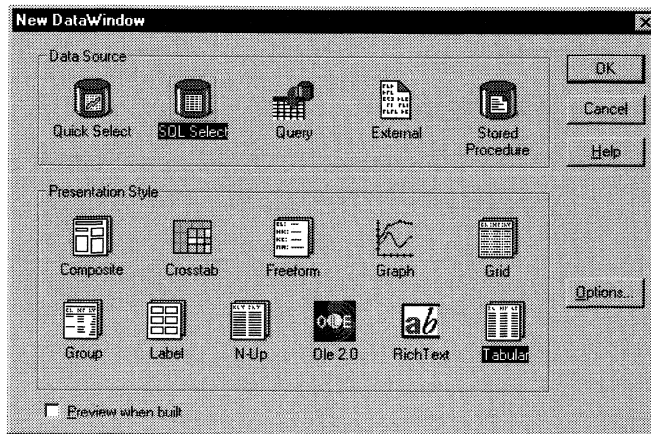
❖ **To create a new DataWindow object:**

- 1 Click the DataWindow button in the PowerBar.

The Select DataWindow dialog box lists the DataWindow objects in the current library.

- 2 Click the New button.

The New DataWindow dialog box displays:



- 3 Choose a presentation style—how you want the data to be arranged—for the DataWindow object.

FOR INFO See "Choosing a presentation style" on page 420.

- 4 (Optional) Choose options for the DataWindow object.

FOR INFO See "Choosing DataWindow object-wide options" on page 428.

- 5 Choose a data source for the DataWindow object.

FOR INFO See "Defining the data source" on page 430.

- 6 If you want to immediately preview (run) the basic DataWindow object after defining the presentation style and data source, select the Preview When Built checkbox.

After choosing and defining the data source, you go to the DataWindow painter workspace. Although you can now save the DataWindow object and begin using it in a window, you may want to enhance it first.

- 7 (Optional) Enhance your DataWindow object using the DataWindow painter workspace.

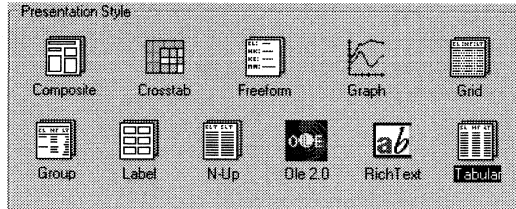
You can customize the display of column data and add objects such as text, bitmaps, computed fields, and graphs.

You can also preview the DataWindow object at any time in the DataWindow painter.

FOR INFO For more information about previewing, see Chapter 15, "Enhancing DataWindow Objects".

Choosing a presentation style

The presentation style you select for a DataWindow object determines the format PowerBuilder uses to first display the DataWindow object in the DataWindow painter workspace. You can use the format as displayed or modify it to meet your needs:



You can choose from the following presentation styles in the New DataWindow dialog box:

- Tabular
- Freeform
- Grid
- Label
- N-Up
- Group
- Composite
- Graph
- Crosstab
- OLE 2.0 (on Windows)
- RichText

Using the Tabular style

The Tabular presentation style presents data with the data columns going across the page and headers above each column. As many rows from the database will display at one time as can fit in the DataWindow object. You can reorganize the default layout any way you want by moving columns and text. Tabular style is often used when you want to group data:

Employee Compensation Report		June 17, 1997 08:23 AM	
Department:	Finance		
Bigelow, Janet	\$31,200.00		
Coe, Kristen	\$36,500.00		
Coleman, James	\$42,300.00		
Davidson, Jo Ann	\$57,090.00		
Higgins, Denis	\$43,700.00		
Jordan, Julie	\$51,432.00		
Lehecq, John	\$75,400.00		
Litton, Jennifer	\$58,930.00		
Shea, Mary Anne	\$138,948.00		
Department Salaries	Total:	\$535,500.00	Minimum Salary: \$31,200.00
	Average:	\$59,500.00	Maximum Salary: \$138,948.00

Page 1 of 8

Using the Freeform style

The Freeform presentation style presents data with the data columns going down the page and labels next to each column. You can reorganize the default layout any way you want by moving columns and text: Freeform style is often used for data entry forms.

Employee ID: 501

First Name: David

Last Name: Scott

Salary: \$96,300.00

Status:

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 grid style is that users can reorder and resize columns during execution.

Original Grid report

This grid report shows employee information. Several of the columns have a large amount of extra white space:

Employee ID	First Name	Last Name	Street	City	State
102	Fran	Whitney	49 East Washington Street	Needham	MA
105	Matthew	Cobb	77 Pleasant Street	Waltham	MA
123	Philip	Chin	59 Pond Street	Atlanta	GA
143	Julie	Jordan	144 Great Plain Avenue	Winchester	MA
160	Robert	Breault	58 Cherry Street	Milton	MA
184	Melissa	Espinoza	112 Apple Tree Way	Stow	MA
191	Jeannette	Bertrand	209 Concord Street	Acton	MA
195	Marc	Dill	89 Hancock Street	Milton	MA
207	Jane	Francis	112 Hawthorne Drive	Concord	MA

Grid report with modified column widths













This grid report was created from the original one by decreasing the width of some columns:

Employee ID	First Name	Last Name	Street	City	State	Phone
102	Fran	Whitney	49 East Washington Street	Needham	MA	(617) 555-3985
105	Matthew	Cobb	77 Pleasant Street	Waltham	MA	(617) 555-3840
123	Philip	Chin	59 Pond Street	Atlanta	GA	(404) 555-2341
143	Julie	Jordan	144 Great Plain Avenue	Winchester	MA	(617) 555-7835
160	Robert	Breault	58 Cherry Street	Milton	MA	(617) 555-3099
184	Melissa	Espinoza	112 Apple Tree Way	Stow	MA	(508) 555-2319
191	Jeannette	Bertrand	209 Concord Street	Acton	MA	(508) 555-8138
195	Marc	Dill	89 Hancock Street	Milton	MA	(617) 555-2144
207	Jane	Francis	112 Hawthorne Drive	Concord	MA	(508) 555-9022

Using the Label style

The Label presentation style shows data as labels. With this style you can create mailing labels, business cards, name tags, index cards, diskette labels, file folder labels, and many other types of labels.

Mailing labels

 Rodrigo Guevara East Main Street Framingham, MA 01701	 Jeannette Bertrand 209 Concord Street Acton, MA 01720	 James Coleman 57 Heather Hill Drive Acton, MA 01720
 Joseph Barker 58 West Drive Bedford, MA 01730	 Irene Barletta 37 Gleason Street Bedford, MA 01730	 Robert Nielsen 55 Sargent Avenue Bedford, MA 01730
 Catherine Pickett 45 Appleton Road Bedford, MA 01730	 Sheila Romero 1 Oakview Terrace Bedford, MA 01730	 Doug Charlton 57 Webster Street Concord, MA 01742
 Scott Evans 10-A Sunrise Circle Concord, MA 01742	 Jane Francis 12 Hawthorne Drive Concord, MA 01742	 Jennifer Litton 17 Downing Street Concord, MA 01742

Business cards

<i>My company</i> Terry Lambert Administration 204 Page St. Canton, MA 94608 Phone: (617) 555-2246 Fax: (617) 555-3692	<i>My company</i> Terry Lambert Administration 204 Page St. Canton, MA 94608 Phone: (617) 555-2246 Fax: (617) 555-3692
<i>My company</i> Terry Lambert Administration 204 Page St. Canton, MA 94608 Phone: (617) 555-2246 Fax: (617) 555-3692	<i>My company</i> Terry Lambert Administration 204 Page St. Canton, MA 94608 Phone: (617) 555-2246 Fax: (617) 555-3692

Name tags

<i>Lynn Page</i> <i>Sales</i>	<i>Charles Crowley</i> <i>Human resources</i>
<i>Dana Burrill</i> <i>Product development</i>	<i>William Caruso</i> <i>Finance</i>

Specifying label properties

If you choose the Label style, you are asked to specify the properties for the label after specifying the data source. You can choose from a list of predefined label types or enter your own specifications manually.

Where label definitions come from

PowerBuilder gets the information about the predefined label formats from a preferences file.

Platform considerations

On Windows This preferences file is called PBLAB60.INI.

On Macintosh This preferences file is called PowerBuilder Label Preferences.

On UNIX This preferences file is called .PBLAB60.INI. It must be copied to each user's home directory from the PowerBuilder installation directory.

Using the N-Up style

The N-Up style presents two or more rows of data next to each other. It is similar to the Label style in that you can have information from several rows in the database across the page. However, the information is not meant to be printed on labels. The N-Up presentation style is useful if you have periodic data; you can set it up so each period repeats in a row.

After you select a data source, you are asked how many rows to display across the page.

For each column in the data source, PowerBuilder defines n columns in the DataWindow object (column_1 to column_ n), where n is the number of rows you specified.

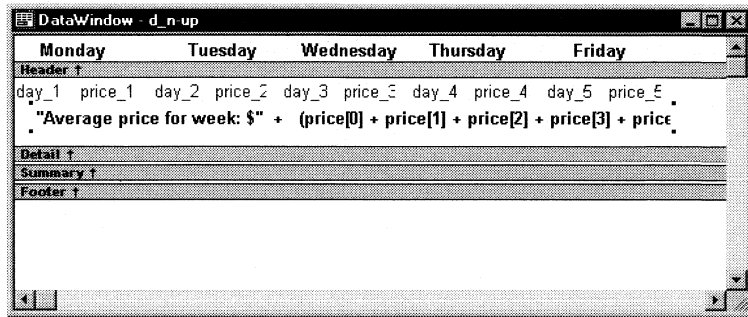
Table example

For a table of daily stock prices, you can define the DataWindow object as five across, so each row in the DataWindow object displays five days' prices (Monday through Friday). Here is a table with two columns that record the closing stock price each day for three weeks (this table is not in the Powersoft Demo Database):

Day	Price	Day	Price
1/31/96	62.00	2/10/96	68.00
2/1/96	63.00	2/11/96	67.00
2/2/96	66.00	2/14/96	67.00
2/3/96	65.00	2/15/96	71.00
2/4/96	62.00	2/16/96	70.00

Day	Price	Day	Price
2/7/96	65.00	2/17/96	72.00
2/8/96	69.00	2/18/96	75.00
2/9/96	66.00		

In the following n-up DataWindow object, 5 was selected as the number of rows to display across the page, so each line in the DataWindow object shows five days' stock prices. A computed field was added to get the average closing price in the week:

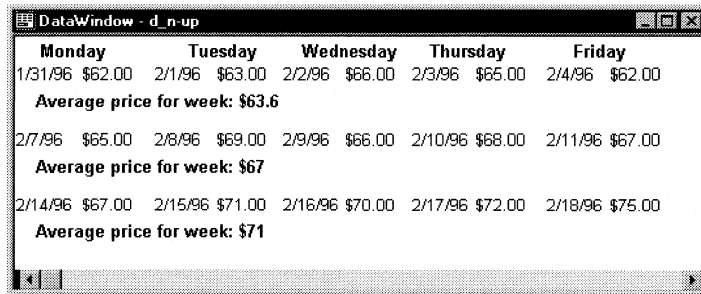


About computed fields in n-up DataWindow objects

You use subscripts, such as `price[0]`, to refer to particular rows in the detail band in n-up DataWindow objects.

FOR INFO For more information, see Chapter 15, "Enhancing DataWindow Objects".

Here is the DataWindow object in preview:



Another way to get multiple-column DataWindow objects

In an n-up DataWindow object, the data is displayed across and then down. If you want your data to go down the page and then across in multiple columns, as in a phone list, you should create a standard tabular DataWindow object, then specify newspaper columns.

FOR INFO For more information on newspaper columns, see Chapter 15, "Enhancing DataWindow Objects".

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.

FOR INFO For more about the Group presentation style, see Chapter 17, "Filtering, Sorting, and Grouping Rows".

Using the Composite presentation style

The Composite presentation style allows you to combine multiple DataWindow objects in the same object. It is particularly handy if you want to print more than one DataWindow object on a page.

FOR INFO For more about the Composite presentation style, see Chapter 19, "Using Nested Reports".

Using the Graph and Crosstab presentation styles

In addition to the (preceding) text-based presentation styles, PowerBuilder provides two styles that allow you to display information graphically: Graph and Crosstab.

FOR INFO For more information about these two presentation styles, see Chapter 20, "Working with Graphs", and Chapter 21, "Working with Crosstabs".

Using the OLE 2.0 presentation style

The OLE presentation style lets you link or embed an OLE object in a DataWindow object.

FOR INFO For information about the OLE 2.0 presentation style, see Chapter 23, "Using OLE in a DataWindow Object".

Using the RichText presentation style

The RichText presentation style lets you combine input fields that represent database columns with formatted text.

FOR INFO For more information about the RichText presentation style, see Chapter 22, "Working with Rich Text".

Choosing DataWindow object-wide options

You can preset the default options that PowerBuilder uses in creating the initial draft of a DataWindow object. These options include the default colors and borders that PowerBuilder uses.

DataWindow generation options are for styles that use the banded workspace, which include Freeform, Grid, Label, N-Up, Tabular, Group, and Crosstab. PowerBuilder maintains a separate set of options for each of these styles.

When you are working on the DataWindow object in the workspace, you can override the values set with DataWindow object generation options.

❖ **To specify default colors and borders for a new DataWindow object:**

- 1 In the New DataWindow dialog box, click Options.

The DataWindow Options property sheet displays.

In the painter

If you are already in the DataWindow painter, you can select Design>Options from the menu bar to display the property sheet. Changes you make will apply to new DataWindow objects. They will not apply to the open DataWindow object.

- 2 Select the Generation tab page if it is not on top.
- 3 Select the presentation style you want from the Presentation Style dropdown listbox.

The values for properties shown on the page are for the currently selected presentation style.

- 4 Change one or more of the following properties:

Property	Meaning for the DataWindow object
Background color	The default color for the background
Text border and color	The default border and color used for labels and headings
Column border and color	The default border and color used for data values

Property	Meaning for the DataWindow object
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

About color selections

If you select WndBkrnd, AppWrkSpc, ButtonFace, or WndText from the Color dropdown listbox, the DataWindow object will use the colors a particular user has specified for his windowing environment.

5 Click OK.

You return to the New DataWindow dialog box.

Your choices are saved

PowerBuilder saves your generation option choices as the defaults to use when creating a DataWindow object with the same presentation style.

Defining the 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.

Data source can also refer to where the data comes from, such as a SQL Anywhere data source (meaning a database file) or a dBASE data source (meaning a DBF file). *Connecting to Your Database* uses the term data source this second way.

About stored procedures

The Stored Procedure data source icon displays only if the DBMS you are currently connected to supports stored procedures that return result sets.

How to choose the data source

If the data is in the database

If the data for the DataWindow object will be retrieved from a database, choose one of the following data sources:

Data source	Use when
Quick Select	The data is from a single table (or from tables that are related through foreign keys) and you only need to choose columns, selection criteria, and sorting (you don't need to specify grouping, computed columns, and so on)
SQL Select	You want more control over the SQL SELECT statement generated for the data source <i>or</i> your data is from tables that are not connected through a key
Query	The data has been defined as a query
Stored Procedure	The data is defined in a stored procedure

If the data is not in a database

Select the External data source if:

- ◆ The DataWindow object will be populated from a script
- ◆ 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)

You can also use an ODBC driver to access data from a text or dBASE file. On Windows, PowerBuilder ships ODBC drivers for both text and dBASE files.

FOR INFO For more information, see *Connecting to Your Database*.

After you choose a data source in the New DataWindow dialog box and click OK, you specify the data. The data source you choose determines which dialog box displays and how you define the data.

Why use a DataWindow if the data is not from a DBMS

Even when the data is not coming from the database, there are many times when you want to take advantage of the intelligence of a DataWindow object:

- ◆ **Data Validation** You have full access to validation rules for data
- ◆ **Display Formats** You can use any existing display formats to present the data, or create your own
- ◆ **Edit Styles** You can use any existing edit styles, such as radio buttons and edit masks, to present the data, or create your own

Using Quick Select

The easiest way to define a data source is using Quick Select. With Quick Select, you can choose columns from one table or from multiple tables if they are joined through foreign keys. After you choose the columns, you can specify:

- ◆ Whether you want to sort the retrieved rows
- ◆ Retrieval criteria for the rows

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

To use these options, choose SQL Select as your data source.

❖ **To define the data source using Quick Select:**

- 1 Click Quick Select in the New DataWindow Object dialog box, select a presentation style, and click OK.

The Quick Select dialog box displays. The Tables box lists tables and views in the current database.

Which tables and views display

The DBMS determines what tables and views display. For some DBMSs, all tables and views display, whether or not you have authorization. Then if you select a table or view you aren't authorized to access, the DBMS issues a message.

For ODBC databases, the tables and views that display depend on the driver for the data source. SQL Anywhere does not restrict the display; so all tables and views display, whether or not you have authorization.

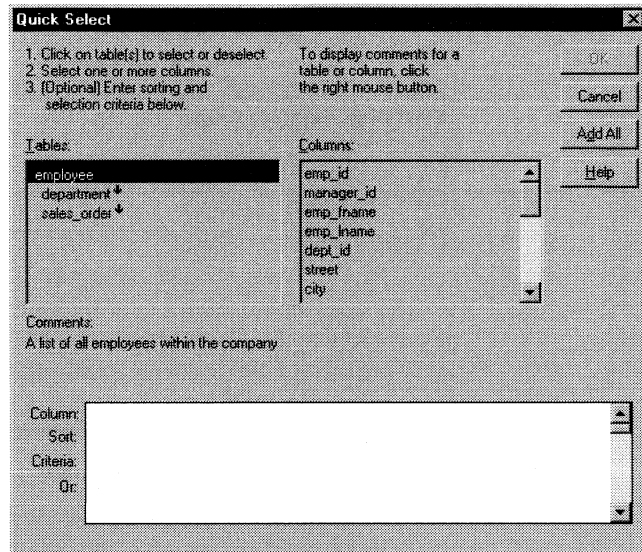
To display a comment about a table, position the pointer on the table and click the right mouse button *or* select the table.

On Macintosh

To display a comment about a table on the Macintosh, position the pointer on the table, and press the CONTROL key while you click the mouse button *or* select the table.

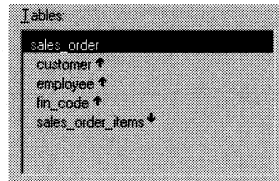
2. Select a table containing the data you want to use.

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:



Meaning of the up and down arrows

An arrow displays next to a table to indicate its relationship to the selected table. The arrow always points in the *many* direction of the relationship—toward the selected table (up) if the selected table contains a foreign key in the relationship and away from the selected table (down) if the selected table contains a primary key in the relationship:



In this example, a foreign key in the sales_order table is mapped to the primary key in the customer, employee, and fin_code tables. The sales_order_items table contains a foreign key mapped to the primary key in the sales_order table.

- 3 Select any additional tables containing data you want to use.

The column names of selected tables display in the Columns box. If you select more than one table, the column names are identified as:

tablename.columnname

For example, department.dept_name and employee.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.

- 4 Select the columns you want to use:
 - ◆ To add a column, select it in the Columns box.
 - ◆ To add all columns, 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.

On Macintosh

To view comments that describe a table or column on the Macintosh, position the pointer on the table or column name, hold down the CONTROL key, and press the mouse button.

The selected columns display at the bottom of the dialog box.

- 5 Use the grid to reorder columns, sort rows before you retrieve data, and specify what data to retrieve:

- ◆ To reorder a column, drag the column name where you want it.
- ◆ To specify the sorting of rows before retrieval, select the sorting order in the Sort row.

FOR INFO For information, see "Specifying sorting criteria" on page 436.

- ◆ To specify what data to retrieve, enter expressions in the Criteria row.

FOR INFO For information, see "Specifying selection criteria" on page 436.

- 6 Click OK.

The Quick Select dialog box closes. You either go to the DataWindow painter workspace or into Preview mode, based on the Preview When Built checkbox in the New DataWindow dialog box.

FOR INFO For more information, see Chapter 15, "Enhancing DataWindow Objects".

In some DataWindow styles, PowerBuilder prompts you for additional information.

The DataWindow painter workspace displays.

Quick Select and retrieval arguments

When you use Quick Select to define the data, you cannot define retrieval arguments for the SELECT statement that are supplied during execution. If you decide later that you want to use retrieval arguments, you can define them from the DataWindow painter workspace by modifying the data source.

FOR INFO For more information, see Chapter 15, "Enhancing DataWindow Objects".

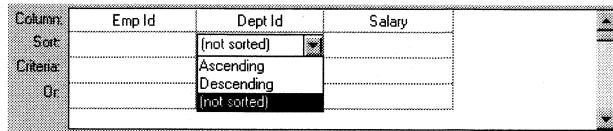
Specifying sorting criteria

In the grid at the bottom of the Quick Select dialog box, you can specify if you want the retrieved rows to be sorted. As you specify sorting criteria, PowerBuilder builds an ORDER BY clause for the SELECT statement.

❖ **To sort retrieved rows on a column:**

- 1 Click in the Sort row for the column you want to sort on.

PowerBuilder displays a dropdown listbox:



- 2 Select the sorting order for the rows: Ascending or Descending.

Multilevel sorts

You can specify as many columns for sorting as you want. PowerBuilder processes the sorting criteria left to right in the grid: the first column with Ascending or Descending specified becomes the highest level sorting column, the next column with Ascending or Descending specified becomes the next level sorting column, and so on.

If you want to do a multilevel sort that doesn't 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 and Marketing who make more than \$50,000.

As you specify selection criteria, PowerBuilder builds a WHERE clause for the SELECT statement.

❖ **To specify selection criteria:**

- 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 criteria, 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 repository (that is, the association was made in the Database painter), the edit style is used in the grid—except that dropdown listboxes are used for columns with code tables and columns using the CheckBox and RadioButton edit styles.

SQL operators
supported in Quick
Select

You can use these SQL relational operators in the retrieval criteria:

Operator	Meaning
=	Is equal to (default operator)
>	Is greater than
<	Is less than
< >	Does not equal
> =	Is greater than or equal to
< =	Is less than or equal to
LIKE	Matches this pattern
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.

The LIKE and NOT LIKE operators 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 7-letter names that begin with Good.

The IN and NOT IN operators Use IN to compare and include a value to a set of values; use NOT IN to compare and include values that are not in a set of values. For example, the following clause selects all employees in department 100, 200, or 500:

```
SELECT * from employee
WHERE dept_id IN (100, 200, 500)
```

Using NOT IN would exclude employees in those departments.

Connection operators

You can use the OR and AND logical operators to connect expressions.

PowerBuilder makes some assumptions based on how you specify selection criteria. When you specify:

- ◆ **Criteria for more than one column on one line** PowerBuilder assumes a logical AND between the criteria. A row from the database is retrieved if *all* criteria in the line are met.
- ◆ **Two or more lines of selection criteria** PowerBuilder assumes a logical OR. A row from the database is retrieved if the criteria in *any* of the lines is met.

By default, criteria expressions in one line are logically ANDed; expressions in different lines are logically ORed. To override these defaults, begin an expression with the AND or OR operator:

Operator	Meaning
OR	The row is selected if one expression OR another expression is true
AND	The row is selected if one expression AND another expression are true

This technique is particularly handy when you want to retrieve a range of values in a column. See example 6 below.

SQL expression examples

Example 1 The following expression in the grid retrieves information for employees whose salaries are less than \$50,000:

Column:	Emp Id	Dept Id	Salary
Sort:			
Criteria:			<50000
Or:			

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE salary < 50000
```

Example 2 The following expression in the grid retrieves information for employees who belong to department 100:

Column:	Emp Id	Dept Id	Salary
Sort:			
Criteria:		100	
Or:			

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id = 100
```

Example 3 The following expressions in the grid retrieve information for employees whose employee ID is greater than 300 *and* whose salary is less than \$50,000:

Column:	Emp Id	Dept Id	Salary
Sort:			
Criteria:	>300		<50000
Or:			

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE emp_id >300 AND salary <50000
```

Example 4 The following expressions in the grid retrieve information for employees who either belong to:

- ◆ Department 100 *and* have a salary less than \$50,000

or

- ◆ A department whose ID is greater than 300, no matter what their salary

Column:	Emp id	Dept Id	Salary
Sort:			
Criteria:		100	<50000
Or:		>300	

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE (dept_id = 100 AND salary < 50000)
OR dept_id > 300
```

Example 5 The following expression in the grid retrieves information for employees who are in department 100 *or* 200 *or* 500:

Column:	Emp id	Dept Id	Salary
Sort:			
Criteria:		IN (100,200,500)	
Or:			

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE dept_id IN (100, 200, 500)
```

Example 6 The following expressions in the grid retrieve information for employees who have an employee ID from 500 to 1000 and a salary from \$30,000 to \$50,000:

Column:	Emp id	Dept Id	Salary
Sort:			
Criteria:	>= 500		>= 30000
Or:	AND <= 1000		AND <= 50000

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_id, dept_id, salary
FROM employee
WHERE (emp_id >= 500 AND emp_id <= 1000)
AND (salary >= 30000 AND salary <= 50000)
```

Example 7 The following expressions in the grid retrieve information for employees who have last names that begin with C or G:

Column:	Emp Last Name	Emp First Name	Salary
Sort:			
Criteria:	LIKE C%		
Or:	LIKE G%		

The SELECT statement that PowerBuilder creates is:

```
SELECT emp_last_name, emp_first_name, salary
FROM employee
WHERE emp_last_name LIKE 'C%'
OR emp_last_name LIKE 'G%'
```

Providing SQL
functionality to users

You can allow your users to specify selection criteria in a DataWindow object using these techniques during execution:

- ◆ You can automatically pop up a window prompting users to specify criteria each time just before data is retrieved.
- FOR INFO For more information, see Chapter 15, "Enhancing DataWindow Objects".
- ◆ You can place the DataWindow object in query mode using the Modify function.

FOR INFO For more information, see *Application Techniques*.

Using SQL Select

In specifying your data for a DataWindow object, you have the most freedom 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 INFO For more information about queries, see "Defining queries" on page 462.

❖ To define the data using SQL Select:

- 1 Click SQL Select in the New DataWindow dialog box and click OK.
The Select Tables dialog box displays.
- 2 Select the tables and/or views that you will use in the DataWindow object.
FOR INFO For more information, see "Selecting tables and views" on page 443.
- 3 Select the columns to be retrieved from the database.
FOR INFO For more information, see "Selecting columns" on page 444.
- 4 Join the tables if you have selected more than one.
FOR INFO For more information, see "Joining tables" on page 447.
- 5 Select retrieval arguments if appropriate.
FOR INFO For more information, see "Using retrieval arguments" on page 448.
- 6 Limit the retrieved rows with WHERE, ORDER BY, GROUP BY, and HAVING criteria, if appropriate.
FOR INFO For more information, see "Specifying selection, sorting, and grouping criteria" on page 450.
- 7 If you want to eliminate duplicate rows, select Distinct from the Options menu. This adds the DISTINCT keyword to the SELECT statement.
- 8 Click the SQL Select button on the PainterBar.
The DataWindow painter workspace displays (unless you need to provide more information, in which case you are prompted for it).

Selecting tables and views

After you have chosen SQL Select in the New DataWindow dialog box, chosen a presentation style, and clicked OK, the Select Tables dialog box displays in the current workspace.

What tables and views display depends on the DBMS. For some DBMSs, all tables and views display, whether or not you have authorization. Then if you select a table or view you aren't authorized to access, the DBMS issues a message.

For ODBC databases, the tables and views that display depend on the driver for the data source. SQL Anywhere does not restrict the display; so all tables and views display, whether or not you have authorization.

The SQL toolbox also displays at the bottom of the workspace; you will use the tabs in the SQL toolbox to specify the SQL Select statement in more detail.

❖ 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 in the workspace 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 in the Select painter workspace. You can move or size each table to fit the workspace as needed.

Specifying what is displayed

You can display the label and data type of each column in the tables in the workspace (the label information comes from the Powersoft repository). If you need more space, you can choose to hide this information. You can also choose to hide the SQL toolbox to give you more room to see the tables.

❖ To hide or display comments, data types, labels, and the SQL toolbox:

- 1 Position the pointer on any unused area of the workspace and select Show from the popup menu.

A cascading menu displays.

- 2 Select or clear Datatypes, Labels, Comments, or SQL Toolbox as needed.

Shortcut

You can click the Toolbox button in the PainterBar to toggle the display of the SQL toolbox.

Colors in the Select painter

The colors used by the Select painter to display the workspace background and table information are specified in the Database painter. You can also set colors for the text and background components in the table header and detail areas.

FOR INFO For more information about specifying colors in the Database painter, see "Modifying database preferences" on page 347.

Adding and removing tables and views

At any time, you can add tables and views to your workspace:

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 popup menu and select Close
Remove all tables and views	Select Design>Undo All from the menu bar

You can also remove individual tables and views from the workspace, or clear them all at once and begin selecting a new set of tables.

How PowerBuilder joins tables

If you select more than one table in the Select painter workspace, PowerBuilder joins columns based on their key relationship.


FOR INFO For information about joins, see "Joining tables" on page 447.

Selecting columns

You can click each column you want to include from the table representations in the workspace. PowerBuilder highlights selected columns and places them in the Selection List at the top of the Select painter.

❖ **To reorder the selected columns:**

- ◆ Drag a column in the Selection List with the mouse. Release the mouse button when the column is in the proper position in the list:

Selection List:  emp_id | emp_lname | emp_fname | dept_id

❖ **To select all columns from a table:**

- ◆ Move the pointer to the table name and select Select All from the popup menu.

❖ **To include computed columns:**

- 1 Click the Compute tab in the SQL toolbox at the bottom of the workspace.

Each row in the Compute tab is a place for entering an expression that defines a computed column.

- 2 Enter an expression for the computed column—for example:

```
salary / 12
```

or a function supported by your DBMS (the following is a SQL Anywhere function):

```
substr("employee"."emp_fname",1,2)
```

You can display the popup menu for any row in the Compute tab. Using the popup menu, you can select and paste columns, functions, and arguments (if you have created any) into the expression:

- ◆ Names of columns in the tables used in the report, form, or pipeline
- ◆ Any retrieval arguments you have specified
- ◆ Functions supported by the DBMS

About these functions

The functions listed here are functions provided *by your DBMS*. They are not PowerBuilder functions. (This is because you are now defining a SELECT statement that will be sent to your DBMS for processing.)

- 3 Click the next row to define another computed column.

or

Click another tab to make additional specifications.

or

Click the SQL Select button to return to the workspace.

PowerBuilder adds the computed columns to the list of columns you have selected.

About defining computed columns here

Computed columns you define in the Select painter are added to the SQL statement and used by the DBMS to retrieve the data. The expression you define here follows your DBMS's rules.

You can also choose to define computed fields, which are created and processed dynamically by PowerBuilder after the data has been retrieved from the DBMS. There are advantages to doing this. For example, work is offloaded from the database server, and the computer fields update dynamically as data changes in the DataWindow object (though if you have many rows, this updating can result in slower performance).

FOR INFO For more information, see Chapter 15, "Enhancing DataWindow Objects".

Displaying the underlying SQL statement

As you specify the data for the DataWindow object in the Select painter, PowerBuilder is generating a SQL SELECT statement. It is this SQL statement that will be sent to the DBMS when you retrieve data into the DataWindow 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 in the bottom of the workspace.

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.

❖ To edit the SELECT statement:

- 1 Select Design>Convert to Syntax from the menu bar.
PowerBuilder displays the SELECT statement in a text window.
- 2 Edit the SELECT statement.

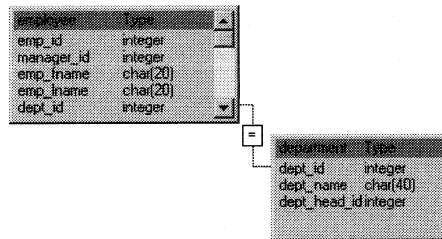
- 3 Return to a painter:
 - ◆ **Select painter.** Select Design>Convert to Graphics from the menu bar.
 - ◆ **DataWindow painter.** Click the SQL Select button.

Joining tables

If the DataWindow object will contain data from more than one table, you should join the tables on their common columns. If you have selected more than one table, PowerBuilder joins columns based on their key relationship:

- ◆ **A primary/foreign key relationship** PowerBuilder automatically joins them.
- ◆ **No key relationship** PowerBuilder makes its best guess and tries to join tables based on common column names and types.

PowerBuilder links joined tables in the Select painter workspace:

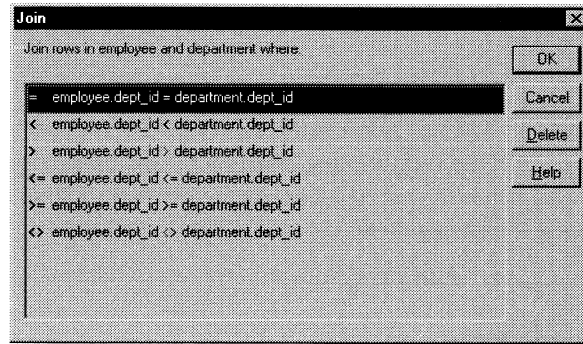


PowerBuilder joins can differ depending on the order in which you select the tables, and sometimes the PowerBuilder best-guess join is incorrect. So you may need to delete a join and manually define a join.

- ❖ **To delete a join:**
 - 1 Click the join operator connecting the tables.
The Join dialog box displays.
 - 2 Click Delete.
- ❖ **To join tables:**
 - 1 Click the Join button.
 - 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 workspace.

The Join dialog box displays:



- 4 Select the join operator you want and click OK.

If your DBMS supports outer joins, outer join options also display in the Join dialog box.

FOR INFO For more about outer joins, see your DBMS documentation.

Using retrieval arguments

If you know which rows will be retrieved into the DataWindow object when you preview it during execution—that is, if you can fully specify the SELECT statement without having to provide a variable—you don't need to specify retrieval arguments.

Adding retrieval arguments

If you decide later that you need arguments, you can return to the Select painter from the DataWindow painter workspace to define the arguments.

Defining retrieval arguments in the DataWindow painter

You can select Rows>Prompt for Criteria from the menu bar. A dialog box will let you identify the column users should be prompted for. This, like the Retrieval Arguments prompt, calls the Retrieve function.

FOR INFO See Chapter 15, "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 textbox. You must pass that information to the SELECT statement as an argument when you preview the report during execution.
- ◆ Retrieving all rows from a table for a department selected from a dropdown listbox. The department is passed as an argument when you preview the report during execution.

Using retrieval arguments during execution

If a DataWindow object has retrieval arguments, call the Retrieve PowerScript function of the DataWindow control to retrieve data during execution and pass the arguments in the function.

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

❖ To define retrieval arguments:

- 1 In the Select painter, select Design>Retrieval Arguments from the menu bar.
- 2 Enter a name and select a data type 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 dropdown listbox 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)
```

PowerBuilder passes the appropriate comma-delimited list to the function (such as *100, 200, 500* if $x[1] = 100$, $x[2] = 200$, and $x[3] = 500$).

When building the SELECT statement, you reference the retrieval arguments in the WHERE or HAVING clause, as described in the next section.

Specifying selection, sorting, and grouping criteria

In the SELECT statement associated with a DataWindow object, you can add selection, sorting, and grouping criteria that are added to the SQL Statement and processed by the DBMS as part of the retrieval.

To do this	Use this clause
Limit the data that is retrieved from the database	WHERE
Sort the retrieved data before it is brought into the DataWindow object	ORDER BY
Group the retrieved data before it is brought into the DataWindow object	GROUP BY
Limit the groups specified in the GROUP BY clause	HAVING

Dynamically selecting, sorting, and grouping data

Selection, sorting, and grouping criteria that you define in the Select painter are added to the SQL statement and processed by the DBMS as part of the retrieval. You can also define selection, sorting, and grouping criteria that are created and processed dynamically by PowerBuilder *after* data has been retrieved from the DBMS.

FOR INFO For more information, see Chapter 17, "Filtering, Sorting, and Grouping Rows".

Referencing retrieval arguments

If you have defined retrieval arguments, you will reference the argument(s) 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 tab 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 in the SQL Toolbox.

Each row in the Where tab is a place for entering an expression that limits the grouping of rows.

- 2 Click in the first row under Column to display columns in a dropdown list.

or

Select Columns from the popup menu.

- 3 Select the column you want to use in the left-hand side of the expression.

The equality (=) operator displays in the Operator column.

Using a function or retrieval argument in the expression

To use a function, select Functions from the popup menu and click a listed function. These are the functions provided by the DBMS.

To use a retrieval argument, select Arguments from the popup 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 popup menu.
- ◆ Paste a value from the database by selecting Value from the popup 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 popup menu. In the Nested Select dialog box you can define a nested SELECT statement. Click Return when you have finished.

- 6 Continue to define additional WHERE expressions as needed.

For each additional expression, select a logical operator (AND or OR) to connect the multiple boolean expressions into one expression that PowerBuilder evaluates as true or false to limit the rows that are retrieved.

- 7 Define sorting (Sort tab), grouping (Group tab), and limiting (Having tab) criteria as appropriate.

- 8 Click the SQL Select button to return to the DataWindow painter workspace.

Defining ORDER BY criteria

You can sort the rows that are retrieved into the DataWindow object by specifying columns that correspond to the ORDER BY clause in the SELECT statement.

For example, if you are retrieving information about employees, you can sort on department, and then within each department, you can sort on employee ID.

❖ **To define ORDER BY criteria:**

- 1 Click the Sort tab in the SQL Toolbox.

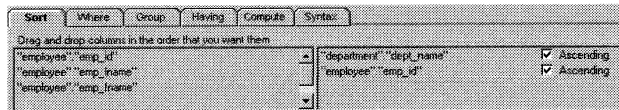
The columns you selected display in the order of selection. You may need to scroll to see your selections.

- 2 Drag the first column you want to sort on to the right side of the Sort tab.

This specifies the column for the first level of sorting. By default, the column is sorted in ascending order. To specify descending order, clear the Ascending checkbox.

- 3 Continue to specify additional columns for sorting in ascending or descending order as needed.

You can change the sorting order by dragging the selected column names up or down. With the following sorting specification, rows will be sorted first by department name, then by employee ID:



- 4 Define limiting (Where tab), grouping (Group tab), and limiting groups (Having tab) criteria as appropriate.
- 5 Click the SQL Select button to return to the DataWindow painter workspace.

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. Here is the corresponding SELECT statement:

```
SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
```

If you specify this with the Employee table in the Powersoft Demo Database, you will get five rows back, one for each department:

Department ID	Sum(salary)	Avg(salary)
100	\$1,292,198.19	\$58,736.28
200	\$919,428.00	\$48,390.95
300	\$535,500.00	\$59,500.00
400	\$698,250.75	\$43,640.67
500	\$315,729.88	\$35,081.10

FOR INFO For more about GROUP BY, see your DBMS documentation.

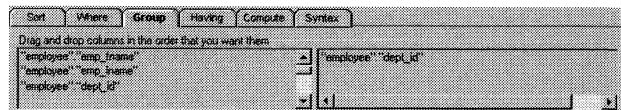
❖ **To define GROUP BY criteria:**

- 1 Click the Group tab in the SQL Toolbox.

The columns in the tables you selected display in the left side of the Group tab. You may need to scroll to see your selections.

- 2 Drag the first column you want to group on to the right side of the Group tab.

This specifies the column for grouping. Columns are grouped in the order they are displayed in the right side of the Group tab. In the following, the DataWindow object will be grouped by department ID:



- 3 Continue to specify additional columns for grouping within the first grouping column as needed.

To change the grouping order, drag the column names in the right side to the positions you want.

- 4 Define sorting (Sort tab), limiting (Where tab), and limiting groups (Having tab) criteria as appropriate.

- 5 Click the SQL Select button to return to the DataWindow painter workspace.

Defining HAVING criteria

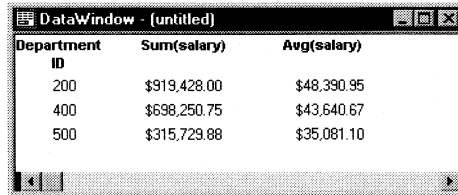
If you have defined groups, you can define HAVING criteria to restrict the retrieved groups. For example, if you group employees by department, you can restrict the retrieved groups to departments whose employees have an average salary of less than \$50,000. This corresponds to:

```

SELECT dept_id, sum(salary), avg(salary)
FROM employee
GROUP BY dept_id
HAVING avg(salary) < 50000

```

If you specify this with the Employee table in the Powersoft Demo Database, you will get three rows back, because there are three departments that have average salaries less than \$50,000:



Department ID	Sum(salary)	Avg(salary)
200	\$919,428.00	\$48,390.95
400	\$698,250.75	\$43,640.67
500	\$315,729.88	\$35,081.10

❖ **To define HAVING criteria:**

- ◆ Click the Having tab in the bottom of the workspace.

Each row in the Having tab is a place for entering an expression that limits which groups are retrieved. For information on how to define criteria on the Having tab, see the procedure in "Defining WHERE criteria" on page 451.

Using Query

When you choose Query as the data source, you select a predefined SQL SELECT statement (a **query**) as specifying the data for your DataWindow object.

❖ **To define the data using Query:**

- 1 Click Query in the New DataWindow dialog box, select a presentation style, and click OK.

The Select Query dialog box displays, listing all queries that have been defined in the current PowerBuilder library.

- 2 Select a query and click OK.

The DataWindow painter workspace displays (with some presentation styles, you need to provide additional information before going to the workspace).

FOR INFO To learn how to create queries, see "Defining queries" on page 462.

Using External

If the data for the DataWindow object is not coming from a database (either through a native Powersoft database interface or through ODBC), specify External as the data source. You then specify the data columns and their types so PowerBuilder can build the appropriate DataWindow object to hold the data. These columns make up the **result set**. PowerBuilder places the columns you specified in the result set in the DataWindow object.

❖ **To define the data using External:**

- 1 Click External in the New DataWindow dialog box, select a presentation style, and click OK.

The Result Set Description dialog box displays for you to specify the first column in the result set.

- 2 Enter the name and type of the column.

Available data types are listed in the dropdown listbox.

- 3 Click Add to enter the name and type of any additional columns you want in the result set.

- 4 Click OK when you have added all the columns you want.

What you do next

In a script, you will need to tell PowerBuilder how to get data into the DataWindow object in your application. Typically, you will import data during execution using a PowerScript Import function (such as ImportFile and ImportString) or do some data manipulation and use the SetItem function to populate the DataWindow.

FOR INFO For more about these functions, see the *PowerScript Reference*.

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 Click the Preview button in the PainterBar.

If the button isn't visible, turn toolbar text off.

- 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 dropdown listbox (either TXT or DBF files).

- 4 Enter the name of the import file and click OK.

Alternatively, you can select the name from the file list. Use the Drives dropdown listbox and the Directories box as needed to display the list of files that includes the one you want.

Using Stored Procedure

A **stored procedure** is a set of precompiled and preoptimized SQL statements that performs some database operation. Stored procedures reside where the database resides, and you can access them as needed.

You can specify a stored procedure as the data source for a DataWindow object if your DBMS supports stored procedures.

FOR INFO For information on support for stored procedures, see your database documentation.

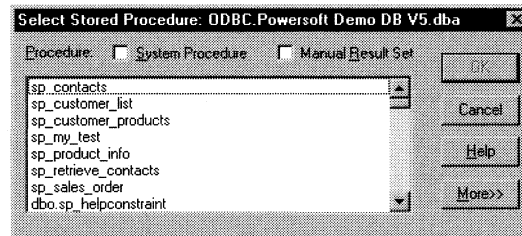
If the Stored Procedure icon is not displayed

The icon for the Stored Procedure data source displays in the New DataWindow dialog box only if the database to which you are connected supports stored procedures.

❖ To define the data using Stored Procedure:

- 1 Select Stored Procedure in the New DataWindow dialog box.
- 2 Select a presentation style and click OK.

The Select Stored Procedure dialog box displays a list of the stored procedures in the current database:



- 3 Select a stored procedure from the list.

To list system procedures, select the System Procedure checkbox.

To see the syntax of the selected stored procedure, click More. If this is not the stored procedure you want, select a different one.

- 4 Specify how you want the result set description built:

- ◆ **Build the result set description automatically** Clear the Manual Result Set checkbox.

PowerBuilder executes the stored procedure and builds the result set description for you. You go to the DataWindow painter workspace with the columns placed in the DataWindow object.

- ◆ **Define the result set description manually** Select the Manual Result Set checkbox and click OK.

In the Result Set Description dialog box:

- ◆ Enter the name and type of the first column in the result set.
- ◆ To Add additional columns, click Add.
- ◆ To define retrieval arguments or change to another stored procedure, click More.

Your preference is saved

PowerBuilder records your preference for building result set descriptions for stored procedure DataWindow objects in the variable `Stored_Procedure_Build` in the PowerBuilder initialization file. If this variable is set to 1, PowerBuilder will automatically build the result set; if the variable is set to 0, you will be prompted to define the result set description.

- 5 When you have defined the entire result set, click OK.

You go to the DataWindow painter workspace with the columns specified in the result set placed in the DataWindow object.

FOR INFO For information about defining retrieval arguments for DataWindow objects, see Chapter 15, "Enhancing DataWindow Objects".

❖ **To edit the result set description:**

- 1 Click SQL Select in the PainterBar.

or

Select Design>Data Source from the menu bar.

- 2 Click More to edit the Execute statement, select another stored procedure, or add arguments.
- 3 When you have defined the entire result set, click OK.

You return to the DataWindow painter workspace with the columns specified in the result set placed in the DataWindow object.

FOR INFO For information about defining retrieval arguments for DataWindow objects, see Chapter 15, "Enhancing DataWindow Objects".

Generating and saving a DataWindow object

Once you have selected a presentation style and data source, PowerBuilder generates the DataWindow object and takes you to the DataWindow painter workspace (with some presentation styles, you are prompted for additional information before you go to the workspace).

When generating the DataWindow object, PowerBuilder may use information from a facility called the **Powersoft repository**. If repository information is available, PowerBuilder uses it.

About the Powersoft repository and DataWindow objects

The Powersoft repository is a set of system tables maintained by the Database painter. It contains information about database tables and columns. Repository information extends database definitions by recording information that is relevant to using database data in screens and reports.

For example, labels and headings you defined for columns in the Database painter are used in the generated DataWindow object. Similarly, if you associated an edit style with a column in the Database painter, that edit style is automatically used for the column in the DataWindow object.

When generating a DataWindow object, PowerBuilder uses the following information from the repository:

For	PowerBuilder uses
Tables	Fonts specified for labels, headings, and data
Columns	Text specified for labels and headings Display formats Validation rules Edit styles

If there is no repository 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 workspace. 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 repository, you can change it in the DataWindow painter workspace. The changes you make in the DataWindow painter apply only to the DataWindow object you are working on.

The advantage of the repository is that it saves time and ensures consistency. You only have to specify the information once, in the database. Since PowerBuilder uses the information whenever anyone creates a new DataWindow object with the data, it's more likely that the appearance and names of data items will be consistent.

FOR INFO For more information about the repository, see Chapter 13, "Managing the Database", and the Appendix, "The Powersoft Repository".

Saving the DataWindow object

When you have created a DataWindow object and are in the DataWindow painter workspace, you should save the DataWindow object. The first time you save it you will give it a name. As you work, you should save your DataWindow object frequently so that you don't lose changes.

❖ To save the DataWindow object:

- 1 Select File>Save from the menu bar.

If you have previously saved the DataWindow object, PowerBuilder saves the new version in the same library and returns you to the DataWindow painter workspace.

If you have not previously saved the DataWindow object, PowerBuilder 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 Click OK.

Naming the DataWindow object

The DataWindow object name can be any valid PowerBuilder identifier up to 40 contiguous characters.

FOR INFO For information about PowerBuilder identifiers, see the *PowerScript Reference*.

Defining queries

A query is a SQL SELECT statement created with the Query painter and saved with a name so that it can be used repeatedly as the data source for a DataWindow object.

Queries save time, because you specify all the data requirements just once. For example, you can specify the columns, which rows to retrieve, and the sorting order in a query. Whenever you want to create a DataWindow object using that data, simply specify the query as the data source.

❖ **To define a query:**

- 1 Click the Query button in the PowerBar.
- 2 Click New in the Select Query dialog box to define a new query.
- 3 Select tables and columns in the Select Tables dialog box.

You can also 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 INFO For more about defining the SELECT statement, see "Using SQL Select" on page 441.

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 In the Query painter, click the Preview button.

or

Select Design>Preview from the menu bar.

PowerBuilder retrieves the rows satisfying the currently defined query in a grid-style DataWindow object.

- 2 Manipulate the retrieved data as you do in the Data Manipulation painter.

You can sort and filter the data, but you cannot insert or delete a row or apply changes to the database.

FOR INFO For more about the Data Manipulation painter, see Chapter 13, "Managing the Database".

- 3 When you have finished previewing the query, click the Preview button to return to the Query painter workspace.

Saving the query

❖ To save a query:

- 1 Select File>Save from the menu bar.

or

Click the Save button.

If you have previously saved the query, PowerBuilder saves the new version in the same library and returns you to the Query painter workspace.

If you have not previously saved the query, PowerBuilder displays the Save Query dialog box.

- 2 Enter a name for the query in the Queries box (see "Naming the query" next).
- 3 (Optional) Enter comments to describe the query.

These comments display in the Library painter. It is a good idea to use comments to remind yourself and others of the purpose of the query.

- 4 Specify the library in which to save the query and click OK.

Naming the query

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

FOR INFO For information about PowerBuilder identifiers, see the *PowerScript Reference*.

A recommendation

When you name queries, use a unique name to identify each one. One approach is a 2-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.

Modifying a query

❖ **To modify a query:**

- 1 Click the Query button in the PowerBar.
The Select Query dialog box displays.
- 2 Select the query you want to modify and click OK.
- 3 Modify the query as needed.

What's next

After you have generated your DataWindow object, you will probably want to preview it to see how it looks. After that, you may want to enhance the DataWindow object in the DataWindow painter workspace before using it. PowerBuilder provides many ways for you to make a DataWindow object easier to use and more informative for users.

See Chapter 15, "Enhancing DataWindow Objects" next.

Enhancing DataWindow Objects

About this chapter

Before you put a DataWindow object into production, you will probably want to enhance it to make it easier to use and interpret data. You do that in the DataWindow painter workspace. This chapter describes basic enhancements you can make to a DataWindow object.

Contents

Topic	Page
Working in the DataWindow painter workspace	469
Previewing a DataWindow object	479
Modifying general DataWindow object properties	496
Reorganizing objects in a DataWindow object	512
Prompting for retrieval criteria in a DataWindow object	519
Adding objects to a DataWindow object	521
Positioning objects in a DataWindow object	538
Storing data in a DataWindow object	539
Retrieving rows as needed	541
Saving retrieved rows to disk	542
Controlling updates	543

Related topics

Other ways to enhance DataWindow objects are covered in later chapters:

Chapter	Explains how to
Chapter 16, "Displaying and Validating Data"	Specify display formats, edit styles, and validation rules for column data
Chapter 17, "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 18, "Highlighting Information in DataWindow Objects"	Highlight data by using conditional expressions to modify the properties of objects in DataWindow objects

Chapter	Explains how to
Chapter 19, "Using Nested Reports"	Place reports inside DataWindow objects
Chapter 20, "Working with Graphs"	Use graphs to visually present information retrieved in a DataWindow object
Chapter 21, "Working with Crosstabs"	Use crosstabs to present analyses of data retrieved in a DataWindow object

Working in the DataWindow painter workspace

Once you have specified your presentation style and data source, PowerBuilder generates a basic DataWindow object and places you in the DataWindow painter workspace where you can enhance the DataWindow object.

This section presents an overview of working in the workspace:

- ◆ "Understanding the DataWindow painter workspace" next
- ◆ "Using the DataWindow painter toolbars" on page 472
- ◆ "Using property sheets in the DataWindow painter" on page 473
- ◆ "Selecting objects in the DataWindow painter" on page 474
- ◆ "Using keyboard shortcuts in the DataWindow painter" on page 476
- ◆ "Resizing bands in the DataWindow painter workspace" on page 478
- ◆ "Using zoom in the DataWindow painter" on page 478
- ◆ "Undoing changes in the DataWindow painter" on page 478

Applicability of information

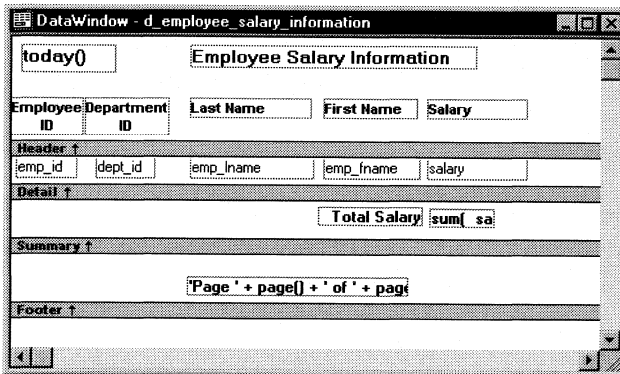
Some of the information presented in this section doesn't apply to all DataWindow object presentation styles.

Understanding the DataWindow painter workspace

With most presentation styles, the DataWindow painter workspace is divided into areas called **bands**. Each band corresponds to a section of the displayed DataWindow object.

A DataWindow object is divided into four bands: header, detail, summary, and footer. Each band is identified by a bar containing the name of the band above the bar and an arrow pointing to the band.

Here is the workspace for a tabular DataWindow object:



Band	Used to display
Header	Information at the top of every screen or page, such as the name of the report or current date
Detail	Data from the database or other data source
Summary	Summary information that displays after all the data, such as totals and counts
Footer	Information displayed at the bottom of every page or screen, such as page number and page count

About the header band

The header band contains heading information that is displayed at the top of every screen or page.

When PowerBuilder generates the basic DataWindow object, the presentation style determines the contents of the header band:

- ◆ If the presentation style is Tabular, Grid, or N-Up, the headings defined for the columns in the Database painter display in the header band and the columns display on a single line across the detail band
- ◆ If the presentation style is Freeform, the header band is empty and labels display in the detail band next to each column

You can specify additional heading information (such as a date) in the header band and can include pictures, graphic objects, and color to enhance the appearance of the band.

Displaying the current date

To include the current date in the header, you place a computed field that uses the Today DataWindow painter function in the header band.

FOR INFO For information, see "Adding computed fields to a DataWindow object" on page 525.

About the detail band

The detail band displays the retrieved data. It is also where the user enters new data and updates existing data. The number of rows of data that display in the DataWindow object at one time is determined by the following expression:

$$\frac{(\text{Height of the DataWindow object} - \text{Height of headers and footers})}{\text{Height of the detail band}}$$

When PowerBuilder generates the basic DataWindow object, the presentation style determines the contents of the detail band:

- ◆ If the presentation style is Tabular, Grid, N-Up, or Label, the detail band displays column names, representing the columns
- ◆ If the presentation style is Freeform, the labels defined for the columns in the Database painter display in the detail band with boxes for the data to the right

How PowerBuilder names the columns in the workspace

If the DataWindow object uses one table, the names of the columns in the workspace 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 workspace are *tablename_columnname*. PowerBuilder prefaces the name of the column with the table name to prevent ambiguity, since different tables can have columns with the same name.

When you design the detail band of a DataWindow object, you can specify display and validation information for each column of the DataWindow object and add other objects, such as text, pictures, drawing objects, and graphs.

About the summary and footer bands

You use the summary and footer bands of the DataWindow 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

These bands can contain any information you want, including text, drawing objects, graphs, and computed fields containing aggregate totals.

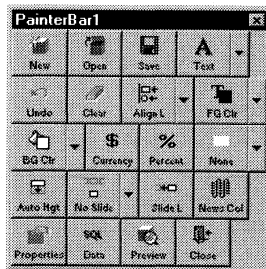
Using the DataWindow painter toolbars

The DataWindow painter contains a customizable PainterBar and a StyleBar.

FOR INFO For more information about using toolbars, see "Using toolbars" on page 18.

PainterBar

The PainterBar includes buttons for standard operations (such as Open, Save, Undo) and for common reporting operations (such as Preview, Percent). It also includes six dropdown toolbars, which are indicated by a small black triangle on the right part of a button:



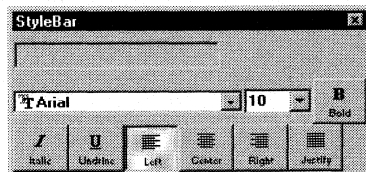
These are the dropdown toolbars that are available:

Dropdown Toolbar	Used to
Background Color	Specify the background color of one or more selected objects
Borders	Specify borders for one or more selected objects

Dropdown Toolbar	Used to
Foreground Color	Specify the foreground color of one or more selected objects. In a text object, the foreground color specifies the color of the text
Layout	Specify the alignment, sizing, and spacing of selected objects
Objects	Specify objects to add to a DataWindow object
Slide	Specify sliding for objects

StyleBar

The StyleBar includes buttons for applying properties (such as font size and bold) to selected text elements:



Using property sheets in the DataWindow painter

Each part of the DataWindow object (such as text, columns, computed fields, bands, graphs, even the DataWindow object itself) has a property sheet. The items in the property sheet are appropriate to the part.

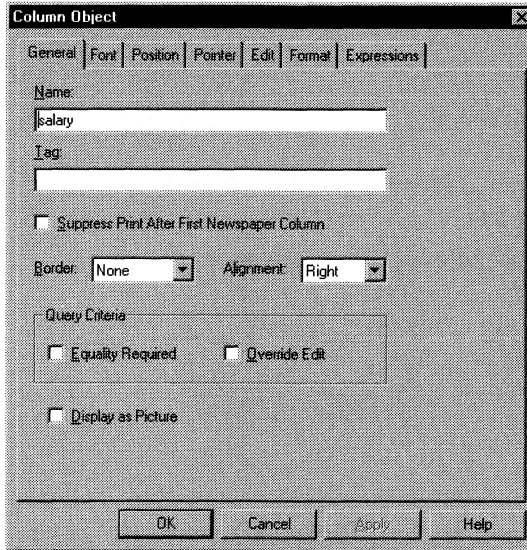
You can use property sheets to modify the parts of the DataWindow object.

❖ To use a property sheet in the DataWindow painter workspace:

- 1 Position the mouse over the part you want to modify.
- 2 Display the part's popup menu and select Properties.

The appropriate property sheet displays.

The following property sheet is for a column. The Column property sheet for a report has seven tabbed property pages of information, which you access by clicking the appropriate tab. For example, if you want to choose an edit style for the column, you click the Edit tab. This brings the Edit property page to the front of the property sheet:



When you want to modify the properties of part of a DataWindow object, display the property sheet and pick the properties you want to change. Click the various tabs to change property pages.

Selecting objects in the DataWindow painter

The DataWindow painter provides several ways to select objects to act on. You can select multiple objects and act on all the selected objects as a unit. For example, you can move all of them or change the fonts used to display text for all of them.

Lasso selection

Use **lasso selection** when possible because it's fast and easy. Lasso selection is another name for the method described below for selecting neighboring multiple objects.

❖ **To select one object in a DataWindow object:**

- ◆ Click it.

The object displays with handles on it. Previously selected objects are no longer selected.

❖ **To select neighboring multiple objects in a DataWindow object (lasso selection):**

- 1 Press and hold the left mouse button (mouse button on the Macintosh) at one corner of the neighboring objects.
- 2 Drag the mouse over the objects you want to select.
PowerBuilder displays a bounding box (the lasso).
- 3 Release the mouse button.
All the objects in the bounding box are selected.

❖ **To select non-neighboring multiple objects in a DataWindow object:**

- 1 Click the first object.
- 2 Press and hold the CTRL key and click additional objects.

On Macintosh

On the Macintosh, press the COMMAND key instead of the CTRL key.

All the objects are selected.

❖ **To select objects by type or position in the DataWindow object:**

- ◆ Do the following:

To select	Select this item from the menu bar	Or press this shortcut key
All objects	Edit>Select>Select All	CTRL+A
All text	Edit>Select>Select Text	
All columns	Edit>Select>Select Columns	
Objects in relation to the currently selected object	Edit>Select>Select Above	CTRL+UP ARROW
	Edit>Select>Select Below	CTRL+DOWN ARROW
	Edit>Select>Select Left	CTRL+LEFT ARROW
	Edit>Select>Select Right	CTRL+RIGHT ARROW

On Macintosh

On the Macintosh, press the COMMAND key instead of the CTRL key.

Displaying information about the selected object

The name, x and y coordinates, width, and height of the selected object are displayed in the MicroHelp bar. If multiple objects are selected, *Group Selected* displays in the Name area and the coordinates and size do not display.

Using keyboard shortcuts in the DataWindow painter

The following table lists the keyboard shortcuts available in the DataWindow painter.

On Macintosh

On the Macintosh, press the COMMAND key instead of the CTRL key and the OPTION key instead of the ALT key.

Action	Key combination	Comments
Boldface text	CTRL+B	Toggles boldface on and off for all selected objects
Center text	CTRL+SHIFT+C	
Clear	DEL	Clears all selected objects
Close painter	CTRL+F4 CTRL+W	
Close PowerBuilder	ALT+F4 CTRL+Q	Available everywhere on Windows and UNIX. Not available on Macintosh
Copy	CTRL+C	Copies objects
Cut	CTRL+X	Cuts objects
Edit text	CTRL+SHIFT+E	Activates the Text box in the StyleBar
File editor	SHIFT+F6	Available everywhere
Italicize text	CTRL+I	Toggles italic on and off for all selected objects
Left justify text	CTRL+SHIFT+L	
Move object	ARROW	

Action	Key combination	Comments
New DataWindow object	CTRL+N	Creates a new DataWindow object
Open DataWindow object	CTRL+O	Opens an existing DataWindow object
Paste	CTRL+V	Paste objects
PowerPanel	CTRL+SHIFT+N	Available everywhere
Preview	CTRL+SHIFT+P	Toggles between preview and the design workspace
Print	CTRL+P	
Resize object	SHIFT+ARROW	
Return to the object	CTRL+SHIFT+O	Returns focus to the object from the StyleBar
Right justify text	CTRL+SHIFT+R	
Save DataWindow object	CTRL+S	
Select above	CTRL+UP ARROW	Selects all objects above the currently selected object
Select all	CTRL+A	Selects all objects in the report
Select below	CTRL+DOWN ARROW	Selects all objects below the currently selected object
Select left	CTRL+LEFT ARROW	Selects all objects that are left of the currently selected object
Select right	CTRL+RIGHT ARROW	Selects all objects that are right of the currently selected object
Switch to	ALT+ESC	Available everywhere
Tab	TAB	Tabs to next object
Tab backwards	SHIFT+TAB	Tabs to previous object
Underline text	CTRL+SHIFT+U	Toggles underline on and off for all selected objects (underline is not supported on UNIX)
Undo/Redo	CTRL+Z	Undoes the most recent change (including the most recent undo)

Resizing bands in the DataWindow painter workspace

You can change the size of any band in the DataWindow object.

- ❖ **To resize a band in the DataWindow painter workspace:**
 - ◆ 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 so you can get a better idea of how your DataWindow object looks. For example, if you are working with a large DataWindow object, you can zoom out so you can see all of it on your screen. Or you can zoom in on a group of objects to better see their details.

- ❖ **To zoom the display in the DataWindow painter:**
 - 1 Select Design>Options from the menu bar.
 - 2 Select the Zoom property page.
 - 3 Select a built-in zoom percentage.
or
Set a custom zoom percentage by typing an integer in the Custom box.

Undoing changes in the DataWindow painter

You can undo your most recent change in the workspace by pressing CTRL+Z or selecting Edit>Undo from the menu bar. (If you have just undone an action, the menu item changes to Edit>Redo.)

On Macintosh

On the Macintosh, press the COMMAND key instead of the CTRL key.

Previewing a DataWindow object

You can preview a DataWindow object to view it as it will appear and test the processing that takes place in it. This section provides information on what you can do while previewing a DataWindow object, including:

- ◆ "Retrieving data" on page 480
- ◆ "Modifying data" on page 483
- ◆ "Sorting and filtering data" on page 484
- ◆ "Viewing row information" on page 485
- ◆ "Importing data into a DataWindow object" on page 485
- ◆ "Using print preview" on page 486
- ◆ "Printing data" on page 488
- ◆ "Saving data in an external file" on page 488
- ◆ "Saving the data in HTML Table format" on page 489
- ◆ "Working with PSR files" on page 491
- ◆ "Mailing reports" on page 493
- ◆ "Working in a grid DataWindow object" on page 494

Changes you make to the DataWindow object while previewing are maintained when you return to the workspace. For example, if you specify sorting while previewing, the sorting becomes part of the design of the DataWindow object.

❖ To preview the DataWindow object:

- 1 Click the Preview button in the PainterBar.
or
Select Design>Preview from the menu bar.

You are now in preview. The bars that indicate the bands disappear, and by default PowerBuilder retrieves all the rows from the database (you are prompted to supply arguments if you defined retrieval arguments).

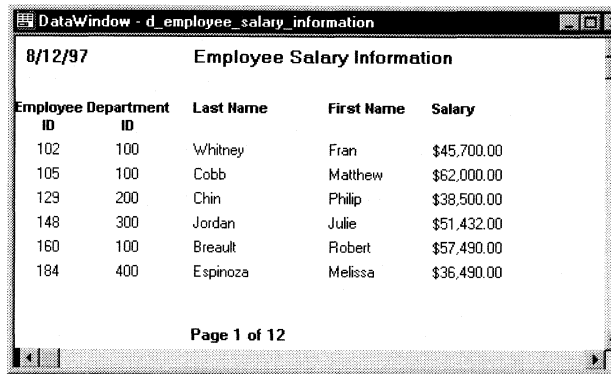
In external DataWindow objects

If the DataWindow 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 485.

In DataWindow objects that have stored data

If the DataWindow object has stored data in it, no data is retrieved from the database.

As the rows are being retrieved, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval:



The screenshot shows a window titled "DataWindow - d_employee_salary_information" with a date of "8/12/97". The window displays a table of "Employee Salary Information". The table has five columns: "Employee ID", "Department ID", "Last Name", "First Name", and "Salary". The data rows are as follows:

Employee ID	Department ID	Last Name	First Name	Salary
102	100	Whitney	Fran	\$45,700.00
105	100	Cobb	Matthew	\$62,000.00
129	200	Chin	Philip	\$38,500.00
148	300	Jordan	Julie	\$51,432.00
160	100	Breault	Robert	\$57,490.00
184	400	Espinoza	Melissa	\$36,490.00

At the bottom of the window, it says "Page 1 of 12".

- 2 Test your DataWindow object.

For example, modify some data, update the database, re-retrieve rows, and so on, as described below.

- 3 To leave preview, click the Preview button.

or

Select Design>Preview from the menu bar.

You return to the workspace.

Retrieving data

Where PowerBuilder gets data

When you preview a DataWindow object, PowerBuilder follows this order of precedence to supply the data in your DataWindow object:

- 1 If you have saved data in the DataWindow object (using Rows>Data from the menu bar), PowerBuilder uses the saved rows from the DataWindow object and does not retrieve data from the database.
- 2 If data caching is on, PowerBuilder uses the data in the cache (if there is any). Data caching is on by default.

- 3 If there is no data in the cache yet or if data caching is not on, PowerBuilder retrieves data from the database automatically, with one exception. If the Retrieve on Preview option is off, you have to request retrieval explicitly, as described next.

Previewing without
retrieving data

If you do not want PowerBuilder to retrieve data from the database automatically when you first preview, you can set the Retrieve on Preview option. Then you can preview the DataWindow object without retrieving data.

❖ **To be able to preview without retrieving data automatically:**

- 1 With the DataWindow object displayed in the workspace, select Design>Options from the menu bar.

The DataWindow Options property sheet displays.

- 2 Select the General tab.
- 3 Deselect the Retrieve on Preview checkbox.

When this checkbox is unchecked, your request to preview the DataWindow object will not result in automatic data retrieval from the database.

PowerBuilder uses
data caching

By default, when PowerBuilder first retrieves data during preview, it stores the data internally. If you go back to the workspace and then return to preview, PowerBuilder displays the stored data instead of retrieving rows from the database again. This can save you a lot of time, since data retrieval can be time consuming.

On Windows 3.1, SHARE must be loaded

In order for PowerBuilder on Windows 3.1 to cache the data, you must be running the DOS SHARE program. If you use Windows for Workgroups, Windows NT, or Windows 95, SHARE is built in; do not run the DOS SHARE program.

On Macintosh

On Macintosh, PowerBuilder does not cache data when you repeatedly preview a DataWindow object. Caching data for display during execution using the Rows>Data command is a separate function and works the same way on Macintosh as on Windows.

How using data from the cache affects you

Because PowerBuilder accesses the cache and does not automatically retrieve data every time you preview, you may not have what you want when you preview. The data you see in preview and the data in the database can be out of sync.

For example, if you are working with live data that changes frequently or with statistics based on changing data and you spend time designing the DataWindow object, the data you are looking at may no longer match the database. In this case, you may want to retrieve again just before printing.

Turning off data caching

If you do not want PowerBuilder to use data from the cache, you can turn off the caching of retrieved data.

❖ **To turn off data caching so that PowerBuilder will retrieve fresh data from the database every time you preview a DataWindow object:**

1 With the DataWindow object displayed in the workspace, select Design>Options from the menu bar.

The DataWindow Options property sheet displays.

2 Select the General tab.

3 Deselect the Retain Data to Design checkbox.

Explicitly retrieving data

You can explicitly request retrieval anytime.

❖ **To retrieve rows from the database:**

◆ Click the Retrieve button in the PainterBar.

or

Select Rows>Retrieve from the menu bar.

Supplying argument values or criteria

If the DataWindow object has retrieval arguments or is set up to prompt for criteria, you will be prompted to supply values for the arguments or to specify criteria.

PowerBuilder retrieves the rows. As PowerBuilder retrieves, the Retrieve button changes to a Cancel button. You can click the Cancel button to stop the retrieval anytime.

Other options that affect retrieval

These other options can affect retrieval:

- ◆ **Retrieve Rows As Needed** Lets you specify that only the rows needed to display the current portion of the DataWindow object should be retrieved. When you scroll downward, additional rows are retrieved. This can speed up reporting in certain situations.

FOR INFO See "Retrieving rows as needed" on page 541.
- ◆ **Retrieve Rows to Disk** (Windows 95, Windows NT, and UNIX only) Lets you specify that PowerBuilder should save retrieved data on your hard disk in a temporary file rather than keep the data in memory. When you preview your DataWindow object, PowerBuilder swaps rows of data from the temporary file into memory as needed.

FOR INFO For information, see "Saving retrieved rows to disk" on page 542.

Modifying data

You can add, modify, or delete rows while previewing. When you have finished manipulating the data, you can apply the changes to the database.

If looking at data from a view or from more than one table

By default, you cannot update data in a DataWindow object that contains a view or more than one table.

FOR INFO For more about updating DataWindow objects, see "Controlling updates" on page 543.

❖ To modify existing data:

- ◆ Tab to the field and enter a new value.

Preview uses validation rules, display formats, and edit styles that you have defined for the columns, either in the Database painter or in this particular DataWindow object.

To save the changes to the database, you must apply them, as described below.

❖ **To add a row:**

- 1 Click the Insert Row button.
PowerBuilder creates a blank row.
- 2 Enter data for a row.

To save the changes to the database, you must apply them, as described below.

Adding a row in an application

Clicking the Insert Row button while previewing is equivalent to calling the InsertRow function and then the ScrollToRow function during execution.

❖ **To delete a row:**

- ◆ Click the Delete Row button.
PowerBuilder removes the row from the display.

To save the changes to the database, you must apply them, as described below.

Deleting a row in an application

Clicking the Delete Row button while previewing is equivalent to calling the DeleteRow function during execution.

❖ **To apply changes to the database:**

- ◆ Click the Update Database button.
PowerBuilder updates the table with all the changes you have made.
-

Applying changes in an application

Clicking the Update Database button while previewing is equivalent to calling the Update function during execution.

Sorting and filtering data

You can define and use sort criteria and filters for the rows in preview. The sort criteria and filters you define while previewing are maintained when you return to the workspace.

FOR INFO See Chapter 17, "Filtering, Sorting, and Grouping Rows".

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 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 into a DataWindow object

While previewing, you can import and display data from an external source. Then you can save the imported data in the database.

❖ **To import data into a DataWindow object while previewing:**

- 1 Select Rows>Import from the menu bar.
- 2 Specify the file from which you want to import the data.

The types of files that you can import into the painter display in the List Files of Type dropdown listbox.

- 3 Click Open.

PowerBuilder reads the data from the file into the painter. You can then click the Update Database button in the PainterBar to add the new rows to the database.

Data from file must match retrieved columns

When importing data from a file, the data must match all the columns in the retrieved data (typically, the columns specified in the SELECT statement), not just the columns that are displayed in the DataWindow object.

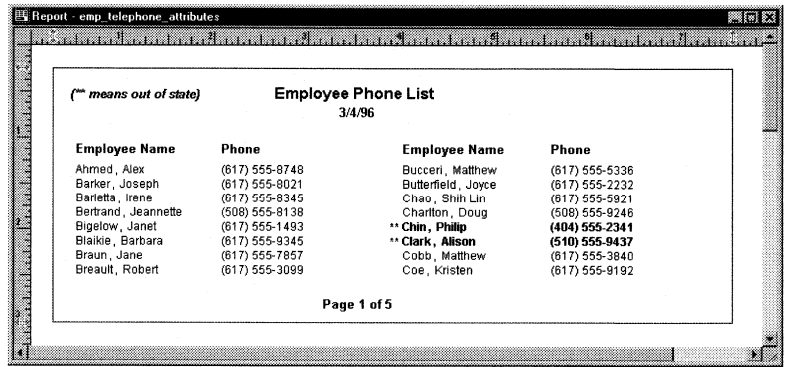
Using print preview

You can print the data displayed while previewing. Before printing, you can preview the output on the screen.

❖ **To preview printed output before printing:**

- ◆ Select File>Print Preview from the menu bar from within preview.

Print Preview displays the DataWindow object as it will print. Rulers display around the page borders.



Using the IntelliMouse pointing device

Using the IntelliMouse pointing device, users can scroll a DataWindow object (at execution time or in preview) by rotating the wheel. Users can also zoom a DataWindow object larger or smaller by holding down the CTRL key while rotating the wheel.

Controlling the display of rulers

You can choose whether to display rulers around page borders.

❖ **To control the display of rulers:**

- ◆ Select/deselect File>Print Preview Rulers from the menu bar.

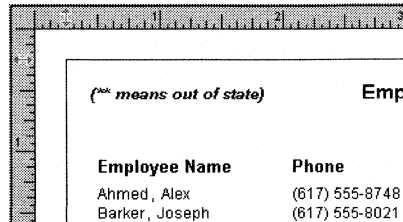
Changing margins

You can dynamically change margins while previewing a DataWindow object.

❖ **To change the margins while previewing:**

- ◆ Drag the margin boundaries on the rulers.

The following picture shows the left and top margin boundaries. There are also boundaries for the right and bottom margins:



Changing margins during execution

Using the Modify function, you can display a DataWindow object in print preview during execution. While in print preview your users can also change margins by dragging boundaries. A user event in the DataWindow control (pbm_dwnprintmarginchange) is triggered when print margins are changed. Changing margins can affect the page count. So if you use the Describe function to display the page count in your application (for example, in MicroHelp), you should code a script for the user event to recalculate the page count.

Zooming the page

You can reduce or enlarge the amount of the page that displays on the screen. This does not affect the printed output.

❖ To zoom the page on the display screen:

- 1 Select File>Print Preview Zoom from the menu bar.
- 2 Select the magnification you want and click OK.

The display of the page zooms in or out as appropriate. The size of the contents of the page changes proportionately as you zoom. This type of zooming affects your display but does not affect printing.

Zooming the contents

In addition to zooming the display on the screen, you can also zoom the contents, affecting the amount of material that prints on a page.

❖ To zoom the contents of a DataWindow object with respect to the printed page:

- 1 Select Design>Zoom from the menu bar.
- 2 Select the magnification you want and click OK.

The contents of the page zooms in or out as appropriate. If you enlarge the contents so they no longer fit, PowerBuilder creates additional pages as needed.

Printing data

You can print your DataWindow object while previewing. You can print all pages, a range of pages, only the current page, or only odd or even pages. You can also specify whether you want multiple copies, collated copies, and printing to a file.

To change printers or settings before printing

You can choose File>Printer Setup from the menu bar in preview. On the Macintosh, you must set the printer specifications before printing.

❖ To print:

- 1 Select File>Print from the menu bar to display the Print dialog box.
- 2 Specify the number of copies to print.
- 3 Specify the pages: select All or Current Page, or type page numbers and/or page ranges in the Pages box.
- 4 Specify all pages, even pages, or odd pages in the Print dropdown listbox.
- 5 If you want to print to a file rather than to the printer, select the Print to File checkbox.
- 6 If you want to change the collating option, deselect or select the Collate Copies checkbox and click OK.
If you specified print to file, the Print to File dialog box displays.
- 7 Enter a filename and click OK.
The extension PRN identifies it as a file prepared for the printer. Change drive and/or directory if you want.

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 in the file format named Powersoft Report.

FOR INFO For more information about the Powersoft Report file format, see "Working with PSR files" on page 491.

❖ 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 dropdown listbox.

If you want the column headers saved in the file, select a file format that includes headers (such as Excel With Headers). When you select a *with headers* format, the names of the database columns (not the column labels) will also be saved in the file.

When you choose a format, PowerBuilder supplies the appropriate file extension on platforms that require them.

Saving the data as a Powersoft report or HTML Table

You can choose Powersoft Report in the Save As dialog box to save a Powersoft PSR file. You can choose HTML Table to save the data in HTML Table format.

- 3 Name the file and click Save.

PowerBuilder saves all displayed rows in the file; all columns in the displayed rows are saved. Filtered rows are not saved.

Saving the data in HTML Table format

One of the formats you can choose to save data in is HTML Table format. When you save in HTML Table format, PowerBuilder saves a style sheet along with the data. If you use this format, you can open the saved file in a browser such as Netscape. Once you have the file in HTML Table format, you can continue to enhance the file in HTML.

About the results

Some presentation styles translate better into HTML than others. The Tabular, Group, Freeform, Crosstab, and Grid presentation styles produce good results. The Composite, RichText, Graph, and OLE 2.0 presentation styles and nested reports produce HTML tables based on the result set (data) only and not on the presentation style. DataWindow objects with overlapping objects in them may not produce the results you want.

An example

The example shows a report in preview and the file saved in HTML Table format in Netscape.

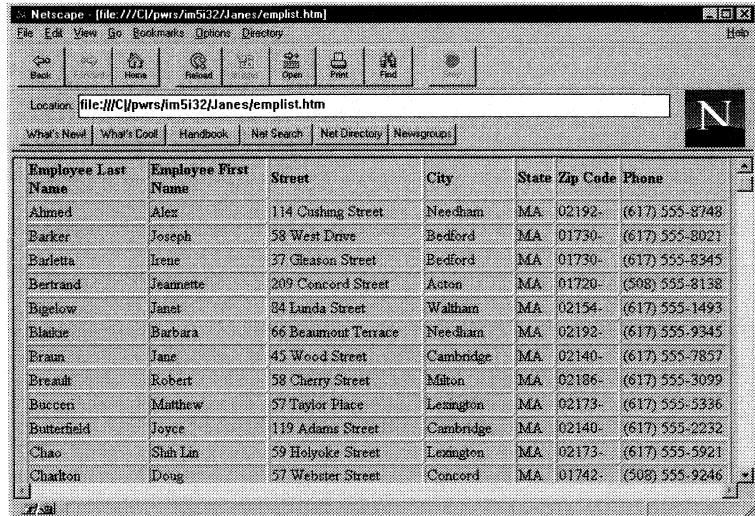
These are the steps you would follow to recreate this example:

- 1 Open a DataWindow or report.
- 2 Preview it:

Employee Last Name	Employee First Name	Street	City	State	Zip Code	Phone
Ahmed	Alex	114 Cushing Street	Needham	MA	02192-	(617) 555-8748
Barker	Joseph	58 West Drive	Bedford	MA	01730-	(617) 555-8021
Barletta	Irene	37 Gleason Street	Bedford	MA	01730-	(617) 555-8345
Bertrand	Jeanette	209 Concord Street	Acton	MA	01720-	(508) 555-8138
Bigelow	Janet	84 Lunda Street	Waltham	MA	02154-	(617) 555-1493
Blaikie	Barbara	66 Beaumont Terrace	Needham	MA	02192-	(617) 555-9345
Braun	Jane	45 Wood Street	Cambridge	MA	02140-	(617) 555-7857
Breault	Robert	58 Cherry Street	Milton	MA	02106-	(617) 555-3099
Bucceri	Matthew	57 Taylor Place	Lexington	MA	02173-	(617) 555-5336
Butterfield	Joyce	119 Adams Street	Cambridge	MA	02140-	(617) 555-2232
Chao	Shih Lin	59 Holyoke Street	Lexington	MA	02173-	(617) 555-5921
Charlton	Doug	57 Webster Street	Concord	MA	01742-	(508) 555-9246

- 3 Select File>Save Rows As from the menu bar.
- 4 Choose the HTML Table format for the file from the Save As Type dropdown listbox.
- 5 Name the file.
PowerBuilder creates a file using the name you supplied and the extension *htm*.
- 6 Open a browser such as Netscape or Internet Explorer.
- 7 Use the browser's file open command to open the HTML file.

This screen shows the file opened in Netscape:



Working with PSR files

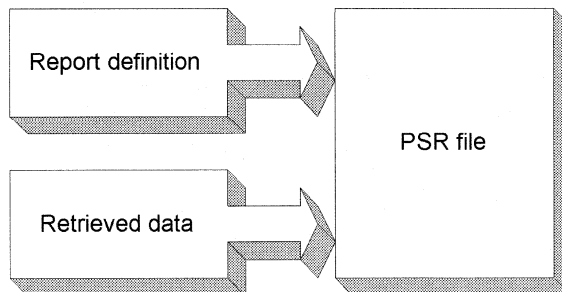
A PSR file is a special file with the extension PSR created by PowerBuilder, InfoMaker, or DataWindow Builder. PSR stands for Powersoft report.

A PSR file contains a report definition (source and object) as well as the data contained in the report when the PSR file was created.

About reports

A report is the same as a nonupdatable DataWindow object.

FOR INFO For more information, see "Reports versus DataWindow objects" on page 415.



You can use a PSR file to save a complete report (report design and data). This can be especially important if you need to keep a snapshot of data taken against a database that changes frequently.

How PSR files are created

PowerBuilder creates a PSR file when you:

- ◆ Mail a report to an InfoMaker user using electronic mail.
FOR INFO See "Mailing reports" on page 493.
- ◆ Save data in the Powersoft Report file format.
FOR INFO See "Saving data in an external file" on page 488.

Opening a PSR file

PSR files are used primarily by InfoMaker, Powersoft's reporting tool. When an InfoMaker user opens a PSR file, InfoMaker displays the report in the Report painter. If InfoMaker is not already running, opening a PSR file automatically starts InfoMaker.

On UNIX InfoMaker is not currently available on UNIX.

On Macintosh On the Macintosh, you cannot open PSR files created in versions earlier than PowerBuilder 6.0.

InfoMaker users can open a PSR file in File Manager or Explorer, in a mail message, and using the File menu in the Report painter. PowerBuilder and DataWindow Builder users can open a PSR file in the Report painter.

Windows and PSR files

When PowerBuilder is installed, the Powersoft report file type is registered with Windows.

- ❖ **To open a PSR file in InfoMaker using Explorer or File Manager or from a mail message:**
 - ◆ Double-click the PSR filename.
InfoMaker displays the report in preview.
- ❖ **To open a PSR file from the menu bar in the Report painter in PowerBuilder, DataWindow Builder, or InfoMaker:**
 - 1 Select File>Open File from the menu bar.
The Select a File Name dialog box displays.

- 2 Select the PSR file you want. Change drives and directories if needed.
The Report painter displays the report in the workspace.
- 3 Click the Preview button to preview the report.

PSR files and retrieval

When you are previewing a PSR file, you see the data that was saved in the file when it was created. This is true until you explicitly retrieve data again using the Retrieve button or Rows>Retrieve from the menu bar.

If you attempt to retrieve data with a PSR file, you must be sure that you are properly connected to the right database. Otherwise, you will receive a database error message.

If you retrieve new data while previewing a PSR file, you cannot go back to the old data contained in the file. To go back to the old data, you must leave the PSR file without saving and then reopen the PSR file.

Mailing reports

On Windows systems, while previewing in the Report painter (but not in the DataWindow painter), you can mail a report as a PSR file to an InfoMaker user who is using a MAPI-compliant mail system such as Microsoft Exchange. (MAPI stands for *messaging application program interface* and is one of the programming interfaces to mail systems.)

FOR INFO For more about PSR files, see "Working with PSR files" on page 491.

Using 16-bit mail clients

Use of the 16-bit mail API (MAPI) is not supported under 32-bit PowerBuilder—you need to use the 32-bit version.

❖ To mail a report:

- 1 While previewing a report in the Report painter, select File>Send from the menu bar.

If you are not logged on to your mail system, you will be prompted for your password.

- 2 Enter your password and click OK.

A form for mailing the report displays. PowerBuilder creates and attaches the appropriate PSR file (which holds the report and data).

- 3 Complete the form and send the message.

PowerBuilder mails the PSR file.

The recipient can open the report by double-clicking it if InfoMaker is installed.

Working in a grid DataWindow object

If you are previewing a grid-style DataWindow object, you can do the following:

- ◆ Resize columns
- ◆ Reorder columns
- ◆ Split the display into two horizontal scrolling regions—you can use this feature to keep one or more columns stationary on the screen while scrolling through other columns.
- ◆ Copy data to the clipboard

These features are also available to your users

Users of your application can also manipulate columns these ways in a grid DataWindow object during execution.

❖ 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 (mouse button on the Macintosh) 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 (mouse button on the Macintosh) on a column heading.

PowerBuilder selects the column and displays a line representing the column border:

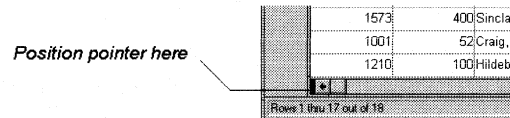
The screenshot shows a grid header with five columns: Employee ID, Department, Name, Salary, and Status. A vertical dashed line is positioned at the right boundary of the 'Department' column, indicating it is selected. Below the header is a horizontal scrollbar.

Employee ID	Department	Name	Salary	Status
-------------	------------	------	--------	--------

- 2 Drag the mouse left or right to move the column.
- 3 Release the mouse button.

❖ **To use split horizontal scrolling in a grid DataWindow object:**

- 1 To divide the grid into two regions that can scroll independently of each other, position the mouse pointer at the left end of the horizontal scrollbar.



The pointer changes to a two-headed arrow.

- 2 Press and hold the left mouse button (mouse button on the Macintosh) and drag the mouse to the right to create a new horizontal scrolling border.
- 3 Release the mouse button.

You now have two independently scrolling regions in the grid DataWindow object.

❖ **To copy data to the clipboard 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 (COMMAND on the Macintosh), then click the headers.
 - ◆ To select cells, press the left mouse button (mouse button on the Macintosh) on the bottom border of a cell and drag the mouse.

Selected cells are highlighted.

- 2 Select Edit>Copy from the menu bar.

or

Press CTRL+C (COMMAND + C on the Macintosh).

The contents of the selected cells are copied to the clipboard. If you copied the contents of more than one column, the data is separated by tabs.

Modifying general DataWindow object properties

This section describes the general DataWindow object properties that you can modify in the workspace. It covers:

- ◆ "Changing the DataWindow object style" next
- ◆ "Setting colors in a DataWindow object" on page 497
- ◆ "Specifying properties of a grid DataWindow object" on page 498
- ◆ "Specifying pointers for a DataWindow object" on page 499
- ◆ "Defining print specifications for a DataWindow object" on page 499
- ◆ "Modifying text in a DataWindow object" on page 503
- ◆ "Defining the tab order in a DataWindow object" on page 504
- ◆ "Naming objects in a DataWindow object" on page 506
- ◆ "Using borders in a DataWindow object" on page 506
- ◆ "Specifying variable-height detail bands in a DataWindow object" on page 507
- ◆ "Modifying the data source of a DataWindow object" on page 509

Changing the DataWindow object style

The general style properties for a DataWindow object include:

- ◆ The unit of measure used in the DataWindow object
- ◆ A timer interval for events in the DataWindow object
- ◆ A background color for the DataWindow object

PowerBuilder assigns defaults when it generates the basic DataWindow object. You can change the defaults.

❖ To change the default style properties:

- 1 Position the pointer in the background of the DataWindow object, display the popup menu, and select Properties.

The DataWindow Object property sheet displays with the General property page on top.

- 2 Click the unit of measure you want to use to specify distances when working with the DataWindow object:
 - ◆ PowerBuilder units (PBUs)
 - ◆ Pixels (smallest element on the display monitor)
 - ◆ Thousandths of an inch
 - ◆ Thousandths of a centimeter

Choosing the unit of measure

If you plan to print the contents of the DataWindow object during execution, change the unit of measure to inches or centimeters to make it easier to specify the margin measurements.

You may also want to use inches or centimeters if you plan to move your PBL to another platform (such as from Windows to the Macintosh) because the results will be more consistent.

- 3 Specify the number of milliseconds you want between internal timer events in the DataWindow object.

This value determines how often PowerBuilder updates the time fields in the DataWindow object. The default is 60,000 milliseconds (one minute).
- 4 Select a background color from the Color dropdown listbox. The default color is the window background color.
- 5 Click OK.

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 colors in a DataWindow object:

- ◆ Do one of the following:

To set colors for	Do this
The DataWindow object's background	Position the mouse on an empty spot in the DataWindow object, display the popup menu, then select Properties. On the General property page of the DataWindow object's property sheet, select a color from the Color dropdown listbox

To set colors for	Do this
A band	Position the mouse pointer on the bar that represents the band, display the popup menu, then select Properties. On the General property page of the band's property sheet, select a color from the Color dropdown listbox. The choice you make here overrides the background color for the DataWindow object
An object	Position the mouse pointer on the object, display the popup menu, then select Properties. For objects that use text, you can set colors for background and text on the Font property page of the property sheet. For drawing objects, you can set colors on the General property page of the property sheet

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 popup menu, and select Properties.
- 2 Select the options you want in the Grid box on the General property page:

Option	Result
On	Grid lines always display
Off	Grid lines never display (users cannot resize columns during execution)
Display Only	Grid lines display only when the DataWindow object displays online
Print Only	Grid lines display only when the contents of the DataWindow object are printed
Column Moving	Columns can be moved during execution
Mouse Selection	Data can be selected during execution (and, for example, copied to the clipboard)
Row Resize	Rows can be resized during execution

Specifying pointers for a DataWindow object

Just as with colors, you can specify different pointers to use when the mouse is over a particular area of the DataWindow object. For example, you might want to change the pointer when the mouse is over a column whose data cannot be changed.

❖ **To change the mouse pointer used during execution:**

- 1 Position the mouse over the element of the DataWindow object whose pointer you want to define, display the popup menu, and select Properties to display the appropriate property sheet.

You can set a pointer for the entire DataWindow object, specific bands, and specific objects.

- 2 Select the Pointer tab.
- 3 Choose the pointer either from the Stock Pointers list or, if you have a file containing pointer definitions (CUR files), enter a pointer filename.

You can use the Browse button to search for the file.

Pointers for reports running on both Windows and Macintosh

If you design pointers for DataWindow objects to be used on both platforms, use only the middle of the image. The outside will not be visible on the Macintosh. On Windows, pointers and icons are 32x32 pixels. On the Macintosh, they are 16x16 pixels.

- 4 Click OK.

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.

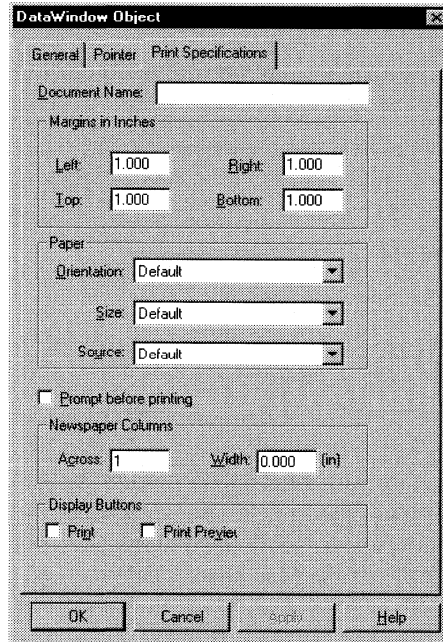
❖ **To define print specifications for a DataWindow object:**

- 1 In the DataWindow painter, select Properties from the DataWindow object's popup menu to display the DataWindow object's property sheet.
- 2 In the Units box on the General property 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 property page uses the units of measure you specified on the General property page. For example, if the unit of measure is inches, the Print Specifications property page looks like this:



- 4 Specify a name in the Document Name box.

This is the name that will be used in the print queue to identify the report.

- 5 Specify the margins for the report.

You can also change margins in print preview while you are actually looking at data. If you change margins while previewing, the changes are recorded here on the Print Specifications page.

FOR INFO See "Using print preview" on page 486.

- 6 Select the paper's orientation, size, and source from the dropdown listboxes.

For orientation, choose from the following:

Setting	Result
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

- 7 If you want to prompt for print setup before printing during execution, select the Prompt Before Printing checkbox.

PowerBuilder will display the standard Print Setup dialog box each time users make a print request.

- 8 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.

FOR INFO See "Printing with newspaper-style columns" next.

- 9 Click OK.

Printing with newspaper-style columns

When you define a DataWindow object, you can specify that it print in multiple columns across the page, like a newspaper. A typical use of newspaper-style columns is a phone list, where you want to have more than one column of names on a printed page. The following DataWindow object (shown in print preview) has two newspaper-style columns:

Employee Phone List			
42397			
Employee Name	Phone	Employee Name	Phone
Whitney, Fran	(617) 555-3985	Chao, Shih Lin	(617) 555-5921
Cobb, Matthew	(617) 555-3840	Blakie, Barbara	(617) 555-9345
Chan, Philip	(404) 555-2341	Smith, Susan	(713) 555-6613
Jordan, Julie	(617) 555-7835	Preston, Mark	(617) 555-5862
Breaux, Robert	(617) 555-3099	Clark, Alison	(510) 555-9437
Espinoza, Melissa	(508) 555-2319	Soo, Hing	(617) 555-8748
Bertrand, Jeannette	(508) 555-8138	Goggin, Kevin	(617) 555-3785
Dill, Marc	(617) 555-2144	Bucceri, Matthew	(617) 555-5336

❖ To define newspaper-style columns for a DataWindow object:

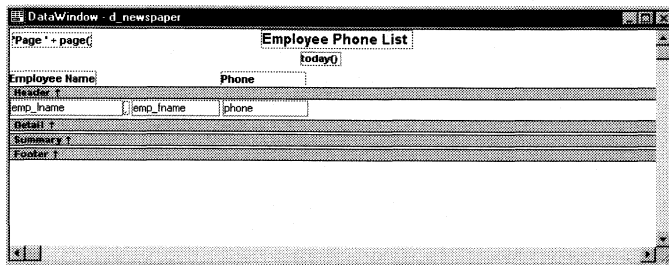
- 1 Build a tabular DataWindow object with the data you want.
- 2 Select Properties from the DataWindow object's popup menu.
- 3 Select the Print Specifications tab.

- 4 Specify the number of columns across the page and the width of columns in the Newspaper Columns box.
- 5 Click OK.
- 6 For each object in the DataWindow object that you do *not* want to appear multiple times on the page (such as headers), select Properties from the object's popup menu. Then select the Suppress Print After First Newspaper Column checkbox.

Example

This example describes how you would create a newspaper-style DataWindow object.

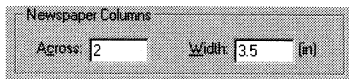
First create a tabular DataWindow object with the last name, first name, and phone number columns. Then add a title, page number, and date. The DataWindow object looks like this in the workspace:



Sliding columns

The Emp_Fname column and the text object holding a comma are defined as Slide Left so they display just to the right of the Emp_Lname column.

Next you specify two columns across and a column width of 3.5 inches in the Newspaper Columns box in the Print Specifications page of the DataWindow object's property sheet:



Now you're ready to preview the DataWindow object.

Use Print Preview to see the printed output

Newspaper-style columns are used only when the DataWindow object is printed. They do not appear when a DataWindow object executes (or in Preview). Therefore, to see them in PowerBuilder, use Print Preview in the DataWindow painter.

When you preview the DataWindow object, PowerBuilder displays the result set in two columns. Everything above the column headers (which includes page number, title, and date) also shows twice because of the 2-column specification. This information should appear only once per page.

To specify that page number, title, and date should appear only once on the page, you need to suppress printing after the first column. For each of these objects, select Properties from the object's popup menu. Then select the Suppress Print After First Newspaper Column checkbox:

Suppress Print After First Newspaper Column

The finished DataWindow object looks like this, with one set of page heading information and two columns of column header and detail information:

Employee Name		Phone	
Whitney, Fran	(617) 555-3985	Chao, Shih Lin	(617) 555-5921
Cobb, Matthew	(617) 555-3840	Blakie, Barbara	(617) 555-9345
Chin, Philip	(404) 555-2341	Smith, Susan	(713) 555-6613
Jordan, Julie	(617) 555-7835	Preston, Mark	(617) 555-5862
Breault, Robert	(617) 555-3099	Clark, Alison	(510) 555-9437
Espinosa, Melissa	(508) 555-2319	Soo, Hing	(617) 555-8748
Bertrand, Jeannette	(508) 555-8138	Goggin, Kevin	(617) 555-3785
Dill, Marc	(617) 555-2144	Bucceri, Matthew	(617) 555-5336

Modifying text in a DataWindow object

When PowerBuilder initially generates the basic DataWindow object, it uses the following:

- ◆ For the text and alignment of column headings and labels, PowerBuilder uses the extended column attributes made in the Database painter.
- ◆ For fonts, PowerBuilder uses the definitions made in the Database painter for the table. If you didn't specify fonts for the table, PowerBuilder uses the defaults set in the Application painter.

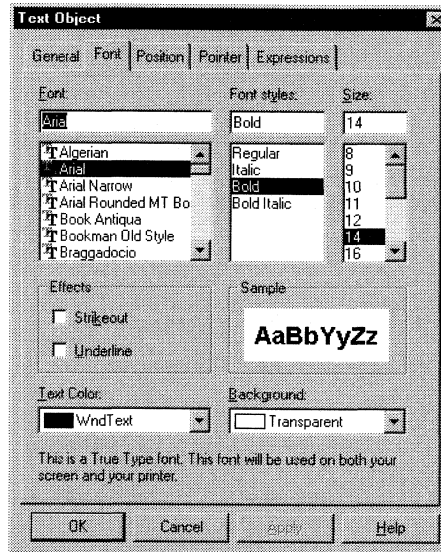
You can override any of these defaults in a particular DataWindow object.

- ❖ **To change text in a DataWindow object:**
 - 1 Select the text.

The first box in the StyleBar is now active.
 - 2 Type the new text.

Use ~n~r to embed a newline character in the text.

- ❖ **To change the text properties for a text object in a DataWindow object:**
 - 1 Select the text object.
 - 2 Do one of the following:
 - ◆ Change the text properties in the StyleBar.
 - ◆ Select the Font property page in the object's property sheet and change the properties there.



Defining the tab order in a DataWindow object

When PowerBuilder generates the basic DataWindow object, it assigns columns a default **tab order**, the default sequence in which focus moves from column to column when a user presses the TAB key during execution. PowerBuilder assigns tab values in increments of 10 in left-to-right and top-to-bottom order.

Tab order is not used in the workspace

Tab order is used when a DataWindow object is executed, but it is not used in the DataWindow painter workspace. In the workspace, the TAB key moves to the objects in the DataWindow object in the order in which the objects were placed in the workspace.

If the DataWindow object contains columns from more than one table

If you are defining a DataWindow object with more than one table, PowerBuilder assigns each column a tab value of 0, meaning the user cannot tab to the column. This is because, by default, multitable DataWindow objects are not updatable—users cannot modify data in them. You can change the tab values to nonzero values to allow tabbing in these DataWindow objects.

FOR INFO For more about controlling updates in a DataWindow object, see "Controlling updates" on page 543.

❖ To change the tab order:

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

The current tab order displays.

- 2 Use the mouse or the TAB key to move the pointer to the tab value you want to change.
- 3 Enter a new tab value (0-9999).

0 removes the column from the tab order (the user cannot tab to the column). It doesn't matter exactly what value you use (other than 0); all that matters is relative value. For example, if you want the user to tab to column B after column A but before column C, set the tab value for column B so it is between the value for column A and the value for column C.

- 4 Repeat the procedure until you have the tab order you want.
- 5 Select Design>Tab Order from the menu bar again.

PowerBuilder saves the tab order.

Each time you select Tab Order, PowerBuilder reassigns tab values to include any columns that have been added to the 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.

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

Naming objects in a DataWindow object

You use names to identify columns and other objects in validation rules, filters, PowerScript functions, and DataWindow painter functions. PowerBuilder names objects it places in the generated DataWindow object (that is, columns, labels, and headings), but you need to name objects you place yourself if you want to refer to them anywhere.

❖ To specify a name for an object in a DataWindow object:

- 1 Select Properties from the object's popup menu and then select the General tab.
- 2 Type the name in the Name box.

Using borders in a DataWindow object

You can place borders around text, columns, graphs, and crosstabs to enhance their appearance. PowerBuilder provides six types of borders: Underline, Box, Resize, Shadow box, 3D Raised, and 3D Lowered:

Amo & Sons

If you specify the Resize border, users can resize the object during execution. Resize borders are particularly useful with graphs.

Border appearance varies

Changing the border style may not have the same effect on all platforms. For example, on Windows 95, the 3D Lowered style looks the same as the Box style for DropDownListBoxes.

❖ To add a border to an object in a DataWindow object:

- 1 Select one or more objects.
- 2 Select the border you want from the Border dropdown toolbar in the PainterBar.

PowerBuilder places the border around the selected objects.

You can also specify a border for an object in the object's property sheet on the General property page. But when you use the property sheet, you can deal with only one object at a time. When you use the Border dropdown toolbar, you can change borders for many objects at once.

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. You don't want to reserve space for that much information for the column in the detail band, since it would make the detail band's height very large, meaning the users could see few rows at a time.

Instead, you want the detail band to resize based on the data in the Memo column. If the Memo column has only one line of text, you want the detail band to be one line. If the Memo column has 20 lines of text, you want the detail band to be 20 lines high.

To provide a detail band that resizes as needed, you specify that the variable-length columns and the band have Autosize Height.

❖ To create a resizable detail band in a DataWindow object:

- 1 Select Properties from the popup menu of a column that should resize based on the amount of data.
- 2 Select the Autosize Height checkbox on the Position property page and click Apply.
- 3 Clear the Auto Horz Scroll checkbox on the Edit property page and click OK.

PowerBuilder will wrap text during preview 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 popup menu.
- 6 Select the Autosize Height checkbox and click OK.

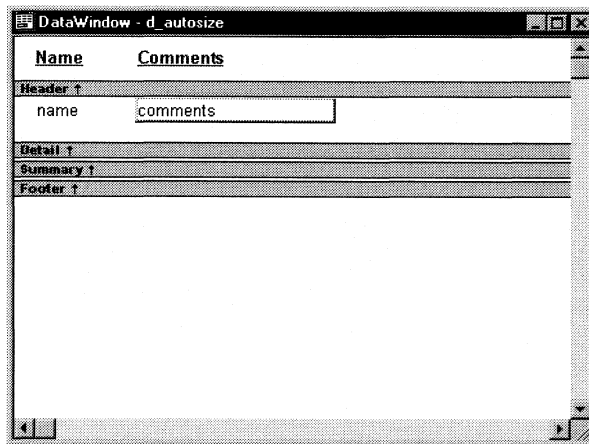
During preview, the detail band will resize based on the contents of the columns you defined as having Autosize Height.

Clipping columns

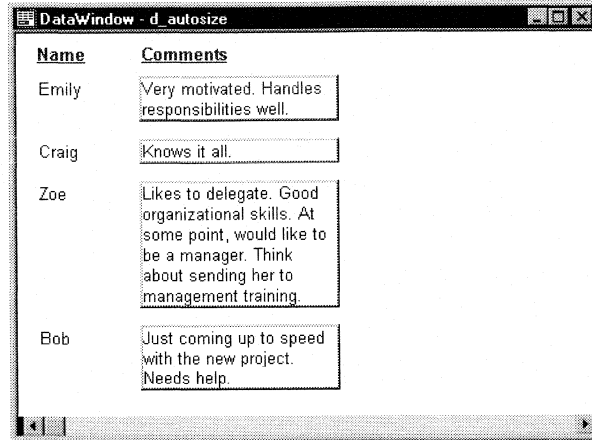
You can have Autosize Height columns without an Autosize Height detail band. If such a column expands beyond the size of the detail band during preview, it will be clipped.

Example

In the following DataWindow object, the Comments column is defined as having Autosize Height, as is the detail band. Horizontal scrolling is turned off for the Comments column:



During execution, the space allocated for a row depends on how much information is in Comments; the detail band expands to fit the text. In the DataWindow object below, the first row takes two lines, the second row takes one line, and so on:



Modifying the data source of a DataWindow object

When modifying a DataWindow object, you might realize that you haven't included all the columns you need. Or you might need to define retrieval arguments. You can modify the data source from the DataWindow painter workspace. How you do it depends on the data source.

Modifying SQL SELECT statements

If the data source is SQL (such as Quick Select, SQL Select, or Query), you can graphically modify the SQL SELECT statement.

❖ To modify a SQL data source:

- 1 Select Design>Data Source from the menu bar.

PowerBuilder returns you to the Select painter. (If you used Quick Select to define the data source, this might be the first time you have seen the Select painter.)

- 2 Modify the SELECT statement graphically using the same techniques as when creating it.

FOR INFO For more information, see "Using SQL Select" on page 441.

Modifying the statement syntactically

Select Design>Convert to Syntax from the menu bar to modify the SELECT statement syntactically.

- 3 Select Design>Data Source from the menu bar to return to the painter workspace.

Some changes you make (such as adding or removing columns) require PowerBuilder to modify the update capabilities of the DataWindow object.

FOR INFO For more information about controlling updates in a DataWindow object, see "Controlling updates" on page 543.

Changing the table

If you change the table referenced in the SELECT statement, PowerBuilder maintains the columns in the workspace (now from a different table) only if they match the data types and order of the columns in the original table.

Modifying the retrieval arguments

You can add, modify, or delete retrieval arguments when modifying your data source.

❖ To modify the retrieval arguments:

- 1 In the Select painter, select 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 INFO For more information about retrieval arguments, see Chapter 14, "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 Select Design>Data Source from the menu bar.

The Modify Result Set Description dialog box displays.

- 2 Review the description and make any necessary changes.
- 3 Click OK.

If the data source is a stored procedure

If you are modifying the result set for a DataWindow object whose data source is a stored procedure, the Modify Result Set Description dialog box contains a More button.

Click More to edit the Execute statement, select another stored procedure, or add retrieval arguments:

Name	Type	Length	Dec
id	long		
company_name	string	35	

Edit Execute: Result Set: 1
 execute dba.sp_customer_list;0
 Page Arguments:

Reorganizing objects in a DataWindow object

This section describes the activities that help you change the layout and appearance of the objects in a DataWindow object, including:

- ◆ "Displaying boundaries for objects in a DataWindow object" next
- ◆ "Using the grid and the ruler in a DataWindow object" on page 513
- ◆ "Deleting objects in a DataWindow object" on page 513
- ◆ "Moving objects in a DataWindow object" on page 514
- ◆ "Copying objects in a DataWindow object" on page 514
- ◆ "Resizing objects in a DataWindow object" on page 515
- ◆ "Aligning objects in a DataWindow object" on page 516
- ◆ "Equalizing the space between objects in a DataWindow object" on page 516
- ◆ "Equalizing the size of objects in a DataWindow object" on page 517
- ◆ "Sliding objects to remove blank space in a DataWindow object" on page 517

Displaying boundaries for objects in a DataWindow object

When reorganizing objects in the workspace, it is sometimes helpful to see how large all the objects are. That way you can easily check for overlapping objects and make sure that the spacing around objects is what you want.

❖ To display object boundaries in a DataWindow object:

- 1 Select Design>Options from the menu bar.
The DataWindow Options property sheet displays.
- 2 Select the Show Edges checkbox on the General property page.
PowerBuilder displays the boundaries of each object in the DataWindow object.

Boundaries display only in the painter workspace

The boundaries displayed for objects are for use only in the painter workspace. 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 objects.

❖ To use the grid and the ruler:

- 1 Select Design>Options from the menu bar.

The DataWindow Options property sheet displays. The Alignment Grid box on the General property page contains the alignment grid options.

- 2 Use the options as needed:

Option	Meaning
Snap to Grid	Make objects 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 FOR INFO See "Changing the DataWindow object style" on page 496

Your choices for the grid and the ruler are saved and used the next time you start PowerBuilder.

Deleting objects in a DataWindow object

❖ To delete objects in a DataWindow object:

- 1 Select the objects you want to delete.

- 2 Click the Clear button.

or

Select Edit>Clear from the menu bar.

or

Press the DEL key.

Moving objects in a DataWindow object

In all presentation styles except Grid

In all presentation styles except Grid, you can move all the objects (such as headings, labels, columns, drawing objects, and graphs) anywhere you want.

❖ **To move objects in a DataWindow object:**

- 1 Select the objects you want to move.
- 2 Do one of the following:
 - ◆ Drag the objects with the mouse.
 - ◆ Press an arrow key to move the objects in one direction.

In grid DataWindow objects

You can reorder columns in a grid DataWindow object during execution.

FOR INFO See "Working in a grid DataWindow object" on page 494.

Copying objects in a DataWindow object

You can copy objects within a DataWindow object and to other DataWindow objects. All properties of the objects are copied.

❖ **To copy an object in a DataWindow object:**

- 1 Select the object in the DataWindow painter workspace.
- 2 Select Edit>Copy from the menu bar.
or
Press CTRL+C (COMMAND+C on the Macintosh).
The object is copied to a private PowerBuilder clipboard.
- 3 Copy (paste) the object to the same DataWindow object or to another one:
 - ◆ To copy the object within the same DataWindow object, select Edit>Paste from the menu bar or press CTRL+V (COMMAND +V on the Macintosh).

- ◆ To copy the object to another DataWindow object, open another instance of the DataWindow painter and open the desired DataWindow object in it. Make that DataWindow object active and paste the object.

PowerBuilder pastes the object at the same location as in the source DataWindow object (if you are pasting into the same DataWindow object, you should move the pasted object so it doesn't cover the original object). PowerBuilder displays a message box if the object you are pasting is not valid in the destination DataWindow object.

Resizing objects in a DataWindow object

You can resize an object using the mouse or the keyboard.

Using the mouse

To resize an object using the mouse, select it, then grab an edge and drag it with the mouse.

Using the keyboard

To resize an object using the keyboard, select it and then do the following:

To make the object	Press
Wider	SHIFT+RIGHT ARROW
Narrower	SHIFT+LEFT ARROW
Taller	SHIFT+DOWN ARROW
Shorter	SHIFT+UP ARROW

In grid DataWindow objects

You can resize columns in grid DataWindow objects.

❖ To resize a column in a grid DataWindow object:

- 1 Position the mouse pointer at a column boundary.
The pointer changes to a 2-headed arrow.
- 2 Press and hold the left mouse button (mouse button on the Macintosh) and drag the mouse to move the boundary.
- 3 Release the mouse button when the column is the correct width.

Aligning objects in a DataWindow object

Often you want to align several objects or make them all the same size. You can use the grid to align the objects or you can have PowerBuilder align them for you.

❖ To align objects in a DataWindow object:

- 1 Select the object whose position you want to use to align the others.
PowerBuilder displays handles around the selected object.
- 2 Extend the selection by pressing and holding the CTRL key (COMMAND on the Macintosh) and clicking the objects you want to align with the first one.
All the objects have handles on them.
- 3 Select Edit>Align Objects from the menu bar.
- 4 From the cascading menu, select the dimension along which you want to align the controls.

For example, to align the objects along the left side, select the first choice on the cascading menu.

PowerBuilder moves all the selected objects to align with the first one.

Equalizing the space between objects in a DataWindow object

If you have a series of objects and the spacing is fine between two of them but the spacing is wrong for the rest, you can easily equalize the spacing around all the objects.

❖ To equalize the space between objects in a DataWindow object:

- 1 Select the two objects whose spacing is correct.
To do so, click one object, then press CTRL (COMMAND on the Macintosh) and click the second object.
- 2 Select the other objects whose spacing you want to be the same as the first two objects by pressing CTRL (COMMAND on the Macintosh) and clicking.
- 3 Select Edit>Space Objects from the menu bar.
- 4 From the cascading menu, select the dimension whose spacing you want to equalize.

Equalizing the size of objects in a DataWindow object

Say you have several objects in a DataWindow object and want their sizes to be the same. You can accomplish this manually or by using the Edit menu.

❖ To equalize the size of objects in a DataWindow object:

- 1 Select the object whose size is correct.
- 2 Select the other objects whose size you want to match the first object by pressing CTRL (COMMAND on the Macintosh) and clicking.
- 3 Select Edit>Size Objects from the menu bar.
- 4 From the cascading menu, select the dimension whose size you want to equalize.

Sliding objects 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 objects in the DataWindow object to the left or up if there is blank space. You can use this feature to remove blank lines in mailing labels or to remove extra spaces between fields (such as first name and last name).

Slide is used by default in nested reports

PowerBuilder uses slide options automatically when you nest a report to ensure that the reports are positioned properly.

❖ To use sliding columns or objects in a DataWindow object:

- 1 Select Properties from the object's popup menu and then select the Position tab.
- 2 Select the Slide options you want:

Option	Description
Left	Slide the column or object to the left if there is nothing to the left. Be sure the object does not overlap the object to the left. Sliding left will not work if the objects overlap
Up - All Above	Slide the column or object up if there is nothing in the row above (the row above must be completely empty for the column or object to slide up)

Option	Description
Up - Directly Above	Slide the column or object up if there is nothing <i>directly above it</i> in the row above

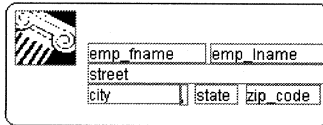
If you are sliding columns up

Even blank columns have height; if you want columns to slide up, you need to specify as Autosize Height all columns above that might be blank and that you want to slide other columns up through.

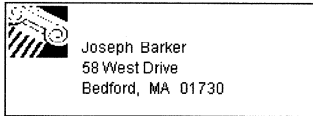
Example

In a mailing label that includes first and last names, as well as address information, you can use sliding to combine the columns appropriately.

In the following label, emp_fname, the comma, state, and zip_code are specified as slide left. Edges are shown to indicate the spacing between the columns. Notice that there is a small amount of space between objects. This space is necessary for Slide Left to work properly:



When you preview (run) the report, the last name, comma, state, and zip code slide left to remove the blank space:



Prompting for retrieval criteria in a DataWindow object

You can define your DataWindow object so that it always prompts for retrieval criteria just before it retrieves data. PowerBuilder allows you to prompt for criteria when retrieving data for a DataWindow control, but not for a DataStore object.

❖ To prompt for retrieval criteria in a DataWindow object:

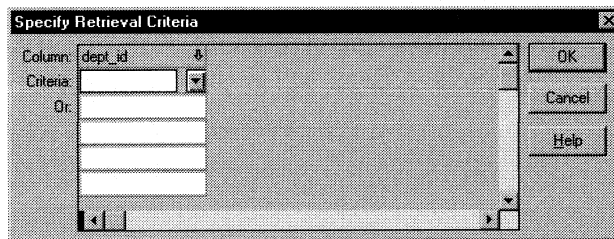
- 1 Select Rows>Prompt for Criteria from the menu bar.

The Prompt For Criteria dialog box displays listing all columns in the DataWindow object.

- 2 Select the columns you want the user to be able to specify retrieval criteria for during execution and click OK.

What happens

When you specify prompting for criteria, PowerBuilder displays the following 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 Prompt for Criteria dialog box displays in the grid. Users can specify criteria here exactly as in the grid with the Quick Select data source.

Criteria specified here are added to the WHERE clause for the SQL SELECT statement defined for the DataWindow object.

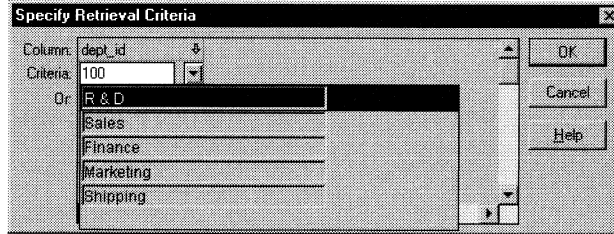
Testing in PowerBuilder

You can test the prompting for criteria by previewing the DataWindow object. Note that by default, after doing the first retrieval, PowerBuilder stores the data internally (caches it) and doesn't re-retrieve it the next time you go to preview. So you won't be prompted for criteria each time you go to preview. You can explicitly retrieve by clicking the Retrieve button.

FOR INFO For more about data caching, see "Retrieving data" on page 480.

Using edit styles

If a column uses a code table or the RadioButton, CheckBox, or DropDownListBox edit style, an arrow displays in the column header and users can select a value from a dropdown listbox when specifying criteria:



If you don't want the dropdown listbox used for a column for specifying retrieval criteria, select the Override Edit checkbox on the General property page of the column's property sheet.

Forcing the entry of criteria

If you have specified prompting for criteria for a column, you can force the entry of criteria for the column by selecting the Equality Required checkbox on the General property page of the column's property sheet. PowerBuilder will underline 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 431 describes in detail how you and your users can specify selection criteria in the grid.

The chapter on dynamic DataWindow objects in *Application Techniques* describes how to write scripts to dynamically allow users to specify retrieval criteria during execution.

Adding objects to a DataWindow object

This section describes adding objects to enhance your DataWindow object:

- ◆ "Adding columns to a DataWindow object" next
- ◆ "Adding text to a DataWindow object" on page 522
- ◆ "Adding drawing objects to a DataWindow object" on page 523
- ◆ "Adding a groupbox to a DataWindow object" on page 523
- ◆ "Adding pictures to a DataWindow object" on page 524
- ◆ "Adding computed fields to a DataWindow object" on page 525
- ◆ "Adding buttons to a DataWindow object" on page 532
- ◆ "Adding graphs to a DataWindow object" on page 536
- ◆ "Adding OLE objects to a DataWindow object" on page 536
- ◆ "Adding reports to a DataWindow object" on page 537

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.

FOR INFO See "Modifying the data source of a DataWindow object" on page 509.

❖ **To add a column from the data source to a DataWindow object:**

- 1 Click the Column button.
or
Select Objects>Column from the menu bar.
- 2 Click where you want to place the column.
The Select Column dialog box displays listing all columns included in the data source of the DataWindow object.
- 3 Select the column and click OK.

Adding text to a DataWindow object

When PowerBuilder generates a basic DataWindow object from a presentation style and data source, it places columns and their headings in the workspace. You can add text anywhere you want to make the DataWindow object easier to understand.

❖ **To add text to a DataWindow object:**

- 1 Click the Text button in the Objects dropdown toolbar.
or
Select Objects>Text from the menu bar.
- 2 Click where you want the text.
PowerBuilder places the text object in the workspace and displays the word *text*.
- 3 Type the text in the textbox at the top of the workspace.
- 4 (Optional) Change the font, size, style, and alignment for the text using the StyleBar.

About the default font and style

When you place text in a DataWindow object, PowerBuilder uses the font and style (such as boldface) defined for the application's text in the Application painter. You can override the text properties for any text in a DataWindow object.

FOR INFO For more about changing the application's default text font and style, see Chapter 2, "Working with Applications".

Adding drawing objects to a DataWindow object

You can add the following drawing objects to a DataWindow object to enhance its appearance:

- Rectangle
- RoundRectangle
- Line
- Oval

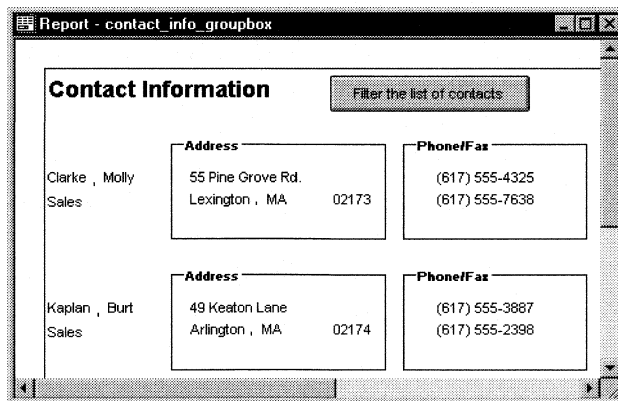
❖ To place a drawing object in a DataWindow object:

- 1 Select the drawing object from the Objects dropdown toolbar or from the Objects menu.
- 2 Click where you want the object to display.
- 3 Resize or move the drawing object as needed.
- 4 Use the drawing object's property sheet 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 groupbox to a DataWindow object

To visually enhance the layout of a DataWindow object, you can add a groupbox. The groupbox is a static frame used to group and label a set of objects in a DataWindow object. The following example shows two groupboxes in a report (nonupdatable DataWindow). The Address groupbox groups address information and the Phone/Fax groupbox groups telephone numbers:



❖ **To add a groupbox to a DataWindow object:**

- 1 Select Objects>Group Box from the menu bar and click in the workspace.
- 2 With the groupbox selected, type the text to display in the frame.
- 3 Move and resize the groupbox 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.

Tips for using pictures

To display a different picture for each row of data, retrieve a column containing picture filenames from the database.

FOR INFO For more information, see "Modifying column properties in the Database painter" on page 356.

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.

FOR INFO For information about these functions, see the *PowerScript Reference*.

❖ **To place a picture in a DataWindow object:**

- 1 Click the Picture button in the Objects dropdown toolbar.
or
Select Objects>Picture from the menu bar.
- 2 Click where you want the picture to display.
The Picture Object property sheet displays.

- 3 Use the Browse button to find the file or enter a filename in the File Name box.

On Windows, the picture must be a bitmap (BMP), runlength-encoded (RLE), or Windows metafile (WMF) file.

On Macintosh systems, the picture must be a bitmap (BMP), runlength-encoded (RLE), or PICT file.

- 4 Name the picture by typing a name in the Name box.
- 5 Click the Original Size checkbox to display the bitmap in its original size.

You can use the mouse to change the size of the bitmap in the DataWindow painter.

- 6 Click the Invert Image checkbox to display the picture with its colors inverted.
- 7 Click the OK button.

You return to the workspace with the picture in place.

Adding computed fields to a DataWindow object

You can use **computed fields** in any band of the DataWindow object. Typical uses include:

- ◆ **Calculations based on column data that change for each retrieved row** For example, if you are retrieving yearly salary, you can define a computed field in the detail band that displays monthly salary (defined as `Salary / 12`).
- ◆ **Summary statistics of the data** For example, if you have a grouped DataWindow object, you can use a computed field to calculate the totals of a column for each group.
- ◆ **Concatenated fields** For example, if you are retrieving first name and last name, you can define a computed field that concatenates the values so they appear with only one space between them (defined as `Fname + " " + Lname`).
- ◆ **System information** For example, you can place the current date and time in a DataWindow object's header by using computed fields (defined as `Today()` and `Now()`).

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 workspace, you can define **computed fields** after you have defined the SELECT statement (or other data source)

The difference between the two ways

When you define the computed column in the Select painter, the value is calculated by the DBMS when the data is retrieved. The computed column's value does not change until data has been updated and retrieved again.

When you define the computed field in the DataWindow painter workspace, the value of the column is calculated in the DataWindow object after the data has been retrieved. The value changes dynamically as the data in the DataWindow object changes.

Example

Consider these columns in a DataWindow object. Cost is computed as Quantity * Price:

Part #	Quantity	Price	Cost
101	100	1.25	125.00

If Cost is defined as a computed column in the Select painter, the SELECT statement is as follows:

```
SELECT part.part_num,  
       part.part_qty,  
       part.part_price,  
       part.part_qty * part.part_price  
FROM part;
```

If the user changes the price of part 101 in the DataWindow object in this scenario, the cost does not change in the DataWindow object until the database is updated and the data is retrieved again. So the user gets this display, with the incorrect cost:

Part #	Quantity	Price	Cost
101	100	2.50	125.00

On the other hand, if Cost is defined as a computed field in the DataWindow object workspace, the SELECT statement looks like the following statement. It does not have a computed column:

```
SELECT part.part_num,
       part.part_qty,
       part.part_price
FROM part;
```

The computed field is defined in the workspace as $\text{Quantity} * \text{Price}$.

In this scenario, if the user changes the price of part 101 in the DataWindow object, the cost changes immediately:

Part #	Quantity	Price	Cost
101	100	2.50	250.00

Recommendation

If you want your DBMS to do the calculations on the server before bringing data down and you don't care about dynamically updating the computed values, define the computed column as part of the SELECT statement.

If you want computed values to change dynamically, define your computed fields in the DataWindow painter workspace, as described next.

Defining a computed field in the DataWindow painter workspace

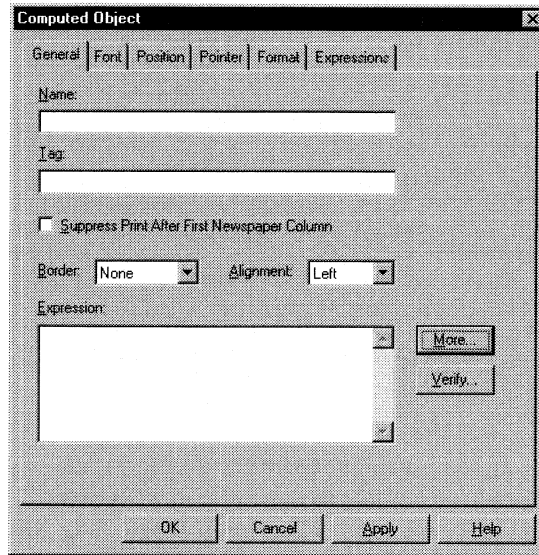
❖ To define a computed field in the DataWindow painter workspace:

- 1 Click the Compute button in the Objects dropdown toolbar.
or
Select Objects>Computed Field from the menu bar.

- 2 Click where you want the computed field.

If the calculation is to be based on column data that changes for each row, make sure you place the computed field in the detail band.

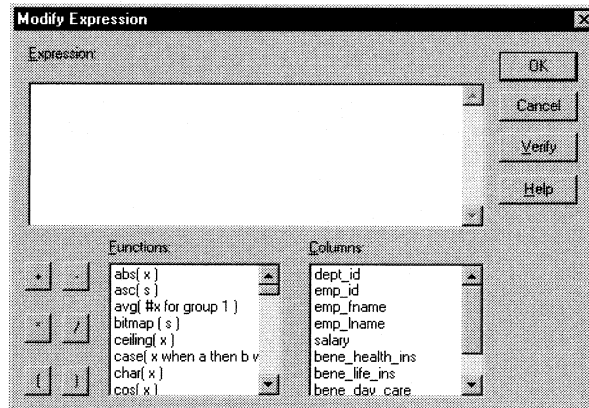
The Computed Object property sheet displays:



- 3 Name the computed field.
- 4 Click the More button.

The Modify Expression dialog box displays listing:

- ◆ DataWindow painter functions you can use in the computed field
- ◆ The columns in the DataWindow object
- ◆ Operators and parentheses



- 5 Enter the expression that defines the computed field (see below).
- 6 (Optional) Click Verify to test the expression.
PowerBuilder analyzes the expression.
- 7 Click OK.
PowerBuilder returns you to the computed column's property sheet.
- 8 Click OK.
You return to the workspace with the computed field in place.

Entering the expression

You can enter any valid DataWindow painter expression when defining a computed field. You can paste operators, columns, and DataWindow painter functions into the expression from information in the Modify Expression dialog box.

DataWindow painter expressions

The expression you are entering is a DataWindow painter expression; it is not a SQL expression processed by the DBMS. So the expression follows the rules for DataWindow painter expressions.

FOR INFO For complete information about DataWindow painter expressions, see the *DataWindow Reference*.

You can use any non-object-level function (built-in or user-defined) in an expression.

You can use the + operator to concatenate strings.

Referring to next and previous rows

You can refer to other rows in a computed field. This is particularly useful in n-up DataWindow objects when you want to refer to another row in the detail band. Use this syntax:

ColumnName[x]

where *x* is an integer. 0 refers to the current row (or first row in the detail band), 1 refers to the next row, -1 refers to the previous row, and so on.

Examples

Here are some examples of computed fields and columns:

To display	Enter this expression	In this band
Current date at top of each page	Today() (a built-in DataWindow painter function)	Header
Current time at top of each page	Now()	Header

To display	Enter this expression	In this band
Current page at bottom of each page	Page()	Footer
Total page count at bottom of each page	PageCount()	Footer
Concatenation of Fname and Lname columns for each row	Fname + " " + Lname	Detail
Monthly salary if Salary column contains annual salary	Salary / 12	Detail
Four asterisks if the value of the Salary column is greater than \$50,000	IF(Salary > 50000, "****", "")	Detail
Average salary of all retrieved rows	Avg(Salary)	Summary
Count of retrieved rows, assuming each row contains a value for EmpID	Count(EmpID)	Summary

For more information

For complete information about the functions you can use in computed fields in the DataWindow painter, see the *DataWindow Reference*.

A shortcut for doing summary statistics

PowerBuilder provides a quick way to create computed fields that summarize values in the detail band.

❖ **To summarize values:**

- 1 Select one or more columns in the DataWindow object's detail band.
- 2 Do one of the following:

To place this computed field	Do this
Average	Select Objects>Average - Computed Field from the menu bar
Count	Select Objects>Count - Computed Field from the menu bar
Sum	Select Objects>Sum - Computed Field from the menu bar or add the built-in Sum button to the PainterBar and use it

PowerBuilder places a computed field in the summary band or in the group trailer band if the DataWindow object is grouped. The band is resized automatically to hold the computed field. If there is already a computed field that matches the one being generated, it is skipped.

Adding custom buttons that place computed fields You can add buttons to the PainterBar in the DataWindow painter that place computed fields using any of the aggregate functions, such as Max, Min, and Median.

❖ **To customize the PainterBar with custom buttons for placing computed fields:**

- 1 Place the mouse pointer over the PainterBar and select Customize from the popup menu.

The Customize dialog box displays.

- 2 Click Custom in the Select palette group to display the set of custom buttons.

- 3 Drag a custom button into the Current toolbar group and release it.

The Toolbar Item Command dialog box displays.

- 4 Click the Function button.

The Function For Toolbar dialog box displays.

- 5 Select a function and click OK.

You return to the Toolbar Item Command dialog box.

- 6 Specify text that displays for the button and click OK.

PowerBuilder places the new button in the PainterBar. You can click it to add a computed field to your DataWindow object the same way you use the built-in Sum button.

A shortcut for placing page numbers and date

You can click buttons in the PainterBar to place a computed field for the current page number and date anywhere in the DataWindow object:

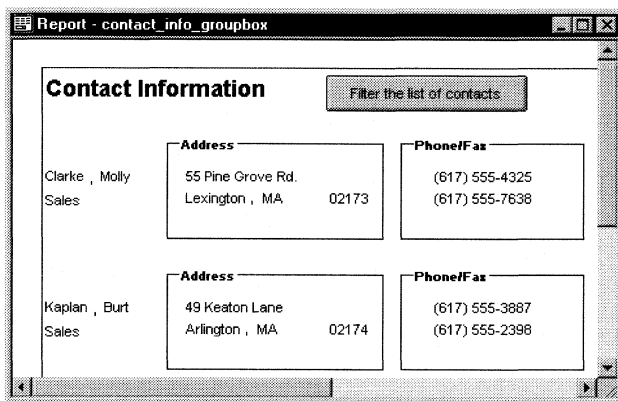
To place this computed field	Do this
'Page ' + page() + ' of ' + pageCount()	Click the Page button or select Objects>Page n of n Computed Field from the menu bar
today()	Click the Today button or select Objects>Today() - Computed Field from the menu bar

Adding buttons to a DataWindow object

Buttons make it easy to provide command button actions in a DataWindow object. No coding is required. Furthermore, the use of buttons (Button objects) in the DataWindow object (rather than CommandButton controls in a window) ensures that the actions appropriate to the DataWindow object are included in the object.

The Button object is a command (or Picture) style button that can be placed in a DataWindow object. When clicked at execution time, the button activates either a built-in or user-supplied action.

The following example shows a button placed in a report (a nonupdatable DataWindow object). Clicking the button brings up the Filter dialog, where users can specify a filter to be applied to the currently retrieved data:



❖ **To add a button to a DataWindow object:**

- 1 Select Objects>Button from the menu bar.
- 2 Click where you want the button to display.

Be careful when putting buttons in the detail band

Buttons in the detail band repeat for every row of data (often not desirable). During retrieval the buttons in the detail band are not visible. So, for example, a Cancel button in the detail band would be unavailable when needed.

You may find it useful to put a Delete button or an Insert button in the detail band. Clicking a Delete button in the detail band will delete the row next to the button clicked. Clicking an Insert button in the detail band will insert a row following the row containing the button clicked.

- 3 With the button still selected, type the text to display on the button.
- 4 Display the button's property sheet (General page).
- 5 Select the action you want to assign to the button from the Action dropdown listbox.

FOR INFO For information about actions, see "Actions assignable to buttons in DataWindow objects" on page 534.
- 6 If you want to add a picture to the button, select the Use Action Default Picture checkbox or enter the name of the image file to display on the button.
- 7 If you want to suppress event processing when the button is clicked at execution time, select the Suppress Event Processing checkbox.

When this option has been selected for the button and the button is clicked at execution time, only the action assigned to the button is executed. The ButtonClicking and the ButtonClicked events are not triggered.
- 8 Click OK.

What happens if Suppress Event Processing is off

If Suppress Event Processing is off and the button is clicked, the ButtonClicking event is fired. Code in the ButtonClicking event (if any) is executed. If the return code is 0, the action assigned to the button is then executed. After the action is executed (or if the return code from the ButtonClicking event is 1), the ButtonClicked event is fired.

Note that the Clicked event is fired before the ButtonClicking event.

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 property sheet for the DataWindow object (not the property sheet 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 property sheet with the Print page on top.
- 2 Select the Print checkbox to have buttons included in the printed output when the DataWindow object is printed.
- 3 Select the Print Preview checkbox to have the buttons display on the screen when viewing the DataWindow object in print preview.

Actions assignable to buttons in DataWindow objects

These are the actions you can assign to a button in a DataWindow object:

Action	What it does	Value	Meaning of action return code from ButtonClicked event
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 actually occurs, 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

Action	What it does	Value	Meaning of action return code from ButtonClicked event
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 and sorts as specified	8	1 if successful -1 if an error occurs
Filter	Displays Filter dialog and filters as specified	9	Number of rows filtered Number < 0 if an error occurs
Delete Row	If button is in detail band, deletes row associated with button; otherwise, deletes the current row	10	1 if successful -1 if an error occurs
Append Row	Inserts row at the end	11	Row number of newly inserted row
Insert Row	If button is in detail band, inserts row using row number associated with the button; otherwise, inserts row using the current row	12	Row number of newly inserted row
Update	Saves changes to the database. If the update is successful, a Commit will be issued; if the update fails, a Rollback will be issued	13	1 if successful -1 if an error occurs
Save Rows As	Displays Save As dialog and saves rows in the format specified	14	Number of rows filtered Number < 0 if an error occurs

Action	What it does	Value	Meaning of action return code from ButtonClicked event
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.

PowerBuilder offers many types of graphs and provides you with the ability to control the appearance of a graph to best meet your application's needs.

FOR INFO For information on using graphs, see Chapter 20, "Working with Graphs".

Adding OLE objects to a DataWindow object

On Windows, you can add the following to a DataWindow object:

- ◆ A column that contains a database binary large object (a blob object) using OLE 2.0
- ◆ OLE 2.0 objects

FOR INFO For information on using OLE in a DataWindow object, see Chapter 23, "Using OLE in a DataWindow Object".

Adding reports to a DataWindow object

You can nest reports (nonupdatable DataWindow objects) in a DataWindow object.

FOR INFO For information on nesting reports, see Chapter 19, "Using Nested Reports".

Positioning objects in a DataWindow object

Each object has several properties that determine how it is positioned within the DataWindow object:

Property	Meaning
Background	Object is behind other objects. 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	Object is placed within one band. It cannot extend beyond the band's border
Foreground	Object is in front of other objects. It is not restricted to one band
Moveable	Object can be moved during execution and in preview. This is useful for designing layout
Resizable	Object can be resized during execution and in preview. This is useful for designing layout
Suppress Print After First Newspaper Column	Object appears only in the first column on the page; in subsequent columns the object does not appear. This is only for newspaper columns, where the entire DataWindow object snakes from column to column

Default positioning

PowerBuilder uses the following defaults when placing new objects:

Object	Default positioning
Graph	Foreground, movable, resizable
All other objects	Band, not movable, not resizable

❖ **To change the position of an object in a DataWindow object:**

- 1 Select Properties from the object's popup menu and then select the Position tab.
- 2 From the Layer option dropdown listbox, select Background, Band, or Foreground.
- 3 Select Resizable or Moveable as appropriate.

Storing data in a DataWindow object

Usually you retrieve data into a DataWindow object during execution—the data is changeable and you want the latest.

But sometimes the data you display in a DataWindow object is static—it never changes. Or you may want a snapshot of the data at a certain point in time. In these situations, you may want to store the data in the DataWindow object itself. That way you don't need to go out to the database or other data source to display the data.

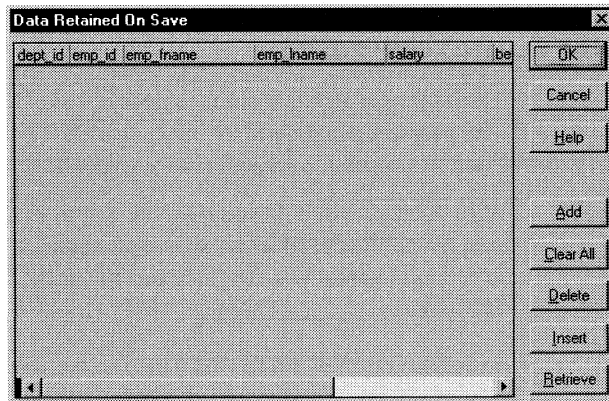
The most common reason to store data in a DataWindow object is for use as a dropdown DataWindow where the data is not coming from a database. For example, you might want to display a list of state postal codes for entering values in a State column in a DataWindow object. You can store those codes in a DataWindow object and use the dropdown DataWindow edit style for the State column.

FOR INFO For more information about using the dropdown DataWindow edit style, see Chapter 16, "Displaying and Validating Data".

❖ To store data in a DataWindow object:

- 1 Select Rows>Data from the menu bar.

The Data Retained On Save dialog box displays. All columns defined for the DataWindow object are listed at the top:



- 2 Do one of the following:
 - ◆ Click Add to create an empty row and type a row of data into the window. You can enter as many rows as you want.
 - ◆ Click Retrieve to retrieve all the rows of data from the database. If you want, you can delete rows you don't want to save or manually add new rows.

Data changes are local to the DataWindow object

Adding or deleting data here does not change the data in the database. It only determines what data will be stored with the DataWindow object when you save it.

- 3 Click OK.

When you save the DataWindow object, the data is stored in the DataWindow object.

Sharing DataWindow objects with other developers

Storing data in a DataWindow object is a good way to share data *and its definition* with other developers. They can simply open the DataWindow object on their computers to get the data and all its properties.

What happens during execution

Data stored in a DataWindow object is stored within the actual object itself. So when a window opens showing such a DataWindow, the data is already there. There is no need to issue Retrieve to get the data.

With one exception, if you *do* issue Retrieve for a DataWindow object stored with data, PowerBuilder handles it the same as a DataWindow object that is not stored with data: PowerBuilder gets the latest data by retrieving rows from the database.

The exception

PowerBuilder never retrieves data into a dropdown DataWindow that already contains data.

Retrieving rows as needed

If your DataWindow object retrieves hundreds of rows, there can be a noticeable delay during execution while all the rows are retrieved and before control returns to the user. In these DataWindow objects, you can have PowerBuilder retrieve only as many rows as it has to before displaying data and returning control to the user.

For example, if your DataWindow object displays only 10 rows at a time, it might make sense to have PowerBuilder retrieve only a small number of rows before presenting the data. Then as the user pages through the data, PowerBuilder continues to retrieve what is necessary to display the new information. There may be slight pauses while PowerBuilder retrieves the additional rows, but the pauses may be worth it if users do not need to wait a long time to start working with data.

❖ **To specify that a DataWindow object retrieve only as many rows as it needs to:**

- ◆ Select Rows>Retrieve>Rows As Needed from the menu bar.

With this setting, PowerBuilder presents data and returns control to the user when it has retrieved enough rows to display in the DataWindow object.

Retrieve Rows As Needed is overridden if you have specified sorting or have used aggregate functions, such as Avg and Sum, in the DataWindow object. This is because PowerBuilder must retrieve every row before it can sort or perform aggregates.

In a multiuser situation, Retrieve Rows As Needed might lock other people from the tables.

Saving retrieved rows to disk

Platform note

This feature is available on Windows 95, Windows NT, and UNIX only.

If you want to maximize the amount of memory available to PowerBuilder and other running applications, you can have PowerBuilder save retrieved data on your hard disk in a temporary file rather than keep the data in memory. PowerBuilder swaps rows of data from the temporary file into memory as needed to display data.

❖ **To maximize available memory by saving retrieved rows to disk:**

- ◆ Select Rows>Retrieve>Rows to Disk from the menu bar.

With this setting, when displaying data, PowerBuilder swaps rows of data from the temporary file into memory instead of keeping all the retrieved rows of data in memory.

Controlling updates

When PowerBuilder generates the basic DataWindow object, it defines whether the data is updatable by default as follows:

- ◆ If the DataWindow object contains columns from a single table and includes that table's key columns, PowerBuilder defines all columns as updatable and specifies a nonzero tab order for each column, allowing users to tab to the columns.
- ◆ If the DataWindow object contains columns from two or more tables or from a view, PowerBuilder defines all columns as not being updatable and sets all tab orders to zero, preventing users from tabbing to them.

You can accept the default settings or modify the update characteristics for a DataWindow object.

If using a Stored Procedure or External data source

If the data source is Stored Procedure or External, you can use the `GetNextModified` function to write your own update script.

FOR INFO For more information see the *PowerScript Reference*.

What you can do

You can:

- ◆ Allow updates in a DataWindow object associated with multiple tables or a view; you can define one of the tables as being updatable
- ◆ Prevent updates in a DataWindow object associated with one table
- ◆ Prevent updates to specific columns in a DataWindow object that is associated with an updatable table
- ◆ Specify which columns uniquely identify a row to be updated
- ◆ Specify which columns will be included in the `WHERE` clause of the `UPDATE` or `DELETE` statement PowerBuilder generates to update the database
- ◆ Specify whether PowerBuilder generates an `UPDATE` statement, or a `DELETE` then an `INSERT` statement, to update the database when users modify the values in a key column

Updatability of views

Some views are logically updatable; some are not.

FOR INFO For the rules your DBMS follows for updating views, see your DBMS documentation.

❖ To specify update characteristics for a DataWindow object:

- 1 Select Rows>Update Properties from the menu bar.

The Specify Update Properties dialog box displays.

- 2 To prevent updates to the data, make sure the Allow Updates box is not selected.

To allow updates, select the Allow Updates box and specify the other settings as described below.

- 3 Click OK.

PowerBuilder returns you to the workspace.

Changing tab values

PowerBuilder does not change the tab values associated with columns after you change the update characteristics of the DataWindow object. So if you have allowed updates to a table in a multitable DataWindow object, you should change the tab values for the updatable columns so users can tab to them.

FOR INFO For more information, see "Defining the tab order in a DataWindow object" on page 504.

Specifying the table to update

Each DataWindow object can update one table:



❖ To specify the table that can be updated using a DataWindow object:

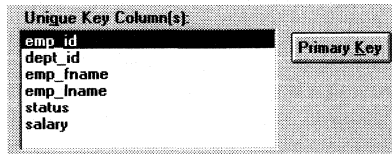
- 1 Select Rows>Update Properties from the menu bar.

The Specify Update Properties dialog box displays.

- 2 Select the table from the Table to Update box.

Specifying the unique key columns

The Unique Key Columns box in the Specify Update Properties dialog box specifies which columns PowerBuilder uses to identify a row being updated. PowerBuilder uses the column or columns you specify here as the key columns when generating the WHERE clause to update the database (as described below):



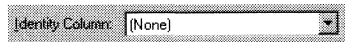
The key columns you select here must uniquely identify a row in the table. They can be the table's primary key, though they don't have to be.

Using the primary key

Clicking the Primary Key button cancels any changes in the Unique Key Columns box and highlights the primary key for the updatable table.

Specifying an identity column

Many DBMSs allow you to specify that the value for a column in a new row is to be automatically assigned by the DBMS. This kind of column is called an **identity column**. Different DBMSs provide different types of identity columns. For example, some DBMSs allow you to define autoincrement columns so that the column for a new row is automatically assigned a value one greater than that of the previous highest value. You could use this feature to specify that order number be automatically incremented when someone adds a new order:

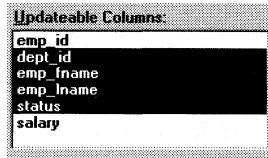


By specifying an identity column in the Specify Update Properties dialog box, you tell PowerBuilder to bring back the value of a new row's identity column after an insert in the DataWindow object so users can see it.

FOR INFO For information about identity columns in your DBMS, see your DBMS documentation.

Specifying updatable columns

You can make all or some of the columns in a table updatable:



Updatable columns are displayed highlighted. In the preceding screen, Dept_ID, Emp_Fname, Emp_Lname, and Status are updatable. The other columns are not.

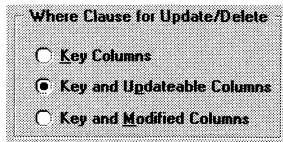
Click a nonupdatable column to make it updatable. Click an updatable column to make it nonupdatable.

Changing tab values

If you have changed the updatability of a column, you should change its tab value in the workspace. For example, if you have allowed a column to be updated, you should change its tab value to a nonzero number so users can tab to it.

Specifying the WHERE clause for update/delete

Sometimes multiple users are accessing the same tables at the same time. In these situations, you need to decide when to allow your application to update the database. If you allow your application to always update the database, it could overwrite changes made by other users:



You can control when updates succeed by specifying which columns PowerBuilder includes in the WHERE clause in the UPDATE or DELETE statement used to update the database:

```
UPDATE table...
SET column = newvalue
WHERE col1 value1
AND col2 = value2 ...
```

```

DELETE
FROM table
WHERE col1 = value1
AND col2 = value2 ...

```

Using timestamps

Some DBMSs maintain timestamps so you can ensure that users are working with the most current data. If the SELECT statement for the DataWindow object contains a timestamp column, PowerBuilder includes the key column and the timestamp column in the WHERE clause for an UPDATE or DELETE statement regardless of which columns you specify in the Where Clause for Update/Delete box.

If the value in the timestamp column changes (possibly due to another user modifying the row), the update fails.

FOR INFO To see whether you can use timestamps with your DBMS, see *Connecting to Your Database*.

Choose one of the following in the Where Clause for Update/Delete box (the results are illustrated by an example following the table):

Option	Result
Key Columns	<p data-bbox="552 894 1219 958">The WHERE clause includes the key columns only (these are the columns you specified in the Unique Key Columns box)</p> <p data-bbox="552 963 1219 1077">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</p> <hr/> <p data-bbox="552 1117 1219 1146">Caution</p> <p data-bbox="552 1151 1219 1299">Be very careful when using this option. If you tell PowerBuilder only to include the key columns in the WHERE clause, if someone else modified the same row after you retrieved it, their changes will be overwritten when you update the database (see the example following this table).</p> <hr/> <p data-bbox="552 1339 1219 1427">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</p>

Option	Result
Key and Updateable Columns	The WHERE clause includes all key and updatable columns The values in the originally retrieved key columns and the originally retrieved updatable columns are compared against the values in the database. If any of the columns have changed in the database since the row was retrieved, the update fails
Key and Modified Columns	The WHERE clause includes all key and modified columns The values in the originally retrieved key columns and the modified columns are compared against the values in the database. If any of the columns have changed in the database since the row was retrieved, the update fails

Example

Consider this situation: a DataWindow object is updating the Employee table, whose key is Emp_ID; all columns in the table are updatable. Say your user has changed the salary of employee 1001 from \$50,000 to \$65,000. This is what happens with the different settings for the WHERE clause columns:

- ◆ If you choose Key Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
```

This statement will succeed *regardless of whether other users have modified the row since your application retrieved the row*. For example, if another user had modified the salary to \$70,000, that change will be overwritten when your application updates the database.

- ◆ If you choose Key and Modified Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
AND Salary = 50000
```

Here the UPDATE statement is also checking the original value of the modified column in the WHERE clause. The statement will fail if another user changed the salary of employee 1001 since your application retrieved the row.

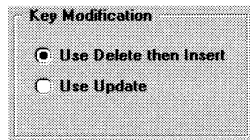
- ◆ If you choose Key and Updateable Columns for the WHERE clause, the UPDATE statement looks like this:

```
UPDATE Employee
SET Salary = 65000
WHERE Emp_ID = 1001
  AND Salary = 50000
  AND Emp_Fname = original_value
  AND Emp_Lname = original_value
  AND Status = original_value
...
```

Here the UPDATE statement is checking all updatable columns in the WHERE clause. This statement will fail if any of the updatable columns for employee 1001 have been changed since your application retrieved the row.

Specifying update when key is modified

The Key Modification property determines the SQL statements PowerBuilder generates whenever a key column—a column you specified in the Unique Key Columns box—is changed:



The options are:

- ◆ Use DELETE then INSERT (default)
- ◆ Use UPDATE

How to choose a setting

Consider the following when choosing the Key Modification setting:

- ◆ If multiple rows are changed, DELETE and INSERT always work. In some DBMSs, UPDATE will fail if the user modifies two keys and sets the value in one row to the original value of the other row.
- ◆ You might choose the setting here based on your DBMS triggers. For example, if there is an Insert trigger, you should select Use Delete then Insert.
- ◆ If only one row can be modified by the user before the database is updated, use UPDATE; it is faster.

About this chapter

This chapter describes how to customize your DataWindow object by modifying the display values in columns and specifying validation rules.

Contents

Topic	Page
About displaying and validating data	552
Working with display formats	554
Working with edit styles	566
Working with validation rules	582
Maintaining the entities	591

About displaying and validating data

When PowerBuilder generates a basic DataWindow object, it uses the extended attributes defined for the data and stored in the Powersoft repository.

FOR INFO For more information about the Powersoft repository, see the Appendix, "The Powersoft Repository".

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 repository.

Presenting the data

When you generate a new DataWindow object, PowerBuilder presents the data according to the properties already defined for a column, such as a column's display format and edit style.

Display formats

Display formats embellish data values while still displaying them as letters, numbers, and special characters. Using display formats, for example, you can:

- ◆ Change the color of numbers to display a negative value
- ◆ Add parentheses and dashes to format a telephone number
- ◆ Add a dollar sign and period to indicate a currency format

FOR INFO For information, see "Working with display formats" on page 554.

Edit styles

Edit styles take precedence over display formats and specify how column data is presented. For example, using edit styles, you can:

- ◆ Display valid values in a dropdown listbox
- ◆ Indicate that a single value is selected by a checkbox
- ◆ Indicate which of a group of values is selected with radio buttons

In the Database painter, an edit style enables users to *change* data in the database by means of an edit mechanism, such as a checkbox or radio buttons.

In the DataWindow painter, an edit style is simply a way of *presenting* data. You can change data in a DataWindow object; you cannot change data in a report.

FOR INFO For more information, see "Working with edit styles" on page 566.

Validating data

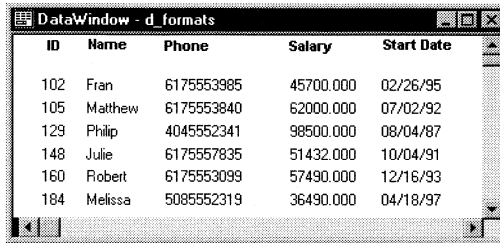
When data is entered in the Data Manipulation painter or in a DataWindow object, PowerBuilder evaluates the data against validation rules defined for that column. If the data is valid, PowerBuilder accepts the entry; otherwise, PowerBuilder displays an error message and does not accept the entry.

FOR INFO For more information, see "Working with validation rules" on page 582.

Working with display formats

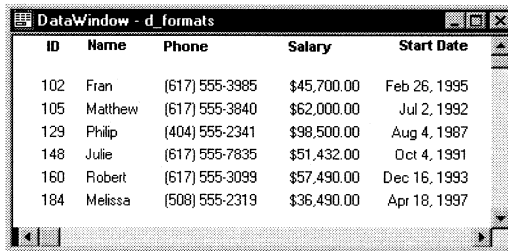
You can use display formats to customize the display of column data in a DataWindow object. Display formats are masks in which certain characters have special significance. For example, you can display currency values preceded by a dollar sign, show dates with month names spelled out, and use a special color for negative numbers. PowerBuilder comes with many predefined display formats. You can use them as is or define your own.

Here is a DataWindow object without any display formats; all values display as they are stored in the database:



ID	Name	Phone	Salary	Start Date
102	Fran	6175553985	45700.000	02/26/95
105	Matthew	6175553840	62000.000	07/02/92
129	Philip	4045552341	98500.000	08/04/87
148	Julie	6175557835	51432.000	10/04/91
160	Robert	6175553099	57490.000	12/16/93
184	Melissa	5085552319	36490.000	04/18/97

Here the Phone, Salary, and Start Date columns are using display formats so the data is easier to interpret:



ID	Name	Phone	Salary	Start Date
102	Fran	(617) 555-3985	\$45,700.00	Feb 26, 1995
105	Matthew	(617) 555-3840	\$62,000.00	Jul 2, 1992
129	Philip	(404) 555-2341	\$98,500.00	Aug 4, 1987
148	Julie	(617) 555-7835	\$51,432.00	Oct 4, 1991
160	Robert	(617) 555-3099	\$57,490.00	Dec 16, 1993
184	Melissa	(508) 555-2319	\$36,490.00	Apr 18, 1997

Display formats not used for data entry

When users tab to a column containing a display format, PowerBuilder removes the display format and displays the raw value for users to edit.

FOR INFO If you want to provide formatting used for data entry, you need to specify edit masks, as described in "Working with edit styles" on page 566.

Using display formats

You work with display formats in the Database painter, the Table painter, and the DataWindow painter.

What you do in the Database painter

In the Database painter, you can:

- ◆ Create, modify, and delete named display formats

The named display formats are stored in the repository. Once you define a display format, it can be used by any column of the appropriate data type in the database.

- ◆ Assign display formats to columns

These formats are used by default when you place the column in a DataWindow object in the DataWindow painter.

What you do in the Table painter

In the Table painter, you can:

- ◆ Assign display formats to columns
- ◆ Remove display formats from columns

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 or Table painter
- ◆ Override the default display format with another named format stored in the repository
- ◆ Create an ad hoc, unnamed format to use with one specific column

Display formats and the repository

Once you have placed a column in a DataWindow object and have given it a display format (either the default format from the assignment made in the Database or Table painter for the column or a format assigned in the DataWindow painter), there is no longer any link to the named format in the repository.

If the definition of the display format later changes in the repository, the format for the column in a DataWindow object will not change. If you want to use the modified format, you can reapply it to the column in the DataWindow painter.

Working with display formats in the Database painter

Typically, you define display formats and associate them with columns in the Database painter, because display formats are properties of the data itself. Once you have associated a display format with a column in the Database painter, it is used by default each time the column is placed in a DataWindow object.

Edit style takes precedence

If a column has an associated edit style, the edit style takes precedence over a display format.

FOR INFO For more information, see "Working with edit styles" on page 566.

❖ **To create or modify a display format:**

- 1 In the Database painter, open the table containing the column you want to apply a display format to.
- 2 Position the pointer on the column, select Properties from the column's popup menu, and select the Display tab.

All defined display formats for the corresponding data type are listed in the Display Format box. If the column has a display format defined, it is selected.

- 3 Click New to create a new format for the column.
or
Click Edit to modify the column's existing format.
- 4 If creating a new format, name it.
- 5 Define the display format using masks.

FOR INFO For information, see "Defining display formats" on page 559.

- 6 (Optional but recommended) Test the format by entering a test value and clicking the Test button.
- 7 Click OK to return to the Properties property sheet.
- 8 Click OK again to update the repository with your changes and to apply the format to the column.

You can use this display format with any column of the appropriate data type in the database.

Another way to create and modify a display format

Select Design>Display Format Maintenance from the menu bar to create and modify display formats.

When you create a display format this way, you are not creating the display format for a specific column and so you must tell PowerBuilder what type of column will use the display format.

❖ **To apply a display format to a column:**

- 1 In the Database painter, position the pointer on the column, select Properties from the popup menu, and select the Display tab.
All defined display formats for the corresponding data type display.
 - 2 Select a format from the list in the Display Format box and click OK.
The column now has the selected format associated with it in the repository.
-

Another way to apply display formats

In the Table painter, select a column and pick a display format from the Format dropdown listbox at the bottom of the workspace.

❖ **To remove a display format from a column:**

- 1 In the Database painter, position the pointer on the column, select Properties from the popup menu, and select the Display tab.
 - 2 Click the highlighted format and click OK.
The display format is no longer associated with the column.
-

Another way to remove display formats

In the Table painter, select a column and specify "(None)" in the Format dropdown listbox at the bottom of the workspace.

Working with display formats in the DataWindow painter

Display formats you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the default format in the DataWindow painter by choosing another format from the repository or defining an ad hoc format for one specific column.

About computed fields

You can assign display formats to computed fields using the same techniques as for columns in a table.

❖ **To specify a display format for a column in the DataWindow painter:**

- 1 In the DataWindow painter, move the pointer to the column, select Properties from the column's popup menu, and then select the Format tab.

Information appropriate to the data type of the selected column displays. The currently used format displays in the Format box. All formats for the data type defined in the repository are listed at the bottom.

- 2 Do one of the following:
 - ◆ Delete the display format.
 - ◆ Select a format in the repository from the list at the bottom of the dialog box.
 - ◆ 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 repository

If you create a format here, it is used only for the current column and is not saved in the repository.

- 3 Test the format by typing a value in the Test Value box and clicking Test.
- 4 Click OK to save your work.

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 Edit>Format from the menu bar and choose the format from the cascading menu.

Customizing the toolbar

You can add buttons to the PainterBar that assign a specified display format to selected columns in reports.

FOR INFO For more information, see Chapter 1, "Working with PowerBuilder".

Defining display formats

Display formats are represented through **masks**, where certain characters have special significance. PowerBuilder supports four kinds of display formats, each using different mask characters:

- ◆ Numbers
- ◆ Strings
- ◆ Dates
- ◆ Times

For example, in a string format mask, each @ represents a character in the string and all other characters represent themselves. So you can use the following mask to display phone numbers:

(@@@) @@@-@@@@

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 will be displayed in parentheses):

\$\$,##0;(\$,##0)

Using keywords

Enclose display format keywords in square brackets. For example, you can use the keyword [General] when you want PowerBuilder to determine the appropriate format for a number.

Using colors

You can define a color for each display format section by specifying a color keyword before the format. The color keyword is the name of the color, or a number that represents the color, enclosed in square brackets: [RED] or [255]. The number is usually used only when a color is required that is not provided by name.

The named color keywords are:

[BLACK]	[MAGENTA]
[BLUE]	[RED]
[CYAN]	[WHITE]
[GREEN]	[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.

The table below lists the blue, green, and red values you can use in the formula to create other colors:

Blue	Green	Red	Number	Color
0	0	255	255	Red
0	255	0	65280	Green
0	128	0	32768	Dark green
255	0	0	16711680	Blue
0	255	255	65535	Yellow
0	128	128	32896	Brown
255	255	0	16776960	Cyan
192	192	192	12632256	Light gray

Using special characters

To include a character in a mask that has special meaning in a display format, such as [, precede the character with a backslash (\). For example, to display a single quotation mark, enter \.

Setting display formats 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.

FOR INFO For more about the GetFormat and SetFormat functions, see the *PowerScript Reference*.

Number display formats

A number display format can have up to four sections, with only the first being required:

Positive-format;negative-format;zero-format>null-format

Special characters

Characters that have special meaning in number display formats are:

Character	Meaning
#	A number
0	A required number; a number will display for every 0 in the mask

Dollar signs, percent signs, decimal points, parentheses, and spaces display as entered in the mask.

International considerations

So that an application you build will run the same in whichever country it is deployed, masks (used in display formats and edit masks) and DataWindow expressions require U.S. notation for numbers. That is, when you specify a number in a DataWindow expression or in a number mask, a comma always represents the thousands delimiter and a period always represents the decimal place.

During execution, the locally correct symbols are displayed for numbers. For example, in countries where a comma represents the decimal place and a period represents thousands, users will see numbers in those formats during execution.

Number keyword

You can use the following keywords as number display formats when you want PowerBuilder to determine an appropriate format to use:

- ◆ [General]
- ◆ [Currency]

Examples

The following table shows how the values 5, -5, and .5 display when different format masks are applied:

Sample format	5	-5	.5
[General]	5	-5	0.5
0	5	-5	1
0.00	5.00	-5.00	0.50

Sample format	5	-5	.5
#,##0	5	-5	1
#,##0.00	5.00	-5.00	0.50
\$/,##0;(\$,##0)	\$5	(\$5)	\$1
\$/,##0;-\$,##0	\$5	-\$5	\$1
\$/,##0;[RED](\$,##0)	\$5	(\$5)	\$1
\$/,##0.00;(\$,##0.00)	\$5.00	(\$5.00)	\$0.50
\$/,##0.00;[RED](\$,##0.00)	\$5.00	(\$5.00)	\$0.50
0%	500%	-500%	50%
0.00%	500.00%	-500.00%	50.00%
0.00E+00	5.00E+00	-5.00E+00	5.00E-01

String display formats

String display formats can have two sections. The first is required and contains the format for strings; the second is optional and specifies how to represent NULLs:

string-format;null-format

In a string format mask, each at-sign (@) represents a character in the string and all other characters represent themselves.

Example

This format mask:

[red](@@@) @@@-@@@@

displays the string 800YESCELT in red as:

(800) YES-CELT

Date display formats

Date display formats can have two sections. The first is required and contains the format for dates; the second is optional and specifies how to represent NULLs:

date-format;null-format

Special characters

Characters that have special meaning in date display formats are:

Character	Meaning	Example
d	Day number with no leading zero	9
dd	Day number with leading zero if appropriate	09
ddd	Day name abbreviation	Mon
dddd	Day name	Monday
m	Month number with no leading zero	6
mm	Month number with leading zero if appropriate	06
mmm	Month name abbreviation	Jun
mmmm	Month name	June
yy	Two-digit year	97
yyyy	Four-digit year	1997

Colons, slashes, and spaces display as entered in the mask.

About 2-digit years

If users specify a 2-digit year in a DataWindow object, PowerBuilder assumes the date is the 20th century if the year is greater than or equal to 50. If the year is less than 50, PowerBuilder assumes the 21st century. For example:

1/1/85 is interpreted as January 1, 1985.

1/1/40 is interpreted as January 1, 2040.

Examples

The following table shows how the date Friday, January 30, 1998, displays when different format masks are applied:

Format	Displays
[red]m/d/yy	1/30/98 in red
d-mmm-yy	30-Jan-98
dd-mmmm	30-January
mmm-yy	Jan-98
dddd, mmm d, yyyy	Friday, Jan 30, 1998

Time display formats

Time display formats can have two sections. The first is required and contains the format for times; the second is optional and specifies how to represent NULLs:

time-format;null-format

Special characters

Characters that have special meaning in time display formats are:

Character	Meaning
h	Hour with no leading zero (for example, 1)
hh	Hour with leading zero if appropriate (for example, 01)
m	Minute with no leading zero (must follow h or hh)
mm	Minute with leading zero if appropriate (must follow h or hh)
s	Second with no leading zero (must follow m or mm)
ss	Second with leading zero (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

On Windows, you can use the following keyword as a time display format:

Keyword	Meaning
[Time]	The time format specified in the Microsoft Windows Control Panel
	<hr/> <p>On Windows 3.1 In addition to selecting the 12-hour radio button in the International Settings dialog box, users must also enter AM in the box on the 12-hour line and PM in the box on the 24-hour line</p> <hr/>

Examples


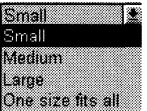




The following table shows how the time 9:45:33:234567 PM displays when different format masks are applied:

Format	Displays
h:mm AM/PM	9:45 PM
hh:mm A/P	09:45 P
h:mm:ss am/pm	9:45:33 pm
h:mm	21:45
h:mm:ss	21:45:33
h:mm:ss:f	21:45:33:2
h:mm:ss:fff	21:45:33:234
h:mm:ss:ffffff	21:45:33:234567
m/d/yy h:mm	1/30/98 21:45

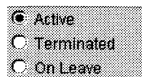
Working with edit styles

You can define edit styles for columns. Edit styles specify how column data is presented in DataWindow objects. Unlike display formats, edit styles don't only affect the display of data; they also affect how users interact with the data during execution. Once you define an edit style, it can be used by any column of the appropriate data type in the database.

You can choose from the following 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	
DropDownListBox	Displays a value from the dropdown listbox For data entry, select or enter a value	
CheckBox	Displays a checkbox selected or cleared For data entry, select or clear the checkbox	
RadioButtons	Displays radio buttons, one of which is selected For data entry, select one of the radio buttons	
Edit Mask	Displays formatted data For data entry, type a value	
DropDownDataWindow	Displays a value from a dropdown DataWindow For data entry, select a value	

For example, if you have a column Status that takes one of three values (A, T, or L), you might assign it the RadioButton edit style:



The Status data will be presented as radio buttons. Users can simply click a button instead of having to type A, T, or L (and you don't have to create a validation rule to validate the typed input).

Here's a DataWindow object that uses several edit styles: Edit Mask, DropDownListBox, CheckBox, and RadioButtons:

Using edit styles

You work with edit styles in the Database painter, Table painter, and DataWindow painter.

What you do in the Database painter

In the Database painter, you can:

- ◆ Create, modify, and delete named edit styles

The edit styles are stored in the repository. Once you define an edit style, it can be used by any column of the appropriate data type in the database.

- ◆ Assign edit styles to columns

These styles are used by default when you place the column in a DataWindow object in the DataWindow painter.

What you do in the Table painter

In the Table painter you can:

- ◆ Assign edit styles to columns and remove them from columns

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 or Table painter
- ◆ Override the default edit style with another named style stored in the repository
- ◆ Create an ad hoc, unnamed edit style to use with one specific column

Edit styles and the repository

Once you have placed a column in a DataWindow object and have given it an edit style (either the default style from the assignment made in the Database painter for the column or a style assigned in the DataWindow painter), PowerBuilder records the name and definition of the edit style in the DataWindow object.

However, if the definition of the edit style later changes in the repository, the edit style for the column in a DataWindow object will not change automatically. You can update the column by reassigning the edit style to it in the DataWindow object.

Working with edit styles in the Database painter

Typically, you define edit styles in the Database painter, because edit styles are properties of the data itself. Once defined in the Database painter, the styles are used by default each time the column is placed in a DataWindow object.

❖ To create or modify an edit style:

- 1 In the Database painter, position the pointer on the column, select Properties from the popup menu, then select the Edit Style tab.

All defined edit styles are listed in the Style Names box. If the column has an edit style defined, it is selected.

- 2 Click New to create a new edit style.

or

Click Edit to modify an existing edit style.

- 3 If creating a new edit style, select the edit style type from the Style dropdown listbox.

- 4 Specify the properties of the edit style.

FOR INFO For information, see "Defining edit styles" on page 570.

- 5 Click OK to return to the Edit Style property page.

- 6 Click OK again to apply the edit style to the column and to store it in the repository.

Another way to create and modify an edit style

Select Design>Edit Style Maintenance from the menu bar to create an edit style (independent of a column), or to modify or delete an existing edit style.

You can use the new or modified edit style with any column of the appropriate data type in the database.

❖ **To apply an existing edit style to a column:**

- 1 In the Database painter, position the pointer on the column, select Properties from the popup menu, then select the Edit Style tab.
- 2 Select a style for the appropriate data type and click OK.

PowerBuilder associates the selected edit style with the column in the repository.

Another way to apply edit styles

In the Database painter, select Definition from the column's popup menu to display the Table painter. Then select a column and pick an edit style from the Edit dropdown listbox at the bottom of the workspace.

❖ **To remove an edit style from a column:**

- 1 In the Database painter, position the pointer on the column, select Properties from the popup menu, then select the Edit Style tab.
- 2 Click the highlighted edit style and click OK.

The edit style is no longer associated with the column.

Another way to remove edit styles

In the Table painter, select a column and specify "(None)" in the Edit dropdown listbox at the bottom of the workspace.

Working with an edit style for a column in the DataWindow painter

Edit styles you assign to a column in the Database painter are used by default when you place the column in a DataWindow object. You can override the edit style in the DataWindow painter by choosing another edit style from the repository or defining an ad hoc style for one specific column.

❖ **To specify an edit style for a column:**

- 1 In the DataWindow painter, move the pointer to the column, select Properties from the column's popup menu, and then select the Edit tab.
- 2 Select the type of edit style you want from the Style dropdown listbox. The information on the Edit property page changes to be appropriate to the type of edit style you selected.
- 3 Do one of the following:
 - ◆ Select an edit style from the Name box.
 - ◆ Create an ad hoc edit style for the column, as described in "Defining edit styles" next.
- 4 Click OK to override the associated edit style.

Defining edit styles

This section describes how to specify each type of edit style.

The Edit edit style

By default, columns use the Edit edit style, which displays data in an edit control. You can customize the appearance and behavior of the edit control by modifying a column's Edit edit style. To do so, select Edit in the Style dropdown listbox and specify the properties for that style:

- ◆ To restrict the number of characters users can enter, enter a value in the Limit box.
- ◆ To convert the case of characters upon display, enter an appropriate value in the Case box.
- ◆ To have entered values display as asterisks for sensitive data, check the Password box.
- ◆ To allow users to tab to the column but not change the value, check the Display Only box.
- ◆ To define a code table to determine which values are displayed to users and which values are stored in the database, check the Use Code Table box and enter display and data values for the code table.

FOR INFO See "Defining a code table" on page 578.

❖ **To use the Edit edit style:**

- 1 Select Edit from the Style box, if it is not already selected.
- 2 Select the properties you want.

The DropDownListBox edit style

You can use the DropDownListBox edit style to have columns display as dropdown listboxes during execution:



Typically, this edit style is used with code tables, where you can specify display values (which users see) and shorter data values (which are stored in the database).

In the DropDownListBox edit style, the display values of the code table display in the ListBox portion of the DropDownListBox. The data values are the values that are put in the DataWindow buffer (and sent to the database when an Update is issued) when the user selects an item in the ListBox portion of the dropdown listbox.

In the preceding example, when users see the value Business Services, the corresponding data value could be 200.

❖ **To use the DropDownListBox edit style:**

- 1 Select DropDownListBox from the Style box.
- 2 Select the appropriate properties from the Options area.
- 3 Enter the value you want to appear in the Display Value box and the corresponding data value in the Data Value box.
- 4 Click the Add, Delete, and Insert buttons to modify the values as appropriate, then click OK when you are done.

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 INFO For information about the ClearValues, GetValue, and SetValue functions, see the *PowerScript Reference*.

For more information For more about code tables, see "Defining a code table" on page 578.

The CheckBox edit style

If a column can take only one of two (or perhaps three) values, you might want to display the column as a checkbox; users can select or clear the checkbox to specify a value. In the following entry from a DataWindow object, users can simply check or clear a box to indicate whether an employee has health insurance:

Health Insurance:

❖ To use the CheckBox edit style:

- 1 Enter in the Text box the text you want displayed next to the checkbox.

Using accelerator keys

If the CheckBox has an accelerator key, enter an ampersand (&) before the letter in the text that represents the accelerator key.

- 2 Enter in the Data Value For box the values you want put in the DataWindow buffer when the CheckBox is checked (on) or unchecked (off).

If you selected the 3 States box, an optional third state box (other) appears, for the case when the condition is neither on nor off.

- 3 Click OK when done.

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 checkbox during execution, PowerBuilder enters the appropriate data value in its buffer. When the Update function is issued, PowerBuilder sends the corresponding data values to the database.

Centering checkboxes without text

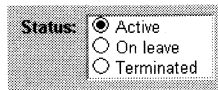
You may find it useful to center checkboxes used for columns of information. First make the text object used for the column header and the column object the same size and left aligned. Then you can center the checkboxes and the column header.

❖ **To center checkboxes without text:**

- 1 In the edit style (checkbox) property page for the column, make sure the Left Text checkbox is not selected and that the Text box where you specify associated text is empty.
- 2 Select the column object and specify centering using the Stylebar or by selecting Properties>General>Alignment>Center from the menu bar.

The RadioButtons edit style

If a column can take one of a small number of values, you might want to display the column as radio buttons:

❖ **To use the RadioButtons edit style:**

- 1 Specify how many radio buttons will display in the Columns Across box.
- 2 Enter a set of display and data values for each button you want to display.

The display values you enter become the text of the buttons; the data values are put in the DataWindow buffer when the button is clicked.

Using accelerator keys

To use an accelerator key on a radio button, enter an ampersand (&) in the Display Value before the letter that will be the accelerator key.

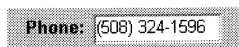
- 3 Click OK when done.

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:

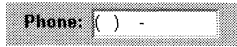


Edit masks consist of special characters that determine what can be entered in the column. They can also contain punctuation characters to aid users.

For example, to make it easier for users to enter 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:



Special characters

Special characters used in edit masks are the same ones used in display formats.

FOR INFO For information about special characters used in masks, see "Defining display formats" on page 559.

Keyboard behavior

Note the following about how certain keystrokes behave in edit masks:

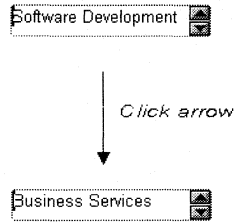
- ◆ Both BACKSPACE and SHIFT-BACKSPACE delete the preceding character
- ◆ DELETE deletes everything that is selected
- ◆ Non-numeric edit masks treat any characters that don't match the mask pattern as delimiters

Also, note certain behavior in the Date, Datetime, and Time edit masks:

- ◆ A nonzero number in the first position of dd, mm, hh, ss, or mm (minutes) followed by the delimiter character, will cause the typed digit to be preceded by 0. For example, typing **1/1/97** results in **01/01/97**.
- ◆ A number in the first position that is greater than the maximum day, month, hour, and so on, causes the last-typed number to be entered as the second position and the first position to be set to 0. The cursor is positioned in front of the second position. For example, entering **959595** results in **05/05/95**.
- ◆ The strings 00/00/00 or 00/00/0000 are interpreted as the NULL value for the column. If users enter 00 for either the month or the day, the other fields must also be zero or the date will be rejected as invalid.

Using spin controls

You can define an edit mask as a **spin control**, a box that contains up and down arrows that users can click to cycle through fixed values. For example, you can set up a code table that provides the valid entries in a column; users simply click an arrow to select an entry. Used this way, a spin control works like a dropdown listbox that displays one value at a time:



FOR INFO For more about code tables, see "Defining a code table" on page 578.

❖ **To use an EditMask edit style:**

- 1 Select EditMask in the Style box if it not already selected.
- 2 Define the mask in the Mask box. Click the special characters in the Masks box to use them.
- 3 Specify other properties for the edit mask.
- 4 (Optional) Test the mask by typing a value in the Test box.
- 5 Click OK when done.

When use your EditMask, check its appearance and behavior. If characters do not appear as you expect, you may 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.

Consider this situation: the Department table includes two columns, Dept_id and Dept_name, to record your company's departments. The Employee table records your employees. The Department column in the Employee table can have any of the values in the Dept_id column in the Department table.

As new departments are added to your company, you want the DataWindow object containing the Employee table to automatically provide the new departments as choices when users enter values in the Department column.

In situations such as these, you can specify the DropDownDataWindow edit style for a column: it is populated from another DataWindow object. When users go to the column, the contents of the DropDownDataWindow display, showing the latest data:

Department:	Business Services	*
	Name	Code
Status:	Software Development	100
	Business Services	200
	Corporate Marketing	300
Start date:	Marketing	400
Expiration date:		

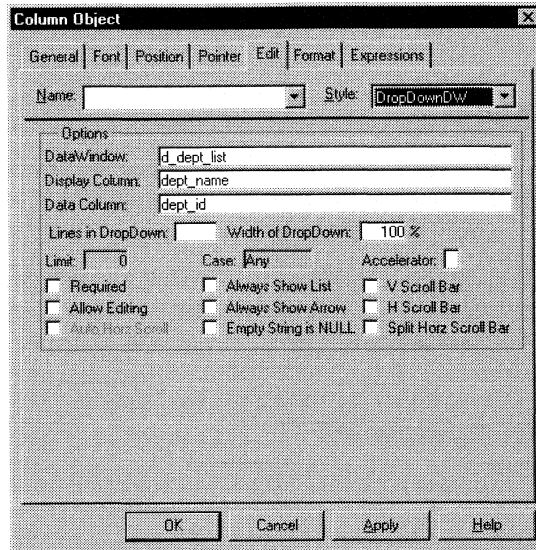
❖ **To use the DropDownDataWindow edit style:**

- 1 Create a DataWindow object that contains the columns in the detail band whose values you want to use in the column.

You will often choose at least two columns: one column that contains values that the user sees and another column containing values to be stored in the database. In the example above, you would create a DataWindow object containing the DeptID and DeptName columns in the Department table. Assume this DataWindow object is named `d_dept_list`.

- For the column getting its data from the DataWindow object, select the DropDownDataWindow edit style.

In the example, you would specify the DropDownDataWindow edit style for the Department column in the Employee table:



- In the DataWindow box, select the DataWindow object that contains the data for the column from the dropdown list (in the example, d_dept_list). (When you click the edit box, you will see the list.)
- In the Display Column box, select the column containing the values that will display in the DataWindow object (in the example, DeptName).
- In the Data Column box, select the column containing the values that will be stored in the database (in the example, DeptID).
- 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 as data is retrieved into the DataWindow object serving as the dropdown DataWindow object.

When the user goes to the column and drops it down, the contents of the dropdown DataWindow object display. When the user selects a display value, the corresponding data value is stored in the DataWindow buffer and is stored in the database when an Update is issued.

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 frequently want to store short, encoded values in the database. But these encoded values may 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 that are saved in the database.

How code tables are implemented

You can define a code table as a property of the following column edit styles:

- ◆ Edit
- ◆ DropDownListBox
- ◆ RadioButtons
- ◆ DropDownDataWindow
- ◆ EditMask, using spin control

The steps to specify the code table property for each edit style are similar: you begin by defining a new edit style in the Database painter. Once you select an edit style, use the following specific procedure to define the code table property.

FOR INFO For how to create an edit style, see "Working with edit styles" on page 566.

Allowing NULL values

An internal PowerBuilder code, NULL!, indicates NULL values are allowed. To use this code, specify NULL! as the data value, then specify a display format for NULLs for the column.

❖ **To define a code table as a property of the Edit edit style:**

- 1 Select the Use Code Table checkbox.
- 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 Using Code Table checkbox.

FOR INFO For more information, see "Validating user input" on page 580.
- 4 Click OK to accept the edit style definition.

❖ **To define a code table as a property of the DropDownList 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 checkbox.

FOR INFO For more information, see "Validating user input" on page 580.
- 3 Click OK to accept the edit style definition.

❖ **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 Click OK to accept the edit style definition.

❖ **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 checkbox.
- 4 Click OK to accept the edit style definition.

❖ **To define a code table as a property of the EditMask edit style:**

- 1 Select the Spin Control checkbox.
- 2 Select the Code Table checkbox.
- 3 Enter the display and data values for the code table.
- 4 Click OK to accept the edit style definition.

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 this example:

Display values	Data values
Massachusetts	MA
Massachusetts	ma
ma	MA
Mass	MA
Rhode Island	RI
RI	RI

If the data is MA or ma, the corresponding display value (Massachusetts) displays. If the data is Ma, there is no match, so Ma displays.

Case sensitivity

Code table processing is case sensitive.

If the code table is in a DropDownListBox edit style, and if the column has a code table that contains duplicate display values, then each value displays only once. So if this code table is defined for a column in a DataWindow object that has a DropDownListBox edit style, Massachusetts and Rhode Island display in the ListBox portion of the DropDownListBox.

Validating user input

When users enter data into a column in a DataWindow object, processing begins at the top of the display value column of the associated code table.

If the data matches a display value, the corresponding data value is put in the internal buffer. For each display value, the first data value is used. Using the sample code table, if the user enters Massachusetts, ma, or Mass, MA is the data value.

You can specify that *only* the values in the code table are acceptable:

- ◆ For a column using the Edit edit style, select the Validate Using Code Table checkbox.

If you have selected the Validate Using Code Table checkbox for the Edit edit style, an ItemError event is triggered whenever a user enters a value not in the code table. Otherwise, the entered value is validated using the column's validation rule, if any, and put in the DataWindow buffer.

- ◆ For the DropDownListBox and DropDownDataWindow edit styles, clear the Allow Editing checkbox: users cannot type a value.

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 data type as the column.

Working with 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, Table painter, or DataWindow painter.

Another technique

You can also perform data validation through code tables, which are implemented through a column's edit style.

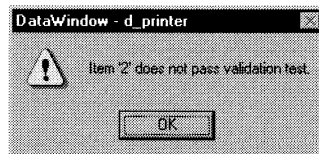
FOR INFO For more information, see "Working with edit styles" on page 566.

Understanding validation rules

Validation rules are criteria that a DataWindow object uses to validate data entered into a column by users. They are Powersoft-specific and therefore not enforced by the DBMS.

Validation rules assigned in the Database painter are used by default when you place columns in a DataWindow object. You can override the default rules in the DataWindow painter.

A validation rule is an expression that evaluates to either TRUE or FALSE. If the expression evaluates to TRUE for an entry into a column, PowerBuilder accepts the entry. If the expression evaluates to FALSE, the entry is not accepted and the ItemError event is triggered. By default, PowerBuilder displays a message box to the user:



You can customize the message displayed when a value is rejected.

You can also code an ItemError script to cause different processing to happen.

FOR INFO For more information, see the chapter on using DataWindow objects in *Application Techniques*.

During execution

In scripts, you can use the GetValidate function to obtain the validation rule for a column and the SetValidate function to change the validation rule for a column.

For information about the GetValidate and SetValidate functions, see the *PowerScript Reference*.

Using validation rules

You work with validation rules in the Database painter, Table painter, and DataWindow painter.

What you do in the Database painter

In the Database painter, you can:

- ◆ Create, modify, and delete named validation rules

The validation rules are stored in the repository. Once you define a validation rule, it can be used by any column of the appropriate data type in the database.

- ◆ Assign validation rules to columns

These rules are used by default when you place the column in a DataWindow object in the DataWindow painter.

What you do in the Table painter

In the Table painter, you can:

- ◆ Assign validation rules to columns and remove them from columns

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 or Table painter
- ◆ Create an ad hoc, unnamed rule to use with one specific column

Validation rules and the repository

Once you have placed a column that has a validation rule from the repository in a DataWindow object, there is no longer any link to the named rule in the repository.

If the definition of the validation rule later changes in the repository, the rule for the column in a DataWindow object will not change.

Defining validation rule properties in the Database painter

Typically, you will define validation rules in the Database painter, because validation rules are properties of the data itself. Once defined in the Database painter, the rules are used by default each time the column is placed in a DataWindow object.

This section describes the ways you can manipulate validation rules in the Database painter.

❖ To create or modify a validation rule:

- 1 In the Database painter, position the pointer on the column, display the column's popup menu, select Properties from the popup menu, then select the Validation tab.

All defined validation rules for the corresponding data type display. If the column has a validation rule associated with it, it is selected.

- 2 Click New to create a new rule.

or

Click Edit to modify an existing rule.

- 3 If creating a new rule, name it.

- 4 Define the expression for the validation rule.

FOR INFO For information, see "Defining the expression" on page 585.

- 5 (Optional) Customize the error message.

FOR INFO For information, see "Customizing the error message" on page 587.

- 6 Click OK to return to the Column property sheet.

- 7 Click OK again to apply the validation rule to the column and to store it in the repository.

You can use this rule with any column of the appropriate data type in the database.

Another way to create and modify a validation rule

Select Design>Validation Maintenance from the menu bar to create a new validation rule (independent of a column), or modify or delete an existing rule.

FOR INFO For information, see "Maintaining the entities" on page 591.

❖ To apply an existing validation rule to a column:

- 1 In the Database painter, position the pointer on the column, display the column's popup menu, select Properties, and select the Validation tab.

All defined validation rules for the corresponding data type display. If the column has a validation rule associated with it, it is selected.

- 2 Select a rule and click OK.

The column now has the selected rule associated with it in the repository. Whenever you use this column in a DataWindow object, it will use this validation rule unless you override it in the DataWindow painter.

Another way to apply a validation rule to a column

Select Definition from a table's popup menu, then select a column, and pick a validation rule from the Validation dropdown listbox in the Extended Attributes groupbox.

❖ To remove a validation rule from a column:

- 1 In the Database painter, position the pointer on the column, display the column's popup menu, select Properties, and select the Validation tab.

All defined validation rules for the corresponding data type display with the validation rule associated with the column highlighted.

- 2 Click the highlighted rule and click OK.

Defining the expression

A validation rule is a boolean expression. PowerBuilder applies the boolean expression to an entered value. If the expression returns TRUE, the value is accepted. Otherwise, the value is not accepted and an ItemError event is triggered.

What expressions
can contain

You can use any valid PowerScript expression in validation rules.

Validation rules can include any non-object-level PowerScript functions, including user-defined functions. DataWindow painter functions are displayed in the Functions listbox and can be pasted into the definition.

FOR INFO For information about these functions, see the *DataWindow Reference*.

Use the notation *@placeholder* (where *placeholder* is any group of characters) to indicate the current column in the rule. When you define a validation rule in the Database painter, PowerBuilder stores it in the repository with the placeholder name. During execution, PowerBuilder substitutes the value of the column for *placeholder*.

Pasting the placeholder

The name of the column for which you initially create the rule is frequently used as the placeholder. A button in the Paste area is labeled with the column name preceded by an @. You can click the button to paste the placeholder into the validation rule.

An example

For example, to make sure that both Age and Salary are greater than zero using a single validation rule, define the validation rule for one of the columns. If defining it for Salary, the expression would be:

```
@salary > 0
```

Then use the same validation rule for Age. At execution time, PowerBuilder substitutes the appropriate values for the column data when the rule is applied.

Using match values for character columns

If the column you are defining the validation rule for is a character column, the Match button is active, allowing you to match the contents of the column to a specified text pattern (for example, `^[0-9]+$` for all numbers and `^[A-Za-z]+$` for all letters).

❖ To specify a match pattern for character columns:

- 1 Click the Match button.

The Match Pattern dialog box displays.

- 2 Enter the text pattern you want to match the column to.

or

Select a displayed pattern.

- 3 (Optional) Enter a test value and click the Test button to test the pattern.
- 4 Click OK when you are satisfied that the pattern is correct.

FOR INFO For more on the Match function and text patterns, see the *DataWindow Reference*.

Customizing the error message

When you define a validation rule, PowerBuilder automatically creates the error message that displays by default when users enter an invalid value:

'Item ~" + @ *ColumnName* + '~' does not pass validation test.'

You can edit the string expression in the Validation Error Message textbox to create a custom error message.

Different syntax in the DataWindow painter

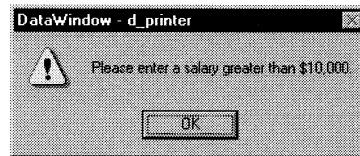
If you're working in the DataWindow painter, you can enter a string expression in the Error Message Expression textbox, but you don't use the @ sign for placeholders. For example, this is the default message:

'Item ~" + *ColumnName* + '~' does not pass validation test.'

A validation rule for the Salary column in the Employee table might have the following custom error message associated with it:

'Please enter a salary greater than \$10,000.'

If users enter a salary less than or equal to \$10,000, the custom error message displays:



Specifying initial values

As part of defining a validation rule, you can supply an initial value for a column.

❖ To specify an initial value for a column:

- 1 Select Properties from the column's popup menu and select the Validation tab.
- 2 Specify a value in the Initial Value box.

Defining a validation rule in the DataWindow painter

Validation rules you assign to a column in the Database or Table painter are used by default when you place the column in a DataWindow object. You can override the validation rule in the DataWindow painter by defining an ad hoc rule for one specific column.

❖ **To specify a validation rule for a column:**

- 1 In the DataWindow painter, select Properties from the column's popup menu, then select the Validation tab.

The Column Object property sheet displays.

- 2 Create or modify the validation expression, as described next.
- 3 (Optional) Enter a string or string expression to customize the validation error message.

FOR INFO For more information, see "Customizing the error message" on page 587.

- 4 Click Verify to verify the syntax.
- 5 Click OK.

Used for current column only

If you create a validation rule here, it is used only for the current column and is not saved in the repository.

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.

PowerBuilder does the conversion for you

If you have associated a validation rule for a column in the Database painter, PowerBuilder automatically converts the syntax to use GetText when you place the column in a DataWindow object.

GetText returns a string. Be sure to use a data conversion function (such as Integer or Real) if you want to compare the entered value with a data type other than string.

FOR INFO For more on the GetText function and text patterns, see the *DataWindow Reference*.

Referring to other columns

You can refer to the values in other columns by specifying their names in the validation rule. You can paste the column names in the rule using the Columns box.

Examples

Here are some examples of validation rules.

Example 1 To check that the data entered in the current column is a positive integer, use this validation rule:

```
Integer(GetText( )) > 0
```

Example 2 If the current column contains the discounted price and the column named Full_Price contains the full price, you could use the following validation rule to evaluate the contents of the column using the Full_Price column:

```
Match(GetText( ), "[0-9]+$") AND  
Real(GetText( )) < Full_Price
```

To pass the validation rule, the data must be all digits (must match the text pattern `^[0-9]+$`) and must be less than the amount in the Full_Price column.

Notice that to compare the numeric value in the column with the numeric value in the Full_Price column, the Real function was used to convert the text to a number.

Example 3 In your company, a product price and a sales commission are related in the following way:

- ◆ If the price is greater than or equal to \$1000, the commission is between 10 percent and 20 percent
- ◆ If the price is less than \$1000, the commission is between 4 percent and 9 percent

The Sales table has two columns, Price and Commission. The validation rule for the Commission column is:

```
(Number(GetText( )) >= If(price >= 1000, .10, .04))  
AND
```

```
(Number(GetText( )) <= If(price >= 1000, .20, .09))
```

A customized error message for the Commission column is:

```
"Price is " + if(price >= 1000,  
"greater than or equal to", "less than") +  
" 1000. Commission must be between " +  
If(price >= 1000, ".10", ".04") + " and " +  
If(price >= 1000, ".20.", ".09.")
```

Maintaining the entities

PowerBuilder provides facilities you can use to create, modify, and delete display formats, edit styles, and validation rules independently of their association with columns.

❖ **To maintain display formats, edit styles, and validation rules:**

- 1 Open the Database painter.
- 2 Select the appropriate entry from the Design menu:
 - ◆ Edit Style Maintenance
 - ◆ Display Format Maintenance
 - ◆ Validation Maintenance

A dialog box displays listing all the corresponding entities that are in the repository.

- 3 Do one of the following:
 - ◆ To create a new entity, click New.
 - ◆ To modify an existing entity, select it, then click Edit.
 - ◆ To delete an existing entity, select it, then click Delete.

Caution

If you delete a display format, edit style, or validation rule, it is removed from the repository. Columns in the database are no longer associated with the entity.

Filtering, Sorting, and Grouping Rows

About this chapter

This chapter describes how you can customize your DataWindow object by doing the following in the DataWindow painter:

- ◆ Defining filters to limit which of the retrieved rows are displayed in the DataWindow object
- ◆ Sorting rows after they have been retrieved from the database
- ◆ Displaying the rows in groups and calculating statistics on each group

Contents

Topic	Page
Filtering rows	594
Sorting rows	597
Grouping rows	600

Filtering rows

You can use WHERE and HAVING clauses and retrieval arguments in the SQL SELECT statement for the DataWindow object to limit the data that is retrieved from the database. This reduces retrieval time and space requirements during execution.

However, you may want to further limit the data that displays in the DataWindow object. For example, you might want to:

- ◆ Retrieve many rows and initially display only a subset (perhaps allowing the user to specify a different subset of rows to display during execution)
- ◆ Limit the data that is displayed using DataWindow painter 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 during execution. Filters can use most DataWindow painter functions or user-defined functions.

Filters don't affect which rows are retrieved

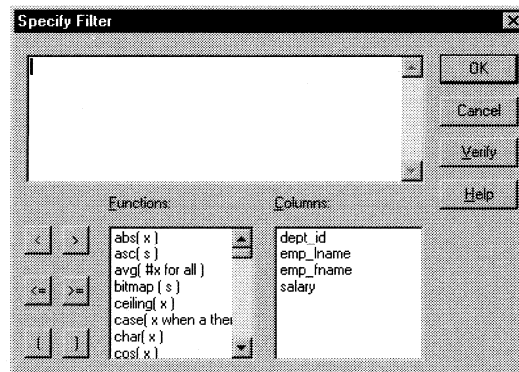
A filter operates against the retrieved data. It does not re-execute the SELECT statement.

Defining a filter

❖ To define a filter:

- 1 In the DataWindow painter, select Rows>Filter from the menu bar.

The Specify Filter dialog box displays:



- 2 In the Specify Filter dialog box, enter a boolean expression that PowerBuilder will test against each retrieved row.

If the expression evaluates to TRUE, the row will be 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

So that an application you build will run the same no matter in which country it is deployed, filter expressions require U.S. notation for numbers. That is, comma always represents the thousands delimiter and period always represents the decimal place when you specify expressions in the development environment.

FOR INFO For information about PowerScript expressions, see the *PowerScript Reference*.

- 3 (Optional) Click Verify to make sure the expression is valid.
- 4 Click OK to return to the workspace.
- 5 (Optional) Test the filter by clicking Preview.

Only rows meeting the filter criteria will be displayed.

Filtered rows and updates

Filtered rows are updated when you update the database.

Removing a filter

❖ **To remove a filter:**

- 1 Select Rows>Filter from the menu bar.
- 2 Delete the filter expression from the Specify Filter dialog box, then click OK.

Examples of filters

Assume that a DataWindow object retrieves employee rows. Three of the columns are Salary, Status, and Emp_Lname:

To display these rows	Use this filter
Employees with salaries over \$50,000	Salary > 50000
Active employees	Status = 'A'

To display these rows	Use this filter
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

FOR INFO You can use the `SetFilter` and `Filter` functions in a script to dynamically modify a filter that was set in the DataWindow painter.

FOR INFO For information about `SetFilter` and `Filter`, see the *PowerScript Reference*.

Sorting rows

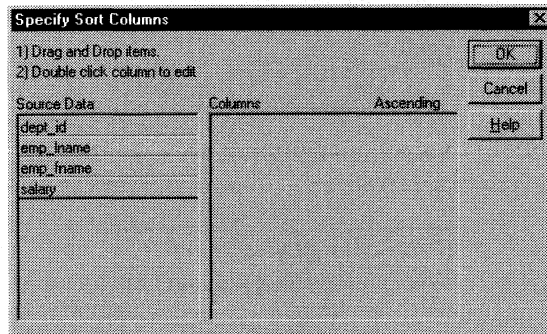
You can use an ORDER BY clause in the SQL SELECT statement for the DataWindow object to sort the data that is retrieved from the database. If you do this, the DBMS itself does the sorting and the rows are brought into PowerBuilder already sorted.

However, you might want to sort the rows after they are retrieved. For example you might want to:

- ◆ Offload the processing from the DBMS
- ◆ Sort on an expression, which is not allowed in the SELECT statement but is allowed in PowerBuilder

❖ **To sort the rows:**

- 1 Select Rows>Sort from the menu bar:

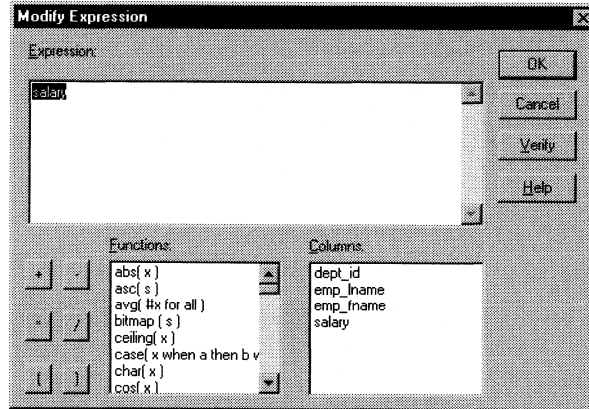


- 2 Drag the columns that you want to sort the rows on to the Columns box, and specify whether you want to sort in ascending or descending order.

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.

If you change your mind

You can remove a column or expression from the sorting specification by simply dragging it and releasing it outside the Columns box.

- 4 Click OK when you have specified all the sort columns and expressions.

Suppressing repeating values

When you sort on a column, you may have several rows with the same value in one column. In this case, you may want to suppress the repeating values in that column.

When you suppress a repeating value, the value displays at the start of each new page and, if you are using groups, each time a value changes in a higher group.

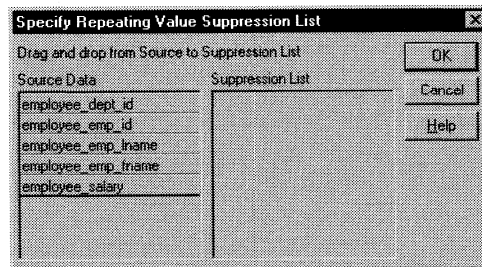
For example, if you have sorted employees by department ID, you might want to suppress all but the first occurrence of each department ID in the DataWindow object:

Dept	ID	Name	Salary
300	390	Davidson , Jo Ann	\$57,090
	586	Coleman , James	\$42,300
	757	Higgins , Denis	\$43,700
	879	Coe , Kristen	\$36,500
	1293	Shea , Mary Anne	\$138,948
	1336	Bigelow , Janet	\$31,200
	1390	Litton , Jennifer	\$58,930
	1483	Letiecq , John	\$75,400
400	184	Espinoza , Melissa	\$36,490
	207	Francis , Jane	\$53,870
	318	Crow , John	\$41,701
	409	Weaver , Bruce	\$46,550
	591	Barletta , Irene	\$45,450
	888	Charlton , Doug	\$28,300
	992	Butterfield , Joyce	\$34,011

❖ **To suppress repeating values:**

- 1 Select Rows>Suppress Repeating Values from the menu bar.

The Specify Repeating Value Suppression List dialog box displays:



- 2 Drag the columns whose repeated values you want to suppress from the Source Data box to the Suppression List box and click OK.

If you change your mind

You can remove a column from the suppression list 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

Total Compensation Report					Page 3 of 3			
Salary Plus Benefits					01/12/96			
Value of health ins. - \$4,000 Value of life insurance - \$25,000 (or salary 2X, 000) Value of day care - \$5,200								
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
400	1507	Ruth	Wetherby	\$35,745	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$40,739
	1576	Scott	Evans	\$68,940	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$74,114
	1607	Mark	Marrin	\$61,300	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$66,433
	1643	Elizabeth	Lambert	\$29,384	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$34,344
	1684	Janet	Hildebrand	\$45,829	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,878
	1740	Robert	Nielson	\$34,889	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$39,878
	1751	Alex	Ahmed	\$34,992	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$39,982
			Total:	\$698,251			Total:	\$784,442
			Average:	\$43,641			Average:	\$49,028
500	191	Jeannette	Bertrand	\$29,800	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$39,962
	703	Jane	Martinez	\$55,501	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,602
	750	Jane	Braun	\$34,300	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$39,286
	868	Felicia	Kua	\$28,200	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$33,153
	921	Charles	Crauley	\$41,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$46,726
	1013	Jaroph	Barker	\$27,290	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$27,438
	1570	Anthony	Robeira	\$34,576	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$39,564
	1615	Sheila	Ramona	\$27,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$37,649
	1658	Michael	Lynch	\$24,903	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$29,838
				Total:	\$303,770			Total:
			Average:	\$33,752			Average:	\$39,358
			Grand total:	\$3,749,147			Grand total:	\$4,170,705
			Overall average:	\$49,989			Overall average:	\$55,609

How to do it

You can create a grouped DataWindow object two ways:

- ◆ Use the Group presentation style to create a grouped DataWindow object from scratch.
- ◆ Take an existing tabular DataWindow object and define grouping in the DataWindow Object painter workspace ("Defining groups in an existing DataWindow object" on page 606).

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 customize the DataWindow object in the DataWindow painter workspace.

❖ To create a basic grouped DataWindow object using the Group presentation style:

- 1 Click the DataWindow button in the PowerBar.

The Select DataWindow dialog box displays listing the DataWindow objects in the current library.

- 2 Click the New button.

The New DataWindow dialog box displays.

- 3 Choose a data source and the Group presentation style, and click OK.

You are prompted to define the data for the DataWindow object.

- 4 Define the data.

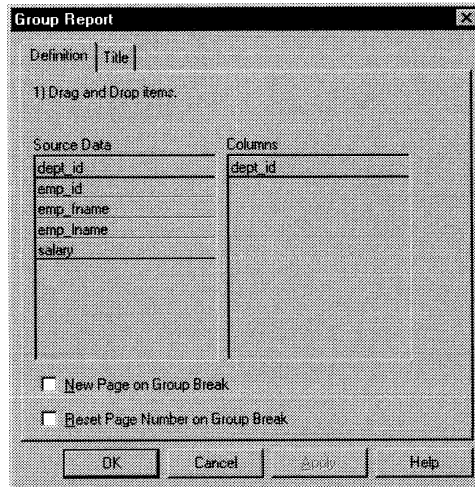
You are prompted to define the grouping column.

- 5 Drag the column(s) you want to group on from the Source Data box to the Columns box.

Multiple columns and multiple group levels

You can specify more than one column but all columns apply to group level one. You can define one group level at this point. Later in the workspace you can define additional group levels.

In the following example, grouping will be by department, as specified by the `dept_id` column:

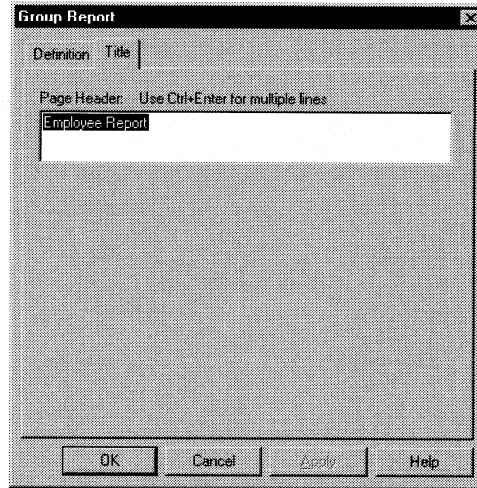


You can double-click a column name and type an expression as the grouping item. You can specify more than one grouping item expression for a group. A break occurs whenever the value concatenated from each column/expression changes.

- 6 If you want a page break each time a grouping value changes, select the New Page On Group Break box.
- 7 If you want page numbering to restart at 1 each time a grouping value changes, select the Reset Page Number On Group Break box.

- 8 Select the Title tab to provide a page header for the report.

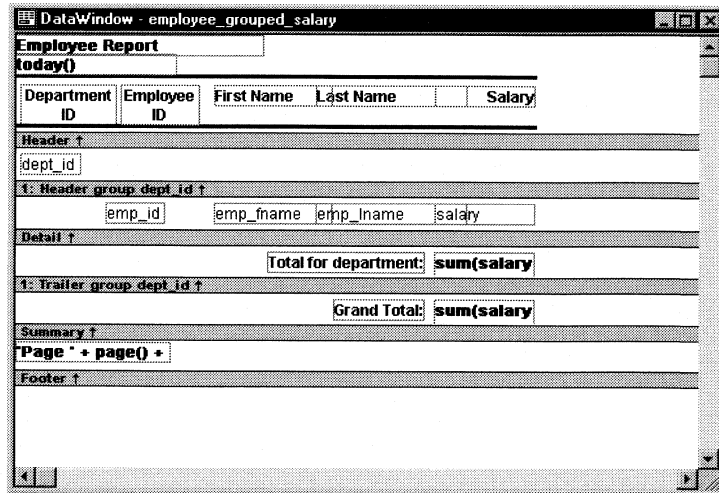
PowerBuilder suggests a header based on your data source. For example, if your data comes from the Employee table, PowerBuilder uses the name of the table in the suggested header:



- 9 Specify the text.
- 10 Click OK.

The workspace displays with the basic grouping properties set.

This is an example of a Group style DataWindow object in the workspace:



What PowerBuilder does

As a result of your specifications, PowerBuilder generates a tabular DataWindow object and:

- ◆ Creates group header and trailer bands
- ◆ Places the column you chose as the grouping column in the group header band
- ◆ Sorts the rows by the grouping column
- ◆ Places the page header and the date (as a computed field) in the header band
- ◆ Places the page number and page count (as computed fields) in the footer band
- ◆ Creates sum computed fields for all numeric columns (the fields are placed in the group trailer and summary bands)

Here is the preceding DataWindow object during preview:

Department ID	Employee ID	First Name	Last Name	Salary
100				
	102	Fran	Whitney	\$45,700
	105	Matthew	Cobb	\$62,000
	160	Robert	Breault	\$57,490
	243	Natasha	Shishov	\$72,995
	247	Kurt	Driscoll	\$48,024
	249	Rodrigo	Guevara	\$42,998
	266	Ram	Gowda	\$59,840
	278	Terry	Melkisetian	\$48,500
	316	Lynn	Pastor	\$74,500

Page 1 of 10

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 objects, specifying display formats, and so on. In particular, see "Defining groups in an existing DataWindow object" next to learn more about the bands in a grouped DataWindow object and how to add features especially suited for grouped DataWindow objects (for example, add a second group level, define additional summary statistics, and so on).

DataWindow object is not updatable by default

When you generate a DataWindow object using the Group presentation style, PowerBuilder makes it not updatable by default. If you want to be able to update the database through the grouped DataWindow object, you need to modify its update characteristics. For more information, see "Controlling updates" on page 543.

Defining groups in an existing DataWindow object

Instead of using the Group presentation style to create a grouped DataWindow object from scratch, you can take an existing tabular DataWindow object and define groups in it.

❖ To add grouping to an existing DataWindow object:

- 1 Start with a tabular DataWindow object that retrieves all the columns you need.
- 2 Specify the grouping columns.
- 3 Sort the rows.
- 4 (Optional) Rearrange the DataWindow object.
- 5 (Optional) Add summary statistics.
- 6 (Optional) Sort the groups.

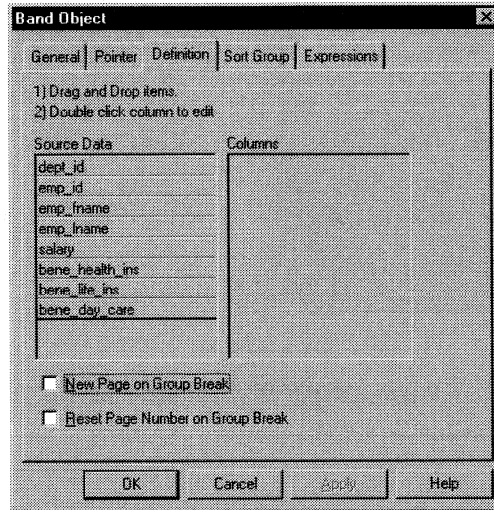
Steps 2 through 6 are described below.

Specifying the grouping columns

❖ To specify the grouping columns:

- 1 In the DataWindow painter, Select Rows>Create Group from the menu bar.

The Band Object property sheet displays:



- 2 Specify the group columns, as described in "Using the Group presentation style" on page 602.

Creating subgroups

After defining your first group, you can define subgroups, which are groups within the group you just defined.

❖ To define subgroups:

- 1 Select Rows>Create Group from the menu bar and specify the column/expression for the subgroup.
- 2 Repeat step 1 to define additional subgroups if you want.

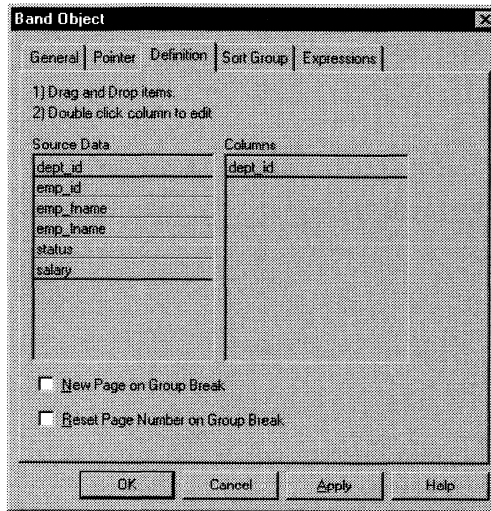
You can specify as many levels of grouping as you need.

How groups are identified

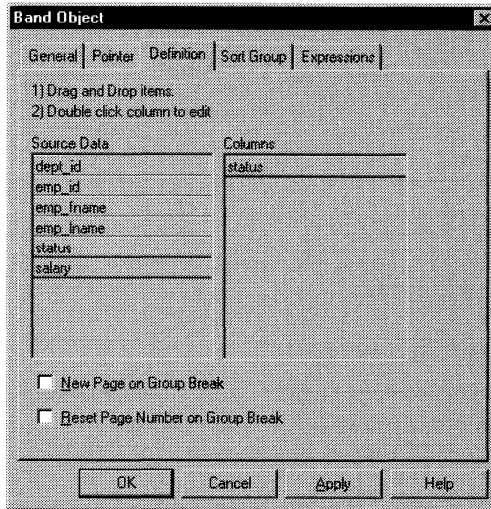
PowerBuilder assigns each group a number (or level) when you create the group. The first group you specify becomes group 1, the primary group. The second group becomes group 2, a subgroup within group 1, and so on.

For example, say you defined two groups. The first group uses the dept_id column and the second group uses the status column.

First group



Second group



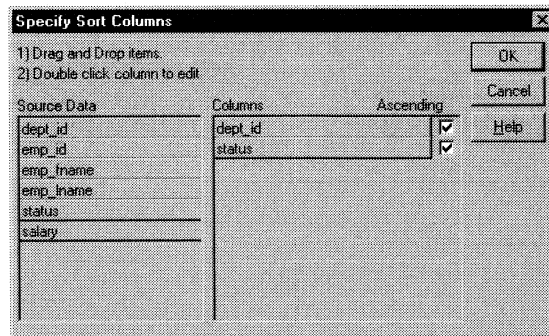
The rows will be grouped first by department (group 1). Within department, rows will be grouped by status (group 2). If you specify page breaks for the groups, a page break will occur when any of these values changes.

You use the group's number to identify it when defining summary statistics for the group, which is described later in this chapter.

Sorting the rows

PowerBuilder does not sort the data when it creates a group. Therefore, if the data source is not sorted, you must sort the data by the same columns (or expressions) specified for the groups.

For example, if you are grouping by Dept_ID then Status as shown above, select Rows>Sort from the menu bar and specify Dept_ID and then Status as sorting columns:



You can also sort on additional rows if you want. For example, if you want to sort by employee ID within each group, specify Emp_ID as the third sorting column.

FOR INFO For more information about sorting, see "Sorting rows" on page 597.

Rearranging the DataWindow object

When you create a group, PowerBuilder creates two new bands for each group:

- ◆ A group header band
- ◆ A group trailer band

The bar identifying the band contains:

- ◆ The number of the group
- ◆ The name of the band
- ◆ The name of each column that defines the group
- ◆ An arrow pointing to the band

The screenshot shows a data window with a table structure. The table has columns: Department ID, Employee ID, First Name, Last Name, Salary, Health Insurance, and Life Insurance. The table is divided into several bands:

- Header ↑**: A shaded band at the top.
- 1: Header group dept_id ↑**: A band containing a table with columns: dept_id, emp_id, emp_fname, emp_lname, salary, Health Insurance, and Life Insurance.
- Detail ↑**: A band for the main data rows.
- 1: Trailer group dept_id ↑**: A band at the bottom of the data rows.
- Summary ↑**: A band below the trailer.
- Footer ↑**: A shaded band at the very bottom.

You can include any object 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. For example, you might move the grouping column from the detail band to the group header band, since it now serves to identify one group rather than each row.

For example, if you group the rows by department and include the department in the group header, the department will display before the first line of data each time the department changes.

The screenshot shows a report design tool interface. At the top, there is a table with columns: Employee ID, First Name, Last Name, Salary, Health Insurance, and Life Insurance. Below this, there are several report bands:

- Header** band: Contains a 'Department ID' label and a 'dept_id' field.
- 1: Header group dept_id** band: Contains fields for 'emp_id', 'emp_fname', 'emp_lname', 'salary', 'Health Insurance', and 'Life Ins'.
- Detail** band: The main data area.
- 1: Trailer group dept_id** band: A band that appears at the end of each group.
- Summary** band: A band for overall report totals.
- Footer** band: The final band at the bottom.

During execution, you see this:

The screenshot shows the report execution. The data is grouped by 'Department ID'. The first group is for 'Department ID 100'. The data rows are as follows:

Employee ID	First Name	Last Name	Salary	Health Insurance	Life Insurance
862	John	Sheffield	\$87,900.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
102	Fran	Whitney	\$45,700.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
582	Peter	Samuels	\$37,400.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
479	Linda	Siperstein	\$39,875.50	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
160	Robert	Breault	\$57,490.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
278	Terry	Melkisetian	\$48,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
249	Rodrigo	Guevara	\$42,998.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
316	Lynn	Pastor	\$74,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
839	Dean	Marshall	\$42,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
243	Natasha	Shishov	\$72,995.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
1250	Emilio	Diaz	\$54,900.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Using the group trailer band

The contents of the group trailer display after the last row for each value that causes a break.

In the group trailer band, you specify the information you want displayed after the last line of identical data for each value in the group. Typically, you include summary statistics here, as described next.

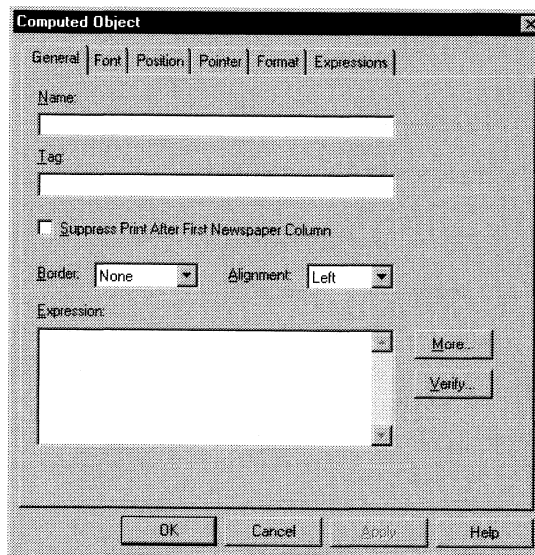
Adding summary statistics

One of the advantages of creating a grouped DataWindow object is that you can have PowerBuilder calculate statistics for each group. To do that, you place computed fields that reference the group. Typically, you place these computed fields in the group's trailer band.

❖ **To add a summary statistic:**

- 1 Click the Computed Field button in the Objects dropdown toolbar.
- 2 Click the workspace where you want the statistic.

The Computed Object property sheet displays:



- 3 (Optional) Name the computed field (this allows you to reference the computed field later if you need to).
- 4 Specify the expression that defines the computed field (see below).
- 5 Click OK.

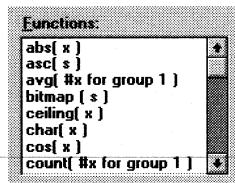
A shortcut to sum values

If you want to sum a numeric column, select the column in the workspace and click the Sum button in the Objects dropdown toolbar.

PowerBuilder automatically places a computed field in the appropriate band in the workspace.

Specifying the expression

Typically, you will use aggregate and other functions in your summary statistic. PowerBuilder lists functions you can use in the Functions box in the Modify Expression dialog box (to display this dialog box, click More in the Computed Object property sheet). When you are defining a computed field in a group header or trailer band, PowerBuilder automatically lists forms of the functions that reference the group:



You can paste these templates into the expression, then replace the #x that is pasted in as the function argument with the appropriate column or expression.

For example, to count the employees in each department (group 1), specify this expression in the group trailer band:

```
Count( Emp_Id for group 1 )
```

To get the average salary of employees in a department, specify:

```
Avg( Salary for group 1 )
```

To get the total salary of employees in a department, specify:

Sum(Salary for group 1)

Employee ID	First Name	Last Name	Salary
Header			
Department ID			
dept_id			
Header group dept_id			
emp_id	emp_fname	emp_lname	salary
Detail			
Average Salary:			avg(salary for
Total Salary:			sum(salary for
Footer group dept_id			
Summary			
Footer			

During execution, you see this:

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

Sorting the groups

You can sort the groups in your DataWindow object. For example, in a DataWindow object showing employee information grouped by department, you might want to sort the departments (the groups) by total salary.

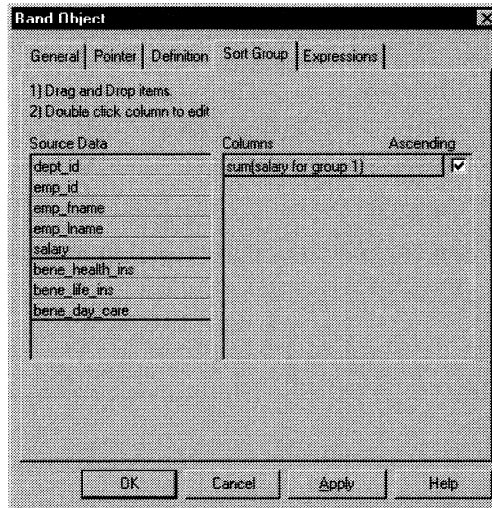
Typically, this involves aggregate functions, as described in "Adding summary statistics" on page 612. In the department salary example, you would sort the groups using the aggregate function Sum to calculate total salary in each department.

❖ **To sort the groups:**

- 1 Place the mouse pointer on the group header bar (not inside the band).
The pointer becomes a double-headed arrow.
- 2 Display the popup menu for the group header, select Properties, and then select the Sort Group tab.
- 3 Drag the column you want to sort the groups by from the Source Data box into the Columns box.

If you chose a numeric column, PowerBuilder uses the Sum function in the expression; if you chose a non-numeric column, PowerBuilder uses the Count function.

For example, if you chose the Salary column, PowerBuilder specifies that the groups will be sorted by the expression **sum(salary for group 1)**:

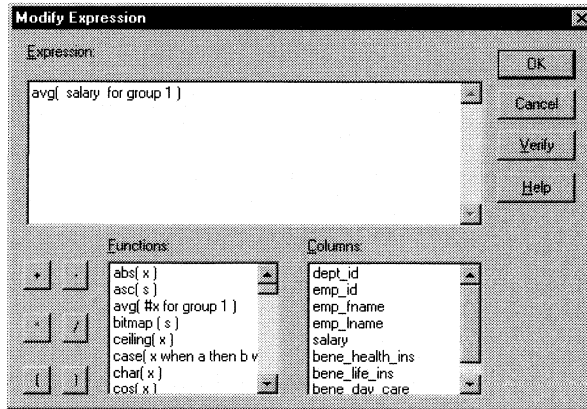


- 4 Select ascending or descending sort as appropriate.
- 5 If you want to modify the expression to sort on, double-click the column in the Columns box.

The Modify Expression dialog box displays.

- 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)**:



- Click OK.

You return to the Band Object property sheet with the expression displayed.

- Click OK.

During execution, the groups will be sorted on the expression you specified.

Highlighting Information in DataWindow Objects

About this chapter

This chapter describes how you modify the way information displays in your DataWindow objects and reports based on the conditions you specify. The conditions are usually related to data values, which are not available until execution time.

A report is the same as a nonupdatable DataWindow object. You can use the techniques described in this chapter in the DataWindow painter or the Report painter to highlight information in DataWindow objects or reports.

Contents

Topic	Page
Overview of highlighting information	618
Conditionally modifying properties at execution time	622
Example 1: creating a gray bar effect	624
Example 2: rotating objects	627
Example 3: highlighting rows of data	629
Example 4: changing the size and location of objects	632
Supplying property values	635
Specifying colors	657

Overview of highlighting information

About properties

Every object in your DataWindow object has a set of *properties* that determines what the object 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.

About modifying properties in the workspace

You define the appearance and behavior of these objects in DataWindow objects in the DataWindow painter. As you do that, you are specifying the objects' properties. For example, when you place a border around a column in the painter workspace, you are setting that column's Border property.

In most cases, the appearance and behavior of objects is fixed; you do not want them to change at execution time. You make headings bold in the workspace and that's the way you want them to be 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 would set the border property on the column's property sheet:

Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$93,177
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$89,650
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$44,892

About modifying properties at execution time

In some cases, however, you might want some properties of objects in DataWindow objects to be driven by the data, which is not known when you are defining the DataWindow object in the painter. For these situations you can define **property conditional expressions**, which are expressions that are evaluated at execution time.

You can use these expressions to conditionally and dynamically modify the appearance and behavior of your DataWindow object during execution. The results of the expressions set the values of properties of objects in the DataWindow object.

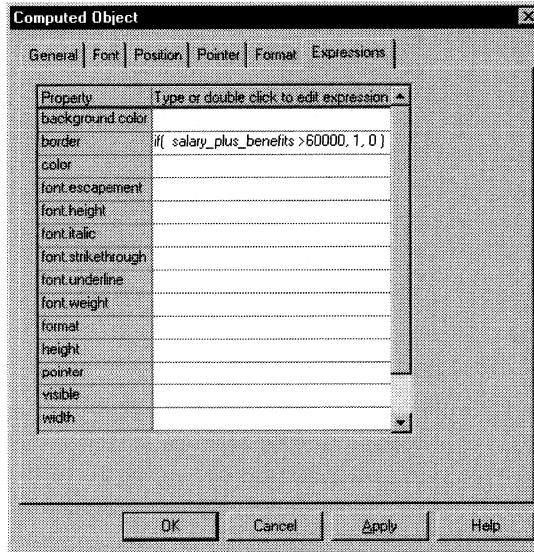
In the following DataWindow object, the Salary Plus Benefits column has a Shadow box border highlighting each data value that is greater than \$60,000:

Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$87,137
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,906
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$93,177
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$69,650
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$44,892

To control the display of the border, you would enter a property conditional expression on the Expressions property page of the column's property sheet. When users run the DataWindow object, PowerBuilder changes the border of individual data values based on your expressed condition (value greater than \$60,000).

About filling in the Expressions property page

The following Expressions property page lists properties for the Salary Plus Benefits column, which is a computed object. In this example, you want to change the Border property, so you put the conditional expression next to Border:



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's true, and what happens if it's false. Parentheses surround the three things and commas separate them:

If(expression, true, false)

This is the expression used in the example. Because the expression is next to the Border property, the values for true and false indicate particular borders. The value 1 means Shadow box border and the value 0 means no border:

If(salary_plus_benefits > 60000, 1, 0)

When users run the DataWindow object, PowerBuilder checks the value in the computed column called salary_plus_benefits to see if it is greater than 60000. If it is (true), PowerBuilder will display the value with the Shadow box border. If not (false), PowerBuilder will display the value with no border.

About specifying properties

Usually you specify a number to indicate what you want for a particular property. For example, the following list shows all of the borders you can specify and the numbers you use. If you want the border property to be Shadow box, you specify **1** in the If statement, for either true or false.

- 0**—None
- 1**—Shadow box
- 2**—Box
- 3**—Resize
- 4**—Underline
- 5**—3D Lowered
- 6**—3D Raised

FOR INFO For details on the values of properties that can be set on the Expressions property page, see "Supplying property values" on page 635. For complete information about what the valid values are for all DataWindow object properties, see the discussion of DataWindow object properties in the *DataWindow Reference* or online Help.

About modifying properties programmatically

You can also programmatically modify the properties of a DataWindow object in a script during execution.

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

Conditionally modifying properties at execution time

Conditionally modifying properties at execution time is a technique you can use to highlight information in a DataWindow object.

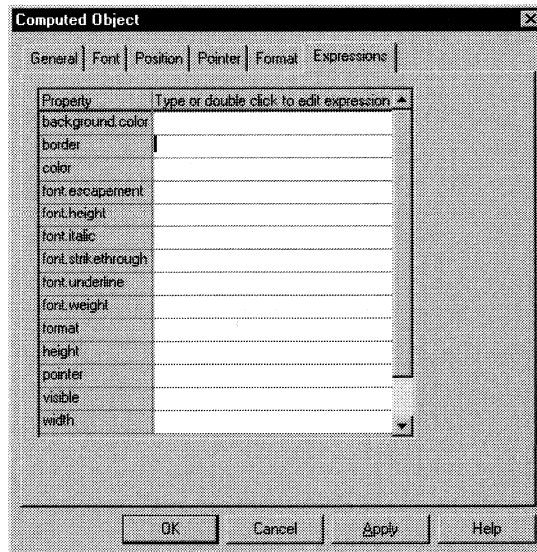
FOR INFO For an explanation of this technique, see "Overview of highlighting information" on page 618.

❖ **To conditionally modify properties at execution time:**

- 1 Position the pointer on the object whose properties you want to modify during execution.
- 2 Select Properties from the object's popup menu and then select the Expressions property page.
- 3 Enter the appropriate expression next to the property you want to change.

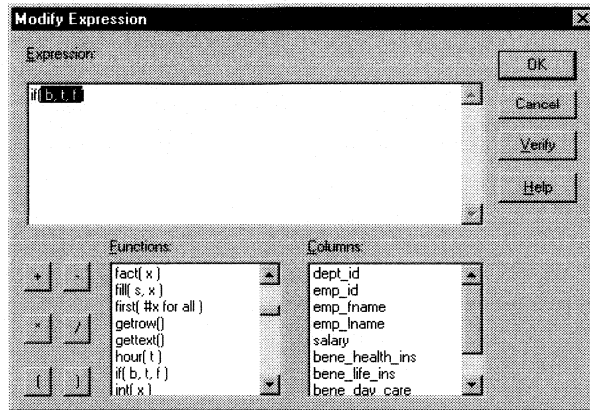
An easy way to enter or change an expression is to use the Modify Expression dialog box.

- 4 To access the Modify Expression dialog box, double-click the line containing the property for which you want to enter an expression:



The Modify Expression dialog box displays.

- 5 Scroll the list of functions in the Functions box until you see the IF function and then select it:



- 6 Replace the *b* (boolean) with your condition (for example, salary>40000).

You can select columns and functions and use the buttons to add the symbols shown on them.

Replace the *t* (true) with the value to use for the property if the condition is true.

Replace the *f* (false) with the value to use for the property if the condition is false.

Values to use for properties are usually numbers. They are different for each property.

FOR INFO For more information about property values that can be set on the Expressions page, see "Supplying property values" on page 635. For complete information about what the valid values are for all DataWindow object properties, see the discussion of DataWindow object properties in the *DataWindow Reference* or online Help.

- 7 Click OK to return to the Expressions property page.
- 8 If you want, enter expressions for other properties of the object.
- 9 Click OK.

You return to the DataWindow painter workspace.

FOR INFO For examples, see "Example 1: creating a gray bar effect" next, "Example 2: rotating objects" on page 627, "Example 3: highlighting rows of data" on page 629, and "Example 4: changing the size and location of objects" on page 632.

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:

Total Compensation Report Salary Plus Benefits								Page 1 of 4 01/10/98
Value of life insurance = \$(5.43 x salary)/1,000 Value of day care = \$5,200								
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
100	102	Fran	Whitney	\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
	105	Matthew	Cobb	\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
	160	Robert	Breault	\$57,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
	243	Natasha	Shishov	\$72,995	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
	247	Kurt	Driscoll	\$48,024	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
	249	Rodrigo	Guevara	\$42,998	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
	266	Ram	Gowda	\$59,840	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
	278	Terry	Melkisetian	\$48,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
	316	Lynn	Pastor	\$74,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905

Adding a rectangle object

To create the gray bar effect, you would add a rectangle object to the detail band and size it so that it surrounds the objects you want highlighted. The rectangle needs to be located in the band (Rectangle property sheet>Position tab>Layer option>Band setting). The rectangle also needs to be specified as Send to Back (select from the pop up menu) so that it will be behind all the values:

Header_group_dept_id_f								
dept_id	emp_id	emp_fname	emp_lname	salary	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	salary + ff()
Detail								

At this point, the DataWindow object shows a rectangle for each detail line. A narrow black line bounds the rectangle:

Total Compensation Report Salary Plus Benefits								Page 1 of 4 01/10/98
Value of life insurance = \$(5.43 x salary)/1,000 Value of day care = \$5,200								
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
100	102	Fran	Whitney	\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
	105	Matthew	Cobb	\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
	160	Robert	Breault	\$57,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
	243	Natasha	Shishov	\$72,995	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
	247	Kurt	Driscoll	\$48,024	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
	249	Rodrigo	Guevara	\$42,998	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
	266	Ram	Gowda	\$59,840	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
	278	Terry	Melkisetian	\$48,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
	316	Lynn	Pastor	\$74,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905

The line bounding the rectangle should be set to None (Rectangle property sheet>General tab>Line option>None setting). You would do this last, after positioning the rectangle object.

At this point the DataWindow object looks as it did originally without the rectangles in the detail lines. The rectangles are there but because they are white and are not bounded with a black line, you cannot see them.

Specifying the condition

Now you would specify the condition for making the rectangle gray. To do this, you would select Properties from the popup menu for the rectangle. On the Expressions property page, you would enter the following expression for the Brush.Color property:

```
If(mod(getrow(),2)=1, rgb(255, 255, 255 ), rgb(240, 240, 240 ))
```

The expression $\text{mod}(\text{getrow}(),2)=1$ distinguishes odd rows from even rows.

The function $\text{mod}(\text{getrow}(),2)$ takes the row number ($\text{getrow}()$), 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.

If the row number is odd (the condition evaluates as true), the rectangle displays as white. The rgb function specifies maximum amounts of red, green, and blue: $\text{rgb}(255, 255, 255)$. Specifying 255 for red, green, and blue results in the color white. If the row number is even (the condition evaluates as false), the rectangle displays as light gray ($\text{rgb}(240, 240, 240)$).

Finishing up

Now the DataWindow object has gray bands. The final adjustment is to change the background color of the computed field used for Salary Plus Benefits. In the workspace, change the background color of the computed field from white to transparent (Computed object property sheet>Font tab>Background option>Transparent):

Total Compensation Report								Page 1 of 4
Salary Plus Benefits								01/10/98
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
100	102	Fran	Whitney	\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
	105	Matthew	Cobb	\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
	160	Robert	Breault	\$57,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
	243	Natasha	Shishov	\$72,985	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
	247	Kurt	Driscoll	\$48,024	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
	249	Rodrigo	Guevara	\$42,988	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
	266	Ram	Gowda	\$59,840	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
	278	Terry	Meikisetian	\$48,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
	316	Lynn	Pastor	\$74,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905

Value of life insurance = \$(5.43 x salary)/1,000
Value of day care = \$5,200

Example 2: rotating objects

The following DataWindow object shows column headers rotated 45 degrees.

On Macintosh rotation is limited

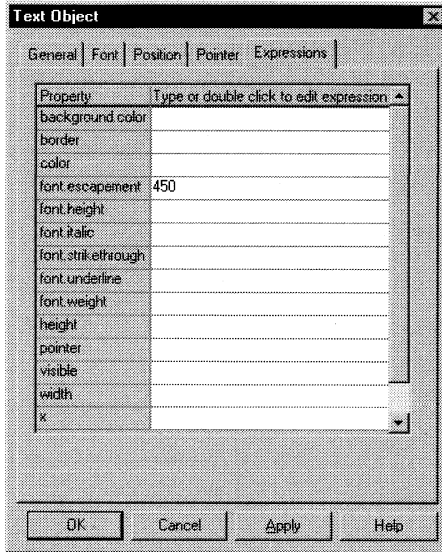
On the Macintosh, you can rotate a text object 90 degrees only. Other values are ignored.

Total Compensation Report				Value of health ins. = \$4,800 Value of life insurance = \$(5.43 x salary)/1,000 Value of day care = \$5,200			Page 1 of 4	
Salary Plus Benefits							01/10/98	
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
100	102	Fran	Whitney	\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
	105	Matthew	Cobb	\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
	160	Robert	Breaut	\$57,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
	243	Natasha	Shishov	\$72,995	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
	247	Kurt	Driscoll	\$48,024	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
	249	Rodrigo	Guevara	\$42,998	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
	266	Ram	Gowda	\$59,840	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
	278	Terry	Melkisetian	\$48,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
	316	Lynn	Pastor	\$74,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905

To rotate each of these three text objects, you would select Properties from the popup menu for each object and make an entry on the Expressions property page.

Specifying rotation

To specify rotation, you would enter the number 450 next to the Font.Escapement property. The number 450 means 45 degrees; the value entered for Font.Escapement is in tenths of degrees:



Notice that you do not have to specify a condition. Typically, you would not specify a condition for object rotation. When you have specified 450 for the three text objects, you end up with all three objects rotated:

Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
\$57,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802

If you try rotation, you may notice that the objects do not appear to be rotated in the workspace. You see the rotation only when you run the DataWindow object.

Example 3: highlighting rows of data

The following DataWindow object is an employee phone list. Out-of-state (not in Massachusetts) employees are shown in bold and preceded by two asterisks (**):

<i>(** means out of state)</i>			
Employee Phone List			
01/10/98			
Employee Name	Phone	Employee Name	Phone
Ahmed, Alex	(617) 555-8748	** Overbey, Rollin	(510) 555-7255
Barker, Joseph	(617) 555-8021	Pastor, Lynn	(617) 555-2001
Barietta, Irene	(617) 555-8345	Pickett, Catherine	(617) 555-3478
Bertrand, Jeannette	(508) 555-8138	Poitras, Kathleen	(617) 555-3920
Bigelow, Janet	(617) 555-1493	Powell, Thomas	(617) 555-1956
Blaikie, Barbara	(617) 555-9345	Preston, Mark	(617) 555-5862
Braun, Jane	(617) 555-7857	Rabkin, Andrew	(617) 555-4444
Breault, Robert	(617) 555-3099	Rebeiro, Anthony	(617) 555-5737
Bucceri, Matthew	(617) 555-5336	Romero, Sheila	(617) 555-8138
Butterfield, Joyce	(617) 555-2232	Samuels, Peter	(617) 555-8342
Chao, Shih Lin	(617) 555-5921	** Savarino, Pamela	(310) 555-1857
Charlton, Doug	(508) 555-9246	Scott, David	(617) 555-3246
** Chin, Philip	(404) 555-2341	Shea, Mary Anne	(617) 555-4616
** Clark, Alison	(510) 555-9437	** Sheffield, John	(713) 555-3877
Cobb, Matthew	(617) 555-3840	Shishov, Natasha	(617) 555-2755
Coe, Kristen	(617) 555-9192	Singer, Samuel	(508) 555-3255
Coleman, James	(508) 555-4735	Siperstein, Linda	(617) 555-6588
Crow, John	(617) 555-3332	** Sisson, Thomas	(713) 555-8390
Crowley, Charles	(617) 555-9425	** Smith, Susan	(713) 555-6613
Davidson, Jo Ann	(617) 555-3870	Soo, Hing	(617) 555-8748
Diaz, Emilio	(617) 555-3567	Sterling, Paul	(508) 555-0295

FOR INFO This DataWindow object uses newspaper columns. To understand how to create this DataWindow object without highlighting data, see "Printing with newspaper-style columns" on page 501.

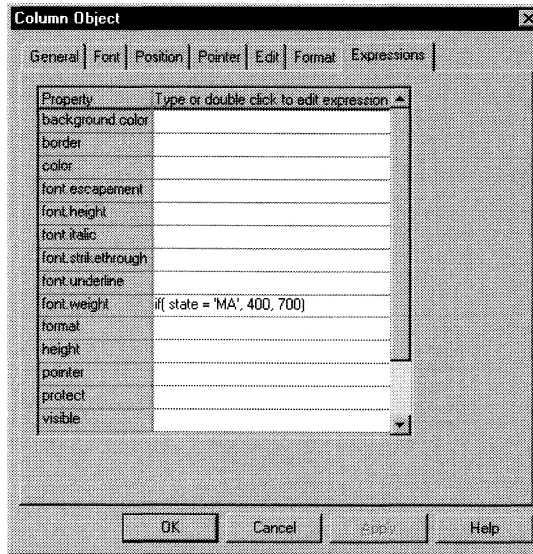
In the workspace, the detail band looks like this. It includes four objects: the employee last name, a comma, the employee first name, and the phone number:

Header 1
emp_lname emp_fname phone
Detail 1

Making the objects display in bold

To make these objects display in bold if the employee is not from Massachusetts, you would use the Expressions property page. To do this, select Properties from the popup menu of each object. For each object, enter the following expression next to the Font.Weight property:

```
If(state = 'MA', 400, 700)
```



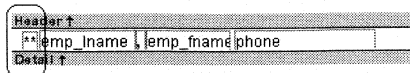
The expression states: If the value of the state column is MA (if this is true), use 400 as the font weight. This means employees from Massachusetts will display in the normal font. For false (any state but MA), use 700 as the font weight. This means all other employees display in bold font.

Logic that relies on the state column

To use logic that relies on the state column, you need to include it in the data source. You can add the column after creating the DataWindow object by modifying the data source. Also notice that the state column does not actually appear anywhere in the DataWindow object. Values must be available but do not need to be included in the DataWindow object.

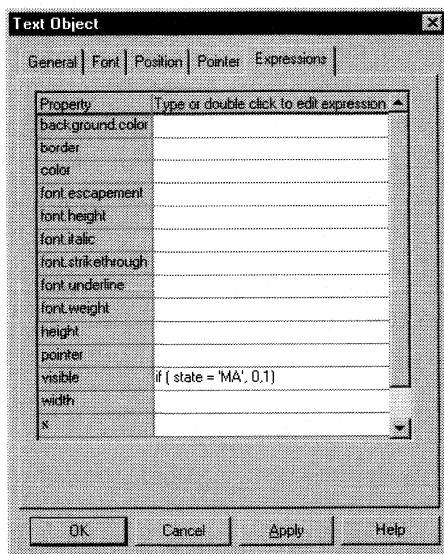
Adding the two asterisks (**)

To insert two asterisks (**) in front of the employee name if the employee is not from Massachusetts, you would first add a text object with the two asterisks in bold, exactly as you want them to display:



Then you use logic on the Visible property. To do this, display the popup menu and select Properties. On the Expressions property page, enter the following expression next to the Visible property:

```
If(state = 'MA', 0, 1)
```



This expression says: If the state of the employee is MA (the true condition), the Visible property of the ** object is off (indicated by 0). If the state of the employee is not MA (the false condition), the Visible property of the ** object is on (indicated by 1). The asterisks will be visible next to that employee's name.

Tip

You can use underlines, italics, strikethrough, borders, and colors to highlight your information. Experiment to get the effects you like.

Example 4: changing the size and location of objects

The following DataWindow object shows a rectangle and a line that change size and location if the current row contains data for a customer from the state of New York. The City and State columns also change location, but the discussion will focus on the rectangle and the line. The process for the columns is analogous:

Customers from New York				
Customer ID	Name	Address	City	State
101	Michaels Devlin	3114 Pioneer Avenue	Rutherford	NJ
102	Beth Reiser	1033 Whippany Road		New York
103	Erin Niedringhaus	1990 Windsor Street	Paoli	PA
104	Meghan Mason	550 Dundas Street East	Knowlton	TN
105	Laura McCarthy	1210 Highway 36	Carmel	IN

In the workspace, the rectangle and line display in one location (and with a single set of dimensions):

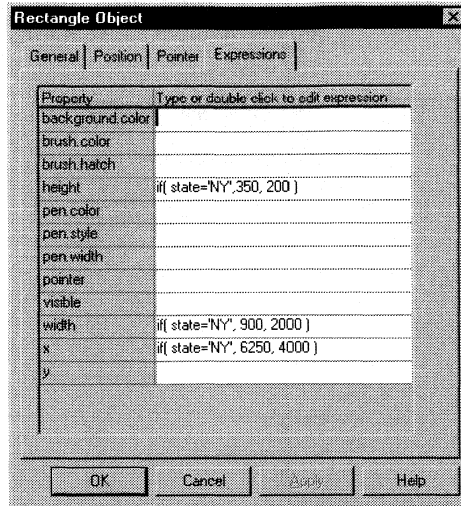
Customers from New York				
Customer ID	Name	Address	City	State
101	Michaels Devlin	3114 Pioneer Avenue	Rutherford	NJ
102	Beth Reiser	1033 Whippany Road		New York
103	Erin Niedringhaus	1990 Windsor Street	Paoli	PA
104	Meghan Mason	550 Dundas Street East	Knowlton	TN
105	Laura McCarthy	1210 Highway 36	Carmel	IN

To change properties of the rectangle and the line for rows with the state column equal to New York, you would specify the following If statement for every size or location property you wanted to change. You would substitute the values appropriate to the property. All of the values used in this example are in thousandths of inches, the unit of measure used for the DataWindow object:

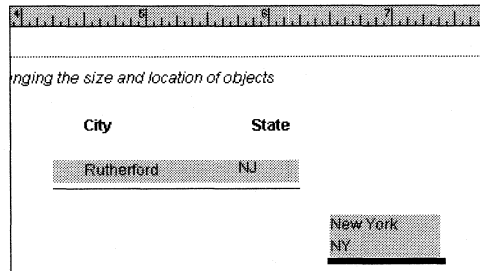
If (state='NY', true value, false value)

Changing properties of the rectangle

For the rectangle, the Expressions property page has the entries shown for the width and height of the rectangle and for its x coordinate:

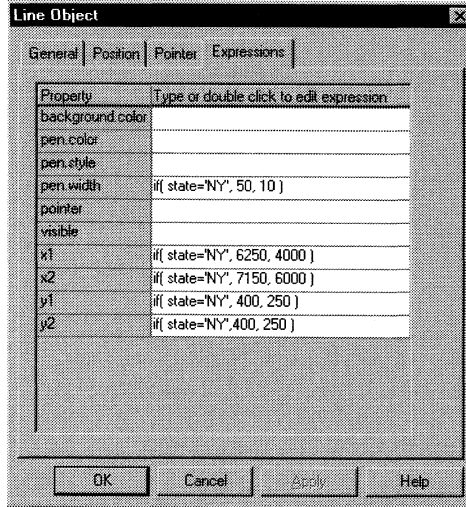


At execution time, the rectangle is one size and x location if the state is NY, and another size and location if the state is not NY. You may notice that the x values seem to be off by .25. This is because the left margin is set to .25, which adds .25 to each x value:



Changing properties of the line

For the line, the Expressions property page has entries for two sets of x1 and x2 coordinates of the line, which define two different line lengths and x positions. It also has y1 and y2 coordinates, which define the two different y positions of the line. Finally, it has an entry for pen width, which defines two different pen widths:



At execution time, the line changes length, x and y position, and pen width depending on whether the state for the current row of information is NY.

Supplying property values

Each property has its own set of property values that you can use to specify for the true and false conditions in your If expression. Usually you specify a number to indicate what you want. For example, if you are working with the Border property, you use the number 0, 1, 2, 3, 4, 5, or 6 to specify a border.

This section describes the valid property values for all of the properties you can control in the Expressions property page. These are the properties:

Property	What you can specify
Background.Color	Background color of an object
Border	Border of an object
Brush.Color	Color of a graphic object
Brush.Hatch	Pattern used to fill a graphic object
Color	Color of text for text objects, columns, and computed fields
Font.Escapement (for rotating objects)	Rotation of an object
Font.Height	Height of text
Font.Italic	Use of italic font for text
Font.Strikethrough	Use of strikethrough for text
Font.Underline	Use of underlining for text
Font.Weight	Weight (for example, bold) of text font
Format	Display format for columns and computed fields
Height	Height of an object
Pen.Color	Color of a line or the line surrounding a graphic object
Pen.Style	Style of a line or the line surrounding a graphic object
Pen.Width	Width of a line or the line surrounding a graphic object
Pointer	Image to be used for the pointer
Visible	Whether an object is visible
Width	Width of an object
X	X position of an object

Property	What you can specify
X1, X2	X coordinates of either end of a line
Y	Y position of an object relative to the band in which it is located
Y1, Y2	Y coordinates of either end of a line

Background.Color

Description Setting for the background color of an object.

Value A number that specifies the object's background color.

FOR INFO For information on specifying colors, see "Specifying colors" on page 657.

Background color of a line The background color of a line is the color that displays between the segments of the line when the pen style is not solid.

Transparent background If Background.Mode is transparent (1), Background.Color is ignored.

Example The following statement specifies that if the current row (person) uses the day care benefit, the background color of the object should be set to light gray (15790320). If not, the background color should be set to white (16777215):

```
If (bene_day_care = 'Y', 15790320, 16777215)
```

In this example, the condition is applied to the Background.Color property in the Expressions property page for three objects: the emp_id column, the emp_fname column, and the emp_lname column.

The following is a portion of the resulting DataWindow object. Notice that the employee ID, first name, and last name have a gray background if the employee uses the day care benefit:

Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care	Salary Plus Benefits
102	Fran	Whitney	\$45,700	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$50,748
105	Matthew	Cobb	\$62,000	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$67,137
160	Robert	Breault	\$57,490	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$67,802
243	Natasha	Shishov	\$72,995	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$78,191
247	Kurt	Driscoll	\$48,024	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,284
249	Rodrigo	Guevara	\$42,998	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$48,031
266	Ram	Gowda	\$59,840	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$60,165
278	Terry	Melkisetian	\$48,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$58,763
316	Lynn	Pastor	\$74,500	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	\$84,905
445	Kim	Lull	\$87,900	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	\$93,177

Border

Description

The type of border for the object.

Value

A number that specifies the type of border. Values are:

- 0—None
- 1—Shadow box
- 2—Box
- 3—Resize
- 4—Underline
- 5—3D Lowered
- 6—3D Raised

Example

The following statement specifies that if the current row (person) has a status of L (is on leave), the status column should display with a Shadow box border:

```
If(status = 'L', 1, 0)
```

In this example, the condition is applied to the Border property of the status column.

The following is a portion of the resulting DataWindow object. Notice that the status On Leave displays with a Shadow box border:

Emp ID	Employee First Name	Employee Last Name	Street	City	State	Zip Code	Phone	Status	Soc. Sec. No.
102	Fran	Whitey	49 East Washinton Street	Needham	MA	02192-	(617) 555-3985	Active	017-34-9033
105	Matthew	Cobb	77 Pleasant Street	Waltham	MA	02154-	(617) 555-3840	Active	052-34-5739
129	Philo	Chin	59 Pond Street	Atlanta	GA	30339-	(404) 555-2341	Active	024-60-8923
148	Julie	Jordan	144 Great Plain Avenue	Winchester	MA	01890-	(617) 555-7835	Active	501-70-4733
160	Robert	Broult	58 Cherry Street	Milton	MA	02186-	(617) 555-3099	Active	025-48-7623
184	Melissa	Essanoza	112 Apple Tree Way	Stow	MA	01775-	(508) 555-2319	Active	025-48-1943
191	Jeanette	Bertrand	209 Concord Street	Acton	MA	01720-	(508) 555-8138	Active	017-34-8821
195	Marc	Dill	89 Hancock Street	Milton	MA	02186-	(617) 555-2144	Active	079-48-6634
207	Jane	Francis	12 Hawthorne Drive	Concord	MA	01742-	(508) 555-9022	Active	501-70-8992
243	Natasha	Shishov	15 Milk Street	Waltham	MA	02154-	(617) 555-2755	Active	043-21-6799
247	Kurt	Driscoll	154 School Street	Waltham	MA	02154-	(617) 555-1234	On Leave	024-60-1768
249	Rodolfo	Guevara	East Main Street	Frammingham	MA	01701-	(508) 555-0029	Active	084-32-9990
266	Ram	Gowda	79 Peace Street	Natick	MA	01760-	(508) 555-8722	Active	017-34-6122

About the value L and the value On Leave

The status column uses an edit style. The internal value for on leave is L and the display value is On Leave. The conditional expression references the internal value L, which is the actual value stored in the database. The DataWindow object shows the value On Leave, which is the display value assigned to the value L in the code table for the Status edit style.

Brush.Color

Description

Setting for the fill color of a graphic object.

Value

A number that specifies the color that fills the object.

FOR INFO For information on specifying colors, see "Specifying colors" on page 657.

Example

See the example for Brush.Hatch, next.

Brush.Hatch

Description

Setting for the fill pattern of a graphic object.

Value

A number that specifies the pattern that fills the object. Values are:

0—Horizontal

1—Bdiagonal (lines from lower left to upper right)

- 2—Vertical
- 3—Cross
- 4—Fdiagonal (lines from upper left to lower right)
- 5—DiagCross
- 6—Solid
- 7—Transparent

Example

In this example, statements check the employee's start date to see if the month is the current month or the month following the current month. Properties of a rectangle object placed behind the row of data are changed to highlight employees with months of hire that match the current month or the month following the current month.

The workspace 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:

today()		Performance Review Reminder		
Department ID	Employee ID	Employee First Name	Employee Last Name	Start Date
Header ↑				
dept_id	emp_id	emp_fname	emp_lname	start_date
Detail ↑				

The following statement is for the Brush.Color property of the rectangle. If the month of the start date matches the current month or the next one, Brush.Color is set to light gray (12632256). If not, it's set to white (16777215), which means it will not show:

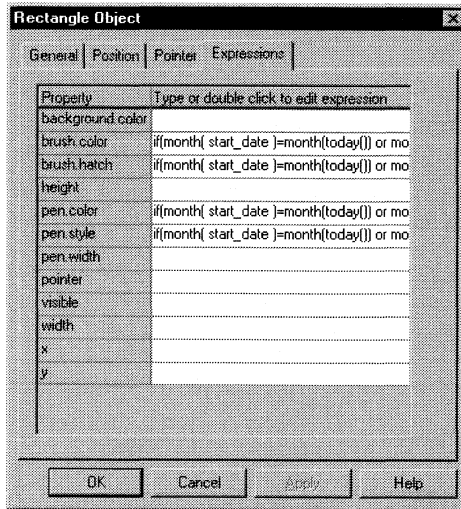
```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
12632256, 16777215)
```

The following statement is for the Brush.Hatch property of the rectangle. If the month of the start date matches the current month or the next one, Brush.Hatch is set to Bdiagonal (1). If not, it's set to Transparent (7), which means it will not show:

```
If(month( start_date ) = month(today())
or month( start_date ) = month(today())+1
or (month(today()) = 12 and month(start_date)=1),
1, 7)
```

The Expressions property page looks like this. Notice that it also includes expressions for Pen.Color and Pen.Style.

FOR INFO For more about these properties, see "Pen.Style" on page 650.



The following is a portion of the resulting DataWindow object. A rectangle with light gray cross-hatching highlights employees whose reviews are due soon:

01/11/98		Performance Review Reminder			
Department ID	Employee ID	Employee First Name	Employee Last Name	Start Date	
400	888	Doug	Charlton	03/10/1991	
200	902	Moira	Kelly	04/01/1991	
200	913	Ken	Martel	04/16/1991	
500	921	Charles	Crowley	04/22/1991	
200	930	Ann	Taylor	05/08/1991	
200	949	Pamela	Savarino	05/08/1991	
100	958	Thomas	Sisson	07/16/1991	
400	992	Joyce	Butterfield	08/13/1991	
500	1013	Joseph	Barker	09/10/1991	
200	1021	Paul	Sterling	10/28/1991	
200	1039	Shih Lin	Chao	11/11/1991	
400	1062	Barbara	Blaikie	11/20/1991	
100	1090	Susan	Srnith	12/13/1991	
200	1101	Mark	Preston	01/09/1992	Review due this month
200	1142	Allison	Clark	01/19/1992	Review due this month
100	1157	Hing	Soo	01/29/1992	Review due this month
200	1182	Kevin	Goggin	02/03/1992	Review due next month
400	1191	Matthew	Bucceri	02/12/1992	Review due next month

Color

Description The text color of the column, computed field, or text object.

Value A number that specifies the color used for text.

FOR INFO For information on specifying colors, see "Specifying colors" on page 657.

Example The following statement is for the Color property of the emp_id, emp_fname, emp_lname, and emp_birth_date columns.

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):

```
If(month(birth_date) = month(today()), 255, 0)
```

The Expressions property page also includes the same conditional expression for the Font.Underline property so that the example shows clearly on paper.

The following shows a portion of the DataWindow object. Employees with birthdays in the current month display underlined and in red:

Employee ID	Employee First Name	Employee Last Name	Birth Date
102	Fran	Whitney	06/05/1958
105	Matthew	Cobb	12/04/1960
129	Philip	Chin	10/30/1966
148	Julie	Jordan	12/13/1951
160	Robert	Breault	05/13/1947
184	Melissa	Espinoza	12/14/1939
191	Jeannette	Bertrand	12/21/1964
195	Marc	Dill	07/19/1963
<u>207</u>	<u>Jane</u>	<u>Francis</u>	<u>09/12/1954</u>
243	Natasha	Shishov	04/22/1949
247	Kurt	Driscoll	03/05/1955
249	Rodrigo	Guevara	11/23/1956

Font.Escapement (for rotating objects)

Description The angle of rotation from the baseline of the text.

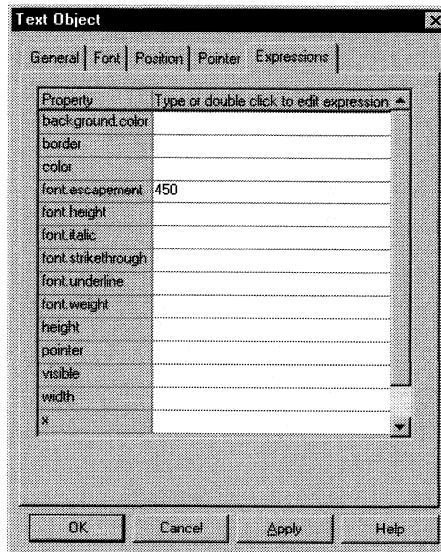
Value An integer in tenths of degrees. For example, 450 means 45 degrees. 0 is horizontal.

On Macintosh rotation is limited

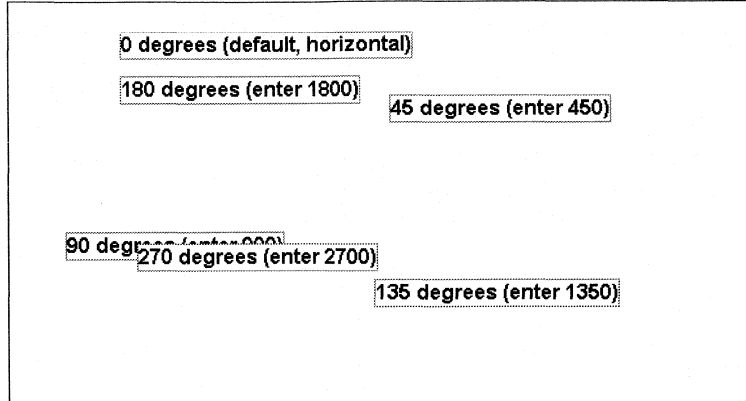
On the Macintosh, you can specify 90 degrees of rotation only. Other values are ignored.

Example

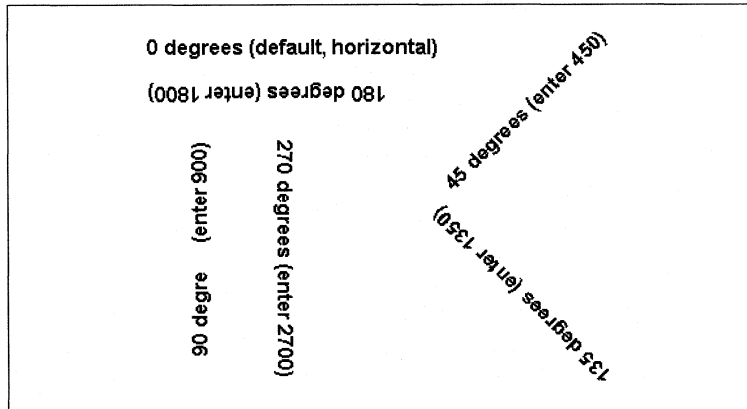
The following Expressions property page shows how you enter rotation for an object. You do not need a conditional expression. Usually you just enter the number of tenths of degrees you want to rotate an object next to the Font.Escapement property:



The following picture shows the workspace with a number of text objects. Each text object shows the Font.Escapement value entered and the number of degrees of rotation. In the workspace, you do not see rotation; it looks as if the objects are all mixed up. Two objects seem to overlay each other:



The next picture shows the same objects at execution time. Each object is rotated appropriately:



How to position objects that are rotated

In the workspace, make the objects movable (display each object's property sheet and select the Moveable checkbox in the Position property page). Then in preview, drag the rotated objects where you want them. To drag a rotated object, you position the pointer in the center of the object and drag from there. When you go back to the workspace, the objects will be wherever you dragged them. They may look incorrectly positioned in the workspace, but they will be correctly positioned when you run the DataWindow object. When you are satisfied with the positioning, you may want to clear the Moveable checkbox for the objects (to ensure that they stay where you want them).

Font.Height

Description

The height of the text.

Value

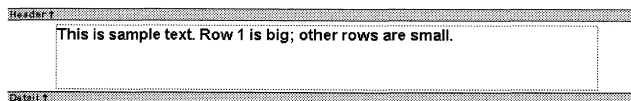
An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels. To specify size in points, specify a negative number.

Example

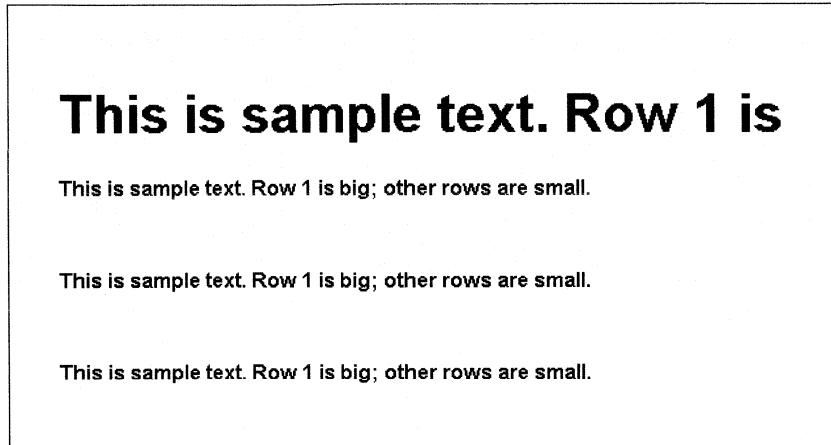
The following statement is included on the Expressions property page for the Font.Height property of a text object. The statement says if it's the first row, show the text 1/2-inch high (500 1/1000ths of an inch) and if it's not the first, show the text 1/5-inch high (200 1/1000ths of an inch):

```
If(GetRow() = 1, 500, 200)
```

The following picture shows the text object in the workspace. Notice that the boundaries of the object had to be extended to allow for the increased size of the text:



This is what the DataWindow object looks like at execution time. The first occurrence of the text object is big (1/2 inch); subsequent ones are small (1/5 inch). Note that the DataWindow object is defined as using thousandths of an inch as its unit of measure:



Font.Italic

Description

A number that specifies whether the text should be italic.

Value

Values are:

0—Not italic

1—Italic

Example

The following statements are included on the Expressions property page for the Font.Italic, Font.Underline, and Font.Weight properties, respectively. If the employee has health insurance, the employee's information displays in italics. If not, the employee's information displays in bold and underlined:

```
If(bene_health_ins = 'Y', 1, 0)
If(bene_health_ins = 'N', 1, 0)
If(bene_health_ins = 'N', 700, 400)
```

The Expressions property page is filled out in this way for four objects in the workspace: the emp_id column, the emp_fname column, the emp_lname column, and the emp_salary column.

The resulting DataWindow object looks like this. Those with health insurance display in italics. Those without health insurance are emphasized with bold and underlining:

				Health insurance		
129	<i>Philip</i>	<i>Chin</i>	\$38,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
195	<i>Marc</i>	<i>Dill</i>	\$54,800.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
299	<i>Rollin</i>	<i>Overbey</i>	\$39,300.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
467	<i>James</i>	<i>Klobucher</i>	\$49,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
641	<i>Thomas</i>	<i>Powell</i>	\$54,600.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<u>667</u>	<u>Mary</u>	<u>Garcia</u>	<u>\$39,800.00</u>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<u>690</u>	<u>Kathleen</u>	<u>Poitras</u>	<u>\$46,200.00</u>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
856	<i>Samuel</i>	<i>Singer</i>	\$34,892.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
902	<i>Moira</i>	<i>Kelly</i>	\$87,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<u>913</u>	<u>Ken</u>	<u>Martel</u>	<u>\$55,700.00</u>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
930	<i>Ann</i>	<i>Taylor</i>	\$46,890.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Font.Strikethrough

Description A number that specifies whether the text should be crossed out.

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 object:

```
If(status = 'L', 1, 0)
```

The workspace includes four objects (emp_id, emp_fname, emp_lname, and emp_salary columns) for which the Expressions property page includes the entry for the Strikethrough property. An extra text object 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 653).

The following is a portion of the resulting DataWindow object. It shows two employees who are On Leave. The four columns of information show as crossed out:

102	Fran	Whitney	\$45,700.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
105	Matthew	Cobb	\$62,000.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
160	Robert	Breault	\$57,490.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
243	Natasha	Shishov	\$72,995.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
247	Kurt	Driecoll	\$48,023.69	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<i>On leave</i>
249	Rodrigo	Guevara	\$42,998.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
266	Ram	Gowda	\$59,840.00	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
278	Terry	Melkisetian	\$48,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
316	Lynn	Pastor	\$74,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
445	Kim	Lull	\$87,900.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
453	Andrew	Rabkin	\$64,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
479	Linda	Siperstein	\$39,875.50	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<i>On leave</i>

Font.Underline

Description

A number that specifies whether the text should be underlined.

Value

Values are:

- 0—Not underlined
- 1—Underlined

Example

The following statement, when applied to the Font.Underline property of columns of employee information, causes the information to be underlined if the employee does not have health insurance:

```
If(bene_health_ins = 'N', 1, 0)
```

FOR INFO For pictures of this example, see "Font.Italic" on page 645.

Font.Weight

Description

The weight of the text.

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 INFO For pictures of this example, see "Font.Italic" on page 645.

Format

Description

The display format for a column.

Values

A string specifying the display format.

Example

The following statement, when applied to the Format property of the Salary column, causes the column to display the word *Overpaid* for any salary greater than \$60,000 and *Underpaid* for any salary under \$60,000:

```
If(salary>60000, 'Overpaid', 'Underpaid')
```

Edit Mask edit style change

The Edit Mask edit style assigned to the salary column had to be changed. Because edit styles take precedence over display formats, it was necessary to change the edit style assigned to the salary column (an Edit Mask edit style) to the Edit edit style.

The following DataWindow object shows the values in the salary column to be either the word *Underpaid* or *Overpaid*:

Employee Report 01/11/98							
Department ID	Employee ID	Employee First Name	Employee Last Name	Salary	Health Ins.	Life Ins.	Day Care
100							
	102	Fran	Whitney	Underpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	105	Matthew	Cobb	Overpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	160	Robert	Breault	Underpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	243	Natasha	Shishov	Overpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	247	Kurt	Driscoll	Underpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	249	Rodrigo	Guevara	Underpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	266	Ram	Gowda	Underpaid	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	278	Terry	Melkisetian	Underpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	316	Lynn	Pastor	Overpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	445	Kim	Lull	Overpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	453	Andrew	Rabkin	Overpaid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Height

Description

The height of the column or other object.

Value

An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example

The following statement causes the height of a rectangle to be .350 inches if the state column for the row has the value NY. Otherwise, the rectangle is .200 inches high:

```
If(state = 'NY',350, 200)
```

FOR INFO For more details and pictures, see "Example 4: changing the size and location of objects" on page 632.

Pen.Color

Description

The color of the line or the outline of a graphic object.

Value

A number that specifies the color of the line or outline.

FOR INFO For information on specifying colors, see "Specifying colors" on page 657.

Example See the example for the Pen.Style property, next.

Pen.Style

Description The style of the line or the outline of a graphic object.

Value Values are:

- 0—Solid
- 1—Dash
- 2—Dotted
- 3—Dash-dot pattern
- 4—Dash-dot-dot pattern
- 5—Null (no visible line)

Example In this example, statements check the employee's start date to see if the month is the current month or the month following the current month. Properties of a rectangle object placed behind the row of data are changed to highlight employees with months of hire that match the current month or the month following the current month.

The workspace 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:

today()		Performance Review Reminder		
Department ID	Employee ID	Employee First Name	Employee Last Name	Start Date
Header 1				
dept_id	emp_id	emp_fname	emp_lname	start_date
Detail 1				

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's 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's 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)
```

The Expressions property page also includes expressions for Brush.Color and Brush.Hatch.

FOR INFO For more about these properties, see "Brush.Hatch" on page 638.

The following is a portion of the resulting DataWindow object. A rectangle with light gray cross-hatching highlights employees whose reviews are due soon. The line enclosing the rectangle is Light Gray and uses the pen style Solid (0):

01/11/98		Performance Review Reminder			
Department ID	Employee ID	Employee First Name	Employee Last Name	Start Date	
400	888	Doug	Charlton	03/10/1991	
200	902	Maira	Kelly	04/01/1991	
200	913	Ken	Martel	04/16/1991	
500	921	Charles	Crowley	04/22/1991	
200	930	Ann	Taylor	05/08/1991	
200	949	Pamela	Savarino	05/08/1991	
100	958	Thomas	Sisson	07/16/1991	
400	992	Joyce	Butterfield	08/13/1991	
500	1013	Joseph	Barker	09/10/1991	
200	1021	Paul	Sterling	10/20/1991	
200	1039	Shih Lin	Chao	11/11/1991	
400	1062	Barbara	Blaikie	11/20/1991	
100	1090	Susan	Smith	12/13/1991	
200	1101	Mark	Preston	01/09/1992	Review due this month
200	1142	Alison	Clark	01/19/1992	Review due this month
100	1157	Hling	Soo	01/29/1992	Review due this month
200	1162	Kevin	Goggin	02/03/1992	Review due next month
400	1191	Matthew	Bucceri	02/12/1992	Review due next month

Pen.Width

Description

The width of the line or the outline of a graphic object.

Value

An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example The following statement causes the width of a line to be .05 inches if the state column for the row has the value NY. Otherwise, the line is .01 inches wide:

```
If(state = 'NY', 50, 10)
```

FOR INFO For more details and pictures, see "Example 4: changing the size and location of objects" on page 632.

Pointer

Description The image used for the mouse pointer when the pointer is over the specified object.

Value A string that specifies a value of the Pointer enumerated data type or the name of a cursor file (CUR) used for the pointer.

Values of the Pointer enumerated data type are:

```
Arrow!  
Cross!  
HourGlass!  
IBeam!  
Icon!  
Size!  
SizeNESW!  
SizeNS!  
SizeNWSE!  
SizeWE!  
UpArrow!
```

Example The following condition, entered for the Pointer property of every object in a row of expense data, changes the pointer to the PowerBuilder column every time the value in the expense column exceeds \$100,000. Note that the pointer has no meaning in a printed report. The pointer is for use on the screen display of a DataWindow object:

```
If(expense > 100000, 'pbcolumn.cur', 'arrow!')
```


Visible

Description Whether the object is visible in the DataWindow object.

Value Values are:

0—Not visible

1—Visible

Example The following statement is for the Visible property of a text object 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 object is visible. Otherwise, it's 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 workspace includes the text object at the right-hand end of the detail line. The text object is visible at execution time only if the value of the status column for the row is L.

The following is a portion of the resulting DataWindow object. It shows two employees who are On Leave. The text object is visible only for the two employees on leave:

102	Fran	Whilney	\$45,700.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
105	Matthew	Cobb	\$62,000.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
160	Robert	Breault	\$57,490.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
243	Natasha	Shishov	\$72,995.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
247	Kurt	Driscoll	\$48,023.68	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	On leave
249	Rodrigo	Guevara	\$42,998.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
266	Ram	Gowda	\$59,840.00	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
278	Terry	Melkisetian	\$48,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
316	Lynn	Pastor	\$74,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
445	Kim	Lull	\$87,900.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
453	Andrew	Rabkin	\$64,500.00	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
479	Linda	Siperstein	\$38,875.50	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	On leave

For the employees on leave, four columns of information show as crossed out. This is specified with property conditional expressions for those columns (described in the section on "Font.Strikethrough" on page 646).

Width

Description	The width of the object.
Value	An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.
Example	<p>The following statement causes the width of a rectangle to be .900 inches if the state column for the row has the value NY. Otherwise, the rectangle is 2.000 inches wide:</p> <pre>If(state = 'NY', 900, 2000)</pre> <p>FOR INFO For more details and pictures, see "Example 4: changing the size and location of objects" on page 632.</p>

X

Description	The distance of the object from the left edge of the DataWindow object. At execution time, the distance from the left edge of the DataWindow object is calculated by adding the margin to the x value.
Value	An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.
Example	<p>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:</p> <pre>If(state = 'NY', 6250, 4000)</pre> <p>FOR INFO For more details and pictures, see "Example 4: changing the size and location of objects" on page 632.</p>

X1, X2

Description	The distance of each end of the line from the left edge of the DataWindow object as measured in the workspace. At execution time, the distance from the left edge of the DataWindow object is calculated by adding the margin to the x1 and x2 values.
-------------	--

Value Integers in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example The following statements for the X1 and X2 properties of a line cause the line to extend from 6.250 to 7.150 inches from the left if the state column for the row has the value NY. Otherwise, the line extends from 4.000 to 6.000 inches from the left:

```
If(state = 'NY', 6250, 4000)
If(state = 'NY', 7150, 6000)
```

FOR INFO For more details and pictures, see "Example 4: changing the size and location of objects" on page 632.

Y

Description The distance of the object from the top of the band in which the object is located.

Value An integer in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example **FOR INFO** For information, see "Example 4: changing the size and location of objects" on page 632.

Y1, Y2

Description The distance of each end of the specified line from the top of the band in which the line is located.

Value Integers in the unit of measure specified for the DataWindow object. Units of measure include PowerBuilder units, thousandths of an inch (1000 = 1 inch), thousandths of a centimeter (1000 = 1 centimeter), or pixels.

Example The following statements for the Y1 and Y2 properties of a line cause the line to be located .400 inches (Y1 and Y2 equal .400 inches) from the top of the detail band, if the state column for the row has the value NY. Otherwise, the line is located .250 inches (Y1 and Y2 equal .250 inches) from the top of the detail band:

```
If(state = 'NY', 400, 250)  
If(state = 'NY', 400, 250)
```

FOR INFO For more details and pictures, see "Example 4: changing the size and location of objects" on page 632.

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 the table below.

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 (*r, g, b*) 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 also specifies the brightest cyan:

RGB (*r, g, b*)

RGB (0,255,255)

16776960 (The RGB function returns this number)

Notice that the expression specifies the maximum for green and blue (255) and 0 for red. 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

This table shows the numbers and expressions to enter for some common colors. The number and expression for a color are equivalent. You can use either:

Color	Expression to enter	Number to enter	How the number is calculated
Black	RGB (0, 0, 0)	0	0 (no color)

Color	Expression to enter	Number to enter	How the number is calculated
Blue	RGB (0, 0, 255)	16711680	$256*256*255$ (blue only)
Cyan	RGB (0, 255, 255)	16776960	$256*255 + 256*256*255$ (green and blue)
Dark Green	RGB (0, 128, 0)	32768	$256*128$ (green only)
Green	RGB (0, 255, 0)	65280	$256*255$ (green only)
Light Gray	RGB (192, 192, 192)	12632256	$192 + 256*192 + 256*256*192$ (some red, green, and blue in equal amounts)
Lighter Gray	RGB (224, 224, 224)	14737632	$224 + 256*224 + 256*256*224$ (some red, green, and blue in equal amounts)
Lightest Gray	RGB (240, 240, 240)	15790320	$240 + 256*240 + 256*256*240$ (some red, green, and blue in equal amounts)
Magenta	RGB (255, 0, 255)	16711935	$255 + 256*256*255$ (red and blue)
Red	RGB (255, 0, 0)	255	255 (red only)
White	RGB (255, 255, 255)	16777215	$255 + 256*255 + 256*256*255$ (red, green, and blue in equal amounts at the maximum of 255)
Yellow	RGB (255, 255, 0)	65535	$255 + 256*255$ (red and green)

Using Nested Reports

About this chapter

This chapter provides information about creating reports that have other reports nested in them.

Contents

Topic	Page
About nested reports	660
Creating a report using the Composite presentation style	664
Placing a nested report in another report	667
Working with nested reports	672

About reports and
DataWindow objects

A report is the same as a nonupdatable DataWindow object. You can create reports in either the DataWindow painter or the Report painter.

This chapter shows the process of nesting reports using the Report painter, but you can do the same things in the DataWindow painter—the results are exactly the same.

FOR INFO For more about reports and DataWindow objects, see Chapter 14, "Defining DataWindow Objects".

About nested reports

A **nested report** is a report in another report.

There are two ways to create reports that have nested reports:

- ◆ Create a composite report using the Composite presentation style
- ◆ Place a nested report in another report

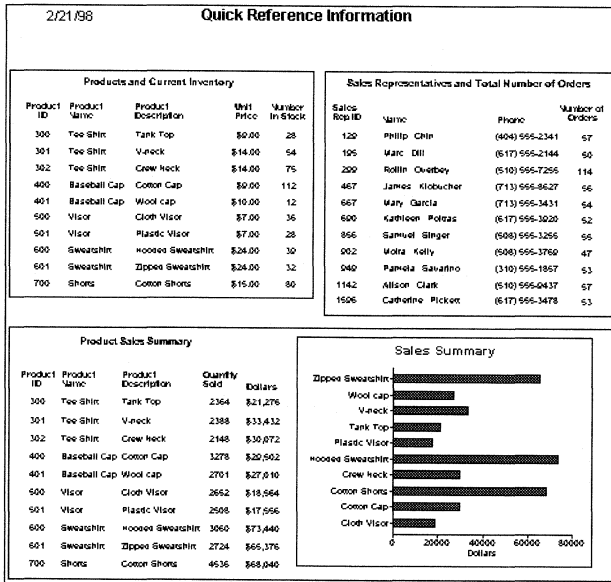
About creating a composite report

You can choose the Composite presentation style to create a new report that consists entirely of one or more nested reports. This type of report is called a **composite report**. A composite report is a container for other reports.

You can use composite reports to print more than one report (DataWindow) on a page.

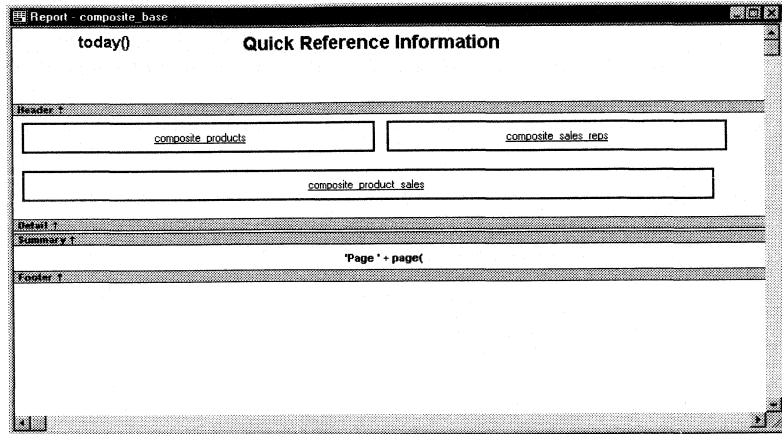
Composite report

For example, the following composite report consists of three tabular reports. One of the tabular reports includes a graph:



Composite report in the workspace

In the workspace, you see three boxes that represent the individual tabular reports that are included in the composite report. The only additional objects in this example are a title, date, and page number:



About placing a nested report within another report

You can place one or more reports within another report. The report you place is called the nested report. You can place a nested report in any type of report except crosstab. Most of the time you will place nested reports in freeform or tabular reports.

Often, the information in the nested report depends on information in the report in which it is placed (the **base report**). The nested report and the base report are related to each other by some common data. The base report and the nested report have a master/detail relationship.

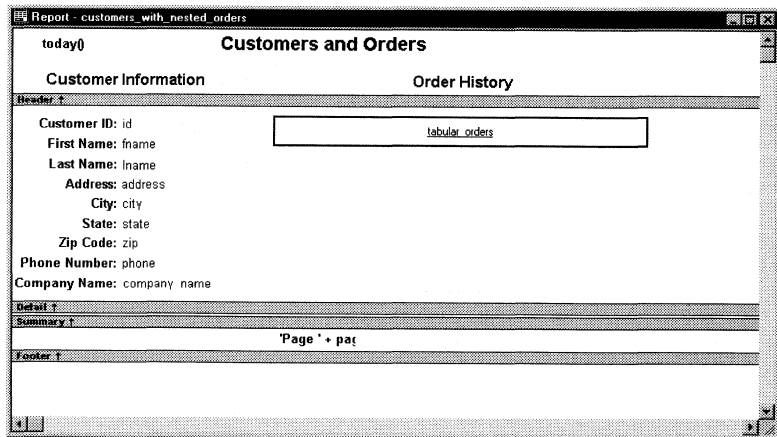
Freeform report with a related nested report

For example, the following freeform report lists all information about a customer and then includes a related nested report (it happens to be a tabular report). The related nested report lists every order that the customer has ever placed. The base report supplies the customer ID to the nested report, which requires a customer ID as a retrieval argument. This is an example of a master/detail relationship—one customer has many orders:

2/21/98		Customers and Orders					
Customer Information			Order History				
Customer ID:	105						
First Name:	Laura						
Last Name:	McCarthy						
Address:	1210 Highway 36						
City:	Carmel						
State:	IN						
Zip Code:	46032						
Phone Number:	(317) 555-8437						
Company Name:	Arno & Sons						
Sales Order ID	Order Date	Sales Rep ID	Line #	Product ID	Quantity	Date Shipped	
2006	09/28/95	299	1	300	48	09/28/95	
2344	03/30/95	195	1	501	36	03/31/95	
2454	06/16/95	299	1	501	36	06/17/95	
2568	09/21/95	896	1	600	36	09/22/95	
			2	601	36	09/22/95	

What you see in the workspace

In the workspace, you see everything in the base report plus a box that represents the related nested report:



What's the difference?

There are two important differences between nesting using the Composite style and nesting a report within a base report.

Data sources The composite report does not have a data source—it is just a container for nested reports. In contrast, a base report with a nested report in it has a data source. The nested report has its own data source.

Related nesting The composite report cannot be used to relate reports to each other in the database sense. One report cannot feed a value to another report, which is what happens in a master/detail report. If you want to relate reports to each other so that you can create a master/detail report, you need to place a nested report within a base report.

How retrieval works

When you preview (run) a composite report, PowerBuilder retrieves all the rows for one nested report, and then for another nested report, and so on until all retrieval is complete.

When you preview (run) a report with another related report nested in it, PowerBuilder retrieves all the rows in the base report first. Then PowerBuilder retrieves the data for all nested reports related to the first row. Next, PowerBuilder retrieves data for nested reports related to the second row, and so on, until all retrieval is complete for all rows in the base report.

FOR INFO For information about efficiency and retrieval, see "Supplying retrieval arguments to relate a nested report to its base report" on page 675.

Limitations on nesting reports

For the most part you can nest the various types of report styles. However, limitations apply to two of them.

Crosstabs You cannot place a crosstab with retrieval arguments within another report as a related nested report. However, you can include a crosstab in a Composite report.

RichText reports You cannot nest a RichText report in any way. You cannot place a RichText report in another report and you cannot include a RichText report in a Composite report.

Creating a report using the Composite presentation style

❖ **To create a report using the Composite presentation style:**

- 1 Click the Report button in the PowerBar.

Customizing the PowerBar

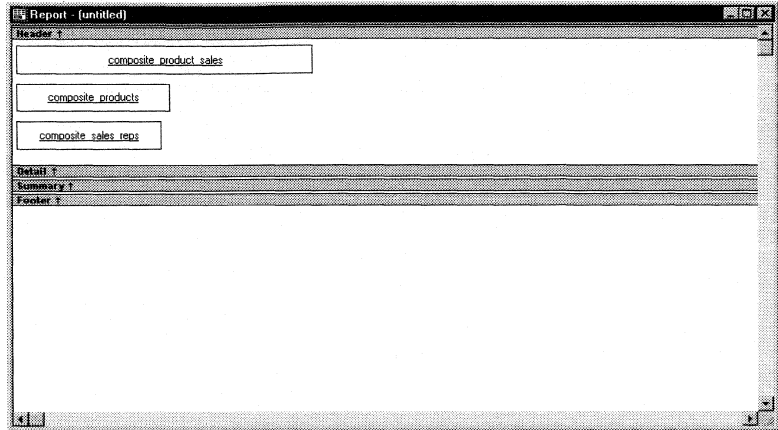
If the Report button is not in your PowerBar, you can add it.

FOR INFO For more information, see "Customizing toolbars" on page 20".

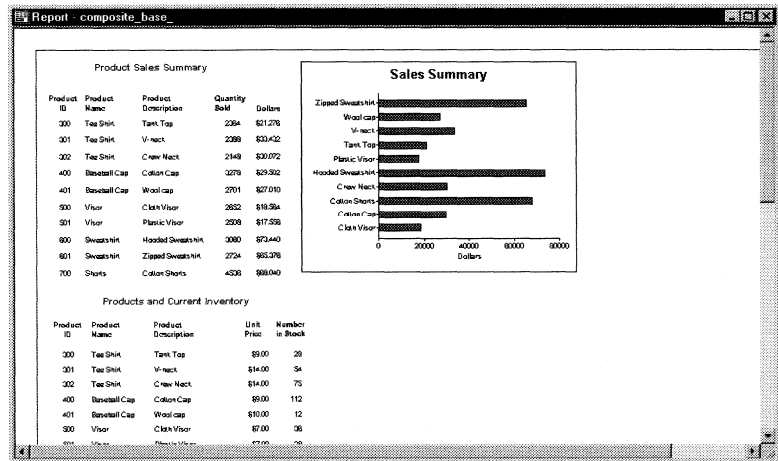
The Select Report dialog box displays listing reports (DataWindow objects) in the selected library.

- 2 Click the New button.
The New Report dialog box displays.
- 3 Click the Composite presentation style.
Notice that all the data sources become unavailable (grayed out) when you click Composite. Composite reports do not have data sources. They include other reports, which have their own data sources.
- 4 Click OK.
The Select Reports dialog box displays listing all reports (DataWindows) that are in the current application's library search path.
- 5 Click the reports you want to include in the composite report and then click OK.

- 6 PowerBuilder places boxes for the selected reports in the workspace. In this example, you see three reports:



- 7 Select File>Save from the menu bar and assign a name to the report.
8 Click the Preview button to see what your report looks like:



Notice that when you preview reports, you go to print preview (which is read-only) instead of standard preview, where you can update data in an updatable DataWindow object.

Previewing composite reports

The Rows>Filter, Rows>Import, and Rows>Sort menu items are grayed when you preview a composite report. You cannot choose these options. If you want to use any of these options, you can go back to the workspace and access the nested report(s), where these options are available in preview.

- 9 Click the Preview button to return to the workspace and continue to enhance the composite report.

Placing a nested report in another report

When you place a nested report in another report, the two reports can be independent of each other or they can be related in the database sense by sharing some common data such as a customer number or a department number. If the reports are related, you need to do some extra things to both the base report and the related nested report.

Usually when you place a report within a report rather than create a composite report, you want to relate the reports. Those instructions are first.

Placing a related nested report in another report

Typically, a related nested report provides the details for a master report. For example, a master report might provide information about customers. A related nested report placed in the master report could provide information about all the orders that belong to each customer.

❖ To place a related nested report in another report:

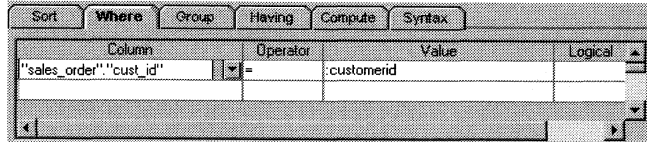
- 1 Create the nested report (DataWindow object) that you plan to place in the base report.
- 2 Define a retrieval argument for the nested report.

For example, suppose the nested report lists orders and you want to list orders for a particular customer. To define a retrieval argument you would:

- ◆ Select Design>Edit Data Source to go to the Select painter.
- ◆ Select Design>Retrieval Arguments from the menu bar in the Select painter.
- ◆ Define a retrieval argument in the Specify Retrieval Arguments dialog box. In the example, *customerID* is the name assigned to the retrieval argument.

- Specify the retrieval argument in a WHERE clause for the SELECT statement.

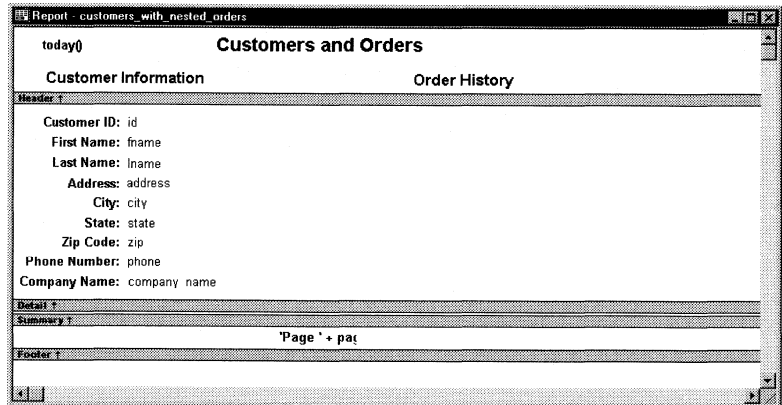
The WHERE clause in this example tells the DBMS to retrieve rows where the value in the column *cust_id* equals the value of the argument *:customerid*:



At this point, when you preview (run) the report, you are prompted to enter a value for *:customerid*. Later in these steps you will specify that the base report supply the values for *:customerid* instead of prompting for values.

- Open or create the report you want to be the base report.

In the example, the base report is one that lists customers and has a place for the order history of each customer:



- Click the Nested Report button in the Objects dropdown toolbar.

or

Select Objects>Report from the menu bar.

- Click in the workspace where you want to place the report.

The Report Object property sheet displays with the Select Report property page on top. The Select Report property page lists defined reports (DataWindow objects) in the current application's library search path.

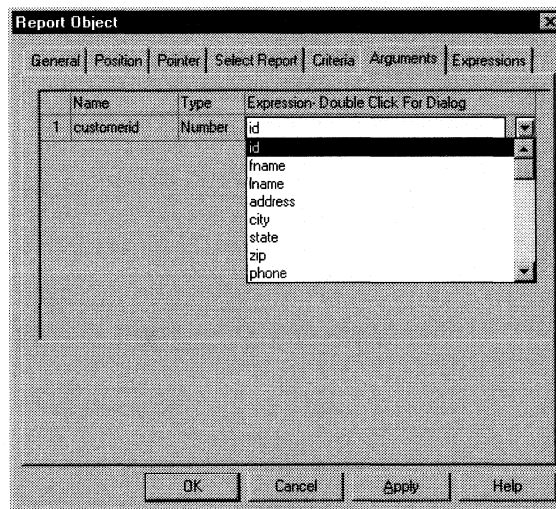
- Select the report you want and click Apply.

- 8 Select the Arguments tab of the property sheet.

The Arguments property page lists arguments defined for the nested report and provides a place for you to specify how information from the base report will be used to supply the values of arguments to the nested report.

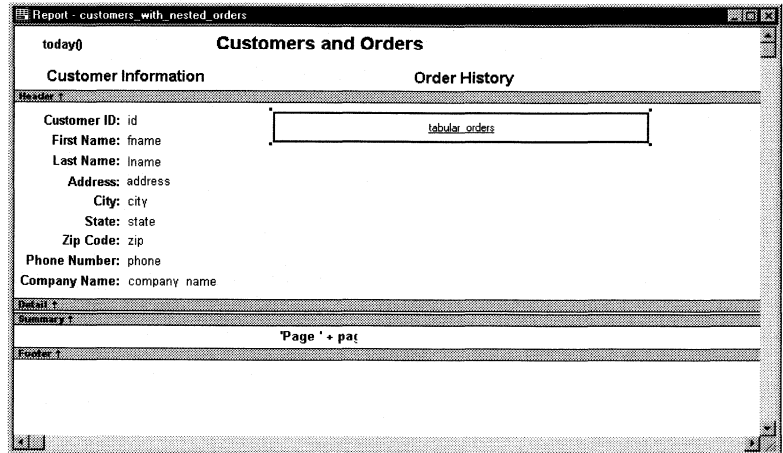
- 9 Select the base report column that will supply the argument's value from the drop down listbox and click OK.

You can also double-click to enter an expression that will be used to supply a value for the argument. In the example, the column named *id* from the base report will supply the value for the argument *:customerid* in the nested report:

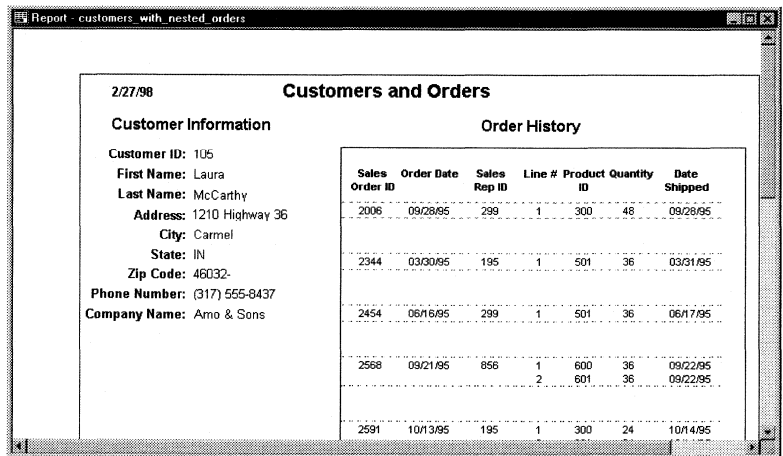


- 10 Click OK.

A box representing the report displays in the workspace, as shown in this example:



- 11 Select File>Save from the menu bar and assign a name to the report.
- 12 Click the Preview button to see what your report looks like:



FOR INFO For more information about what you can do while previewing a report, see Chapter 15, "Enhancing DataWindow Objects".

- 13 Click the Preview button to return to the workspace and continue to enhance the report.

Placing an unrelated nested report in another report

When you place an unrelated nested report in a base report, the entire nested report will appear with *each* row of the base report.

❖ **To place an unrelated nested report in another report:**

- 1 Open the report you want to be the base report.
- 2 Click the Nested Report button in the Objects dropdown toolbar.
or
Select Objects>Report from the menu bar.
- 3 Click in the workspace where you want to place the report.

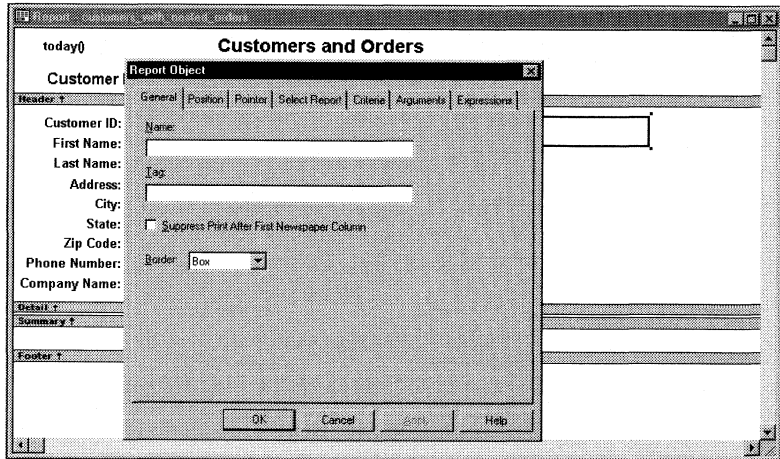
The Report Object property sheet displays with the Select Report property page on top. The Select Report property page lists defined reports (DataWindow objects) in the current application's library search path.
- 4 Select the report you want and click OK.

A box representing the report displays in the workspace.
- 5 Select File>Save from the menu bar and assign a name to the report.
- 6 Click the Preview button to see what your report looks like.

FOR INFO For more information about what you can do while previewing a report, see Chapter 15, "Enhancing DataWindow Objects".
- 7 Click the Preview button to return to the workspace and continue enhancing the report.

Working with nested reports

When you use nested reports either in composite reports or in other base reports, several enhancements and options are available. An easy way to see what you can do is to display the property sheet for the nested report. The following screen shows the property sheet for a related nested report:



Many of the options in the property sheet are described in Chapter 15, "Enhancing DataWindow Objects"—not here. For example, using borders on nested reports is like using borders on any object.

This section describes activities that apply only to nested reports or that have special meaning for nested reports. It covers:

- ◆ "Adjusting nested report width" next
- ◆ "Changing a nested report from one report to another" on page 673
- ◆ "Modifying the contents of a nested report" on page 673
- ◆ "Adding another nested report to a composite report" on page 674
- ◆ "Supplying retrieval arguments to relate a nested report to its base report" on page 675
- ◆ "Specifying criteria to relate a nested report to its base report" on page 676
- ◆ "Using options for nested reports" on page 678

Adjusting nested report width

When you preview a report with nested reports, you may find that the width of the nested report is unacceptable. This can happen, for example, if you change the design of the nested report or if you use newspaper columns in a nested report.

❖ **To adjust report width:**

- 1 In the workspace, position the pointer near a vertical edge of the nested report and press the left mouse button.
- 2 Drag the edge to widen the nested report.
- 3 Click the Preview button to check the new width.

Changing a nested report from one report to another

When you are working with nested reports, you may want to change the nested report that is used. For example, you may work on several versions of a nested report and need to update the version of the nested report that the composite or base report uses.

❖ **To change the nested report:**

- 1 Select Properties from the nested report's popup menu and then select the Select Report tab.

The Select Report property page lists all the reports in the current application's library search path.

- 2 Select the report you want to use and click OK.

The name of the report that displays in the box in the workspace changes to the new one.

Modifying the contents of a nested report

While you are working with nested reports, you may want to modify the contents of the nested report. You can do this directly from the composite report or base report that contains the nested report.

❖ **To modify the contents of a nested report from the composite report or base report:**

1 Position the pointer on the nested report whose contents you want to modify and display the popup menu.

2 Select Modify Report from the popup menu.

The nested report opens and displays in the Report painter (or DataWindow painter) workspace. Both the composite or base report and the nested report are open.

3 Modify the report.

4 Select File>Close from the menu bar.

You are prompted to save your changes.

5 Click OK.

You return to the composite report or to the base report that includes the nested report.

Adding another nested report to a composite report

After you have created a composite report, you may want to add another report.

FOR INFO For information on adding a nested report to a report (not a composite report), see "Placing a related nested report in another report" on page 667 or "Placing an unrelated nested report in another report" on page 671.

❖ **To add another nested report to a composite report:**

1 Open the composite report.

2 Click the Report button in the Objects dropdown toolbar.

or

Select Objects>Report from the menu bar.

3 Click in the workspace where you want to place the report.

The Report Object property sheet displays with the Select Report property page on top. The Select Report property page lists defined reports (DataWindow objects) in the current application's library search path.

4 Select the report you want and click OK.

A box representing the report displays in the workspace.

Supplying retrieval arguments to relate a nested report to its base report

The most efficient way to relate a nested report to its base report is to use retrieval arguments. If your nested report has arguments defined, you use the process described in this section to supply the retrieval argument value from the base report to the nested report. (The process described is part of the whole process covered in "Placing a related nested report in another report" on page 667.)

Why retrieval arguments are efficient

Some DBMSs have the ability to bind input variables in the WHERE clause of the SELECT statement. When you use retrieval arguments, a DBMS *with this capability* sets up placeholders in the WHERE clause and compiles the SELECT statement *once*. PowerBuilder retains this compiled form of the SELECT statement for use in subsequent retrieval requests.

Requirements for reusing the compiled SELECT statement

To enable PowerBuilder to retain and reuse the compiled SELECT statement:

- ◆ The database interface must support SQL caching.
- ◆ You must enable caching in the database profile. You should set the SQLCache DBParm parameter to the number of levels of nesting plus 5.

FOR INFO For more information, see the description of the SQLCache DBParm parameter in *Connecting to Your Database*.

Nested reports in composite reports

If the base report is a composite report, you need to define retrieval arguments for the composite report before you can supply them to the nested report.

Display the properties sheet for the composite report and select the Retrieval Arguments tab. Then define the retrieval arguments that the nested report needs, taking care to specify the correct type.

❖ To supply a retrieval argument value from the base report to the nested report:

- 1 Make sure that the nested report has been set up to take one or more retrieval arguments.

FOR INFO See "Placing a nested report in another report" on page 667.

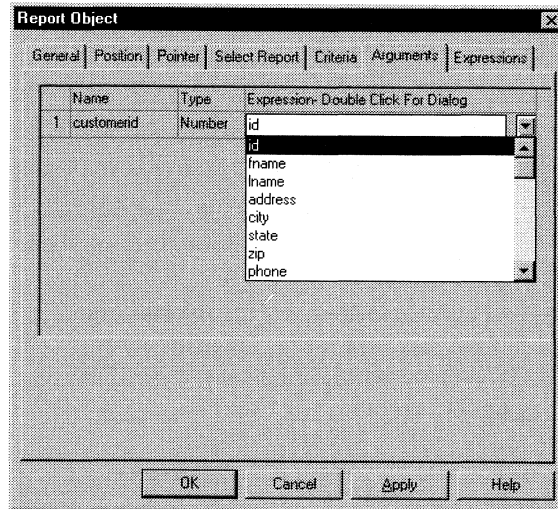
- 2 Select Properties from the nested report's popup menu and then select the Arguments tab.

The Arguments property page lists arguments defined for the nested report and provides a place for you to specify how information from the base report will supply the value of the argument to the nested report.

- 3 Select the base report column that will supply the argument's value from the dropdown listbox and click OK.

You can also click the dropdown arrow to display a list of the columns in the base report. You can double-click to enter an expression that will supply a value for the argument.

In the example, the column named *id* from the base report will supply the value for the argument *:customerid* in the nested report:



You return to the workspace. When you run the report now, you will not be prompted for retrieval argument values for the nested report. The base report will supply the retrieval argument values automatically.

Specifying criteria to relate a nested report to its base report

If you do not have arguments defined for the nested report and if database efficiency is not an issue, you can place a nested report in another report and specify criteria to pass values to the related nested report.

How the DBMS processes SQL if you use the specify criteria technique

If you use the specify criteria technique, the DBMS repeatedly recompiles the SELECT statement and then executes it. The recompilation is necessary for each possible variation of the WHERE clause.

❖ **To specify criteria to relate a nested report to its base report:**

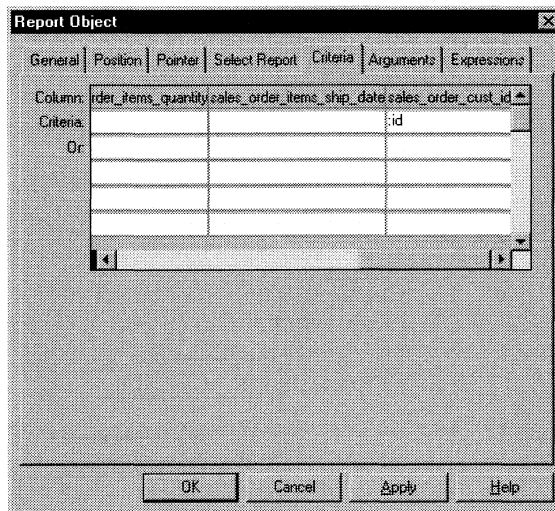
- 1 Select Properties from the nested report's popup menu and then select the Criteria tab.

The Criteria property page lists all the columns retrieved for the nested report and provides a place for you to specify how information from the base report will supply the retrieval criteria to the nested report.

- 2 Enter the retrieval criteria and click OK.

The rules for specifying criteria are the same as for specifying criteria in the Quick Select data source. Multiple criteria in one criteria line are ANDed together. Criteria entered on separate lines are ORed together.

In this example, the customer ID (the *id* column) is the retrieval criteria being supplied to the nested report. Notice that the *id* column is preceded by a colon (:), which is required:



You return to the workspace. When you run the report now, PowerBuilder retrieves rows in the nested report based on the criteria you have specified. In the example, the customer ID column in the base report determines which rows from the sales_order table are included for each customer.

Using options for nested reports

Using the Autosize Height option

Autosize Height must be on for all nested reports except graphs. This option ensures that the height of the nested report can change to accommodate the rows that are returned.

This option is on by default for all nested reports except graphs. Usually there is no reason to change it. If you do want to force a nested report to have a fixed height, you can turn this option off.

Note that the detail band also has an Autosize Height option. The option is on by default and must be on for the Autosize Height option for the nested report to work properly.

❖ To change the Autosize Height option in a nested report:

- 1 Select Properties from the nested report's popup menu and then select the Position tab.
- 2 Select/clear the Autosize Height checkbox.

If you put a nested report in a band other than the detail band

Bands other than the detail band do not have an Autosize Height option. If you place a nested report in a band other than the detail band, you must size the band itself to hold the report.

Using the Slide option

PowerBuilder determines the appropriate Slide options when positioning the nested report(s) and assigns default values. Usually you should not change the default values:

- ◆ The Slide Left option is on by default for grid and crosstab style reports and off by default for all others. Having Slide Left on for grid and crosstab ensures that these reports break horizontally on whole columns and not in the middle of a column.
- ◆ The Slide Up All Above/Up Directly Above options ensure that the nested report uses just as much vertical space as it needs. One of these options is on by default for all nested reports.

FOR INFO For more information, see "Sliding objects to remove blank space in a DataWindow object" on page 517.

Using the Start On New Page option (composite only)

The Start On New Page option forces a new page for a nested report used *in a composite report*. By default, this option is off.

❖ **To specify that a nested report in a composite report should begin on a new page:**

- 1 Select Properties from the nested report's popup menu and then select the General tab.
- 2 Select the Start On New Page checkbox.

A checkmark displays indicating the option is selected.

Using the Trail the Footer option (composite only)

The Trail the Footer option controls the placement of the footer for the last page of a nested report *in a composite report*. By default, this option is on. The footer appears directly under the contents of the nested report and not at the bottom of the page.

❖ **To specify that the footer should appear at the bottom of the page:**

- 1 Select Properties from the nested report's popup menu and then select the General tab.
- 2 Clear the Trail the Footer checkbox.

The checkmark next to the option disappears, indicating the option is no longer selected. The footer will appear at the bottom of the page on all pages of the nested report, including the last page. Note that if another nested report begins on the same page, the footer from the earlier report might be misleading or confusing.

About this chapter

This chapter describes how to build and use graphs in PowerBuilder.

Contents

Topic	Page
Overview of graphs	682
Using graphs in DataWindow objects	689
Using the Graph presentation style	707
Defining a graph's properties	708
Using graphs in windows	723

Overview of graphs

Often the best way to display information is graphically. Instead of showing users a series of rows and columns of data, you can present information as a graph in a DataWindow object or window. For example, in a sales application, you might want to present summary information in a column graph.

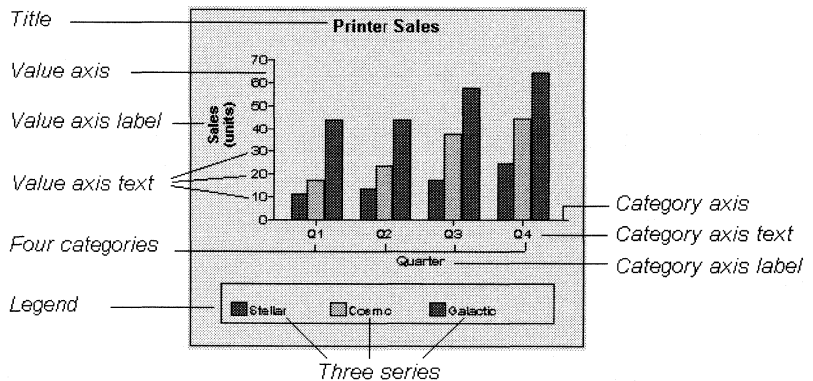
PowerBuilder provides many types of graphs and allows you to customize your graphs in many ways. Probably most of your use of graphs will be in a DataWindow object—the source of the data for your graphs will be the database. You can also use graphs as standalone controls in windows (and user objects) and populate the graphs with data through scripts.

The way you define graphs is the same whether you are using them in a DataWindow object or directly in a window. But the way you manipulate graphs is different in a DataWindow object from in a window.

Before using graphs in an application, you need to understand the parts of a graph and the kinds of graphs that PowerBuilder provides.

Parts of a graph

Here is a column graph created in PowerBuilder that contains most major parts of a graph. It shows quarterly sales of three products: Stellar, Cosmic, and Galactic printers:



How data is represented

Graphs display data points. To define graphs, you need to know how the data is represented. PowerBuilder organizes data into three components:

Component	Meaning
Series	A set of data points Each set of related data points makes up one series. In the preceding graph, there is a series for Stellar sales, another series for Cosmic sales, and another series for Galactic sales. Each series in a graph is distinguished by color, pattern, or symbol
Categories	The major divisions of the data Series data are divided into categories, which are often non-numeric. In the preceding graph, the series are divided into four categories: Q1, Q2, Q3, and Q4. Categories represent values of the independent variable(s)
Values	The values for the data points (dependent variables)

Organization of a graph

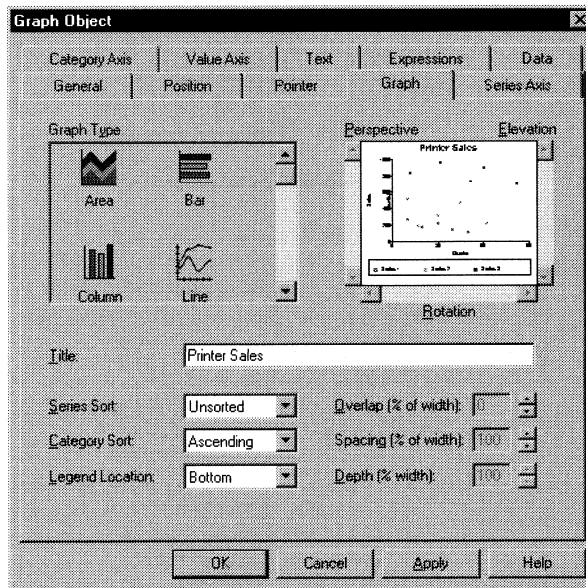
The following table lists the parts of a typical graph:

Part of graph	What it is
Title	An optional title for the graph. The title appears at the top of the graph
Value axis	The axis of the graph along which the values of the dependent variable(s) are plotted. In a column graph, as shown in the preceding graph, the Value axis corresponds to the y axis in an XY presentation. But in other types of graphs, such as a bar graph, the Value axis can be along the x dimension
Category axis	The axis along which are plotted the major divisions of the data, representing the independent variable(s). In the preceding graph, the Category axis corresponds to the x axis. It plots four categories: Q1, Q2, Q3, and Q4. These form the major divisions of data in the graph
Series	A set of data points. There are three series in the preceding graph: Stellar, Cosmic, and Galactic. In bar and column charts, each series is represented by bars or columns of one color or pattern
Series axis	The axis along which the series are plotted in three-dimensional (3D) graphs

Part of graph	What it is
Legend	An optional listing of the series. The preceding graph contains a legend that shows how each series is represented in the graph

Types of graphs

PowerBuilder provides many graph types for you to choose from. You choose the type from the Graph property page of the Graph Object property sheet:



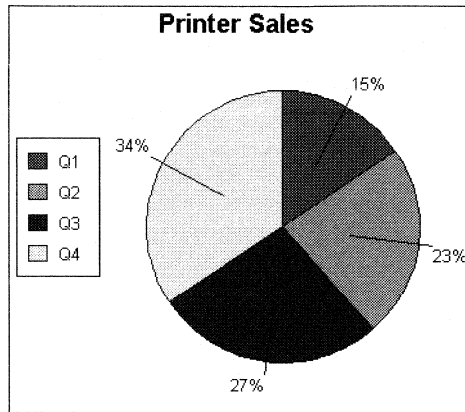
Area, bar, column, and line graphs

Area, bar, column, and line graphs are conceptually very similar. They differ only in how they physically represent the data values—whether they use areas, bars, columns, or lines to represent the values. All other properties are the same. Typically you use area and line graphs to display continuous data and use bar and column graphs to display noncontinuous data.

The only difference between a bar graph and a column graph is the orientation: in column graphs, values are plotted along the y axis and categories are plotted along the x axis. In bar graphs, values are plotted along the x axis and categories are plotted along the y axis.

Pie graphs

Pie graphs typically show one series of data points with each data point shown as a percentage of a whole. The following pie graph shows the sales for Stellar printers for each quarter. You can easily see the relative values in each quarter. (PowerBuilder automatically calculates the percentages of each slice of the pie.)



You can have pie graphs with more than one series if you want; the series are shown in concentric circles. Multiseries pie graphs can be useful in comparing series of data.

Scatter graphs

Scatter graphs show xy data points. Typically you use scatter graphs to show the relationship between two sets of numeric values.

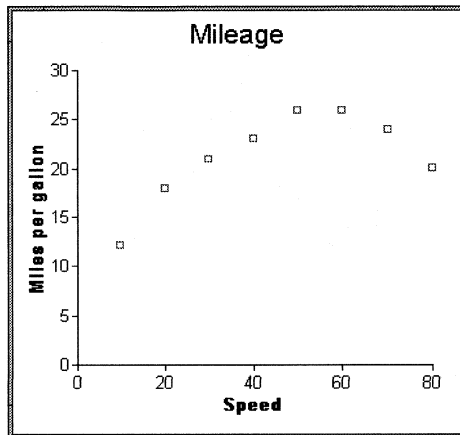
Scatter graphs do not use categories. Instead, numeric values are plotted along both axes—as opposed to other graphs, which have values along one axis and categories along the other axis.

For example, the following data shows the effect of speed on the mileage of a sedan:

Speed	Mileage
10	12
20	18
30	21
40	23

Speed	Mileage
50	26
60	26
70	24
80	20

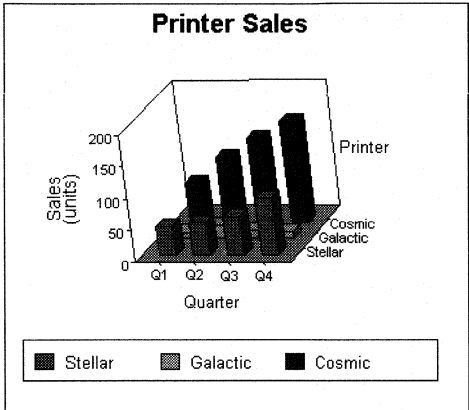
Here is the data in a scatter graph:



You can have multiple series of data in a scatter graph. In the above example, you might want to plot mileage versus speed for several makes of cars in the same graph.

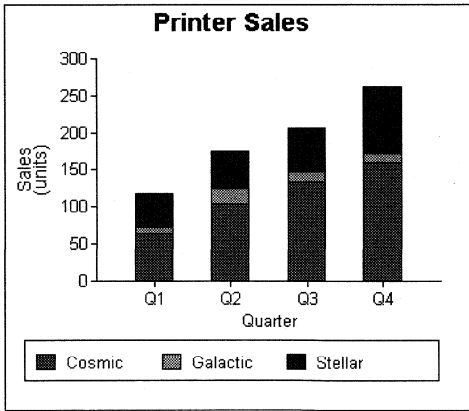
Three-dimensional graphs

You can also create 3-dimensional (3D) graphs of area, bar, column, line, and pie graphs. In 3D graphs (except for 3D pie graphs), series are plotted along a third axis (the Series axis) instead of along the Category axis. You can specify the perspective to use to show the third dimension:



Stacked graphs

In bar and column graphs, you can choose to stack the bars and columns. In stacked graphs, each category is represented as one bar or column instead of as separate bars or columns for each series:



Using graphs in applications

You can use graphs in DataWindow objects and in windows. You specify the properties of a graph, such as its type and title, the same way in a DataWindow object as in a window.

Using graphs in user objects

You can also use graphs in user objects. Everything in this chapter about using graphs in windows also applies to using graphs in user objects.

The major differences between using a graph in a DataWindow object and using a graph in a window (or user object) are:

- ◆ Specifying the data for the graph

In DataWindow objects, you associate columns in the database with the axes of a graph. In windows, you write scripts containing PowerScript functions to populate a graph.

- ◆ Specifying the location of the graph

In DataWindow objects, you can place a graph in the foreground and allow users to move and resize the graph during execution or you can place a graph in a band and prevent movement. In windows, graphs are placed like all other window controls.

Using graphs in DataWindow objects

You will probably use most of your graphs 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 an object in a DataWindow object
Here you use a graph to enhance the display of information in a DataWindow object, such as a tabular or freeform DataWindow object. This technique is described next.
- ◆ By using the Graph presentation style
Here the entire DataWindow object is a graph. The underlying data isn't visible. This technique is described in "Using the Graph presentation style" on page 707.

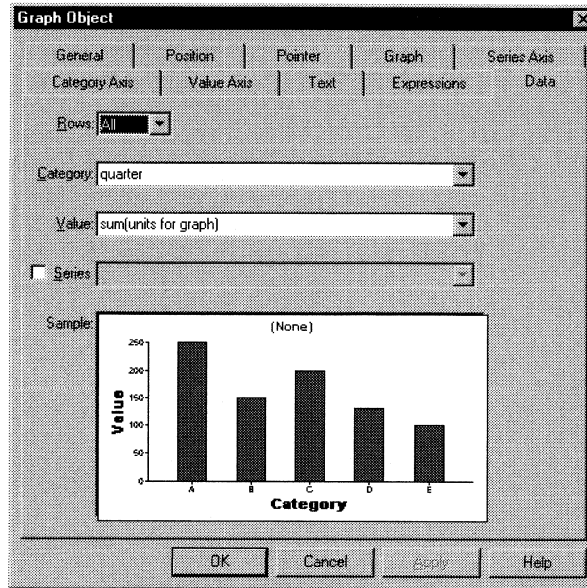
Placing a graph in a DataWindow object

❖ To place a graph in a DataWindow object:

- 1 Open the DataWindow painter and select the DataWindow object that will contain the graph.
The DataWindow painter workspace displays the DataWindow object.
- 2 Click the Graph button in the Objects dropdown toolbar.

- Click where you want the graph.

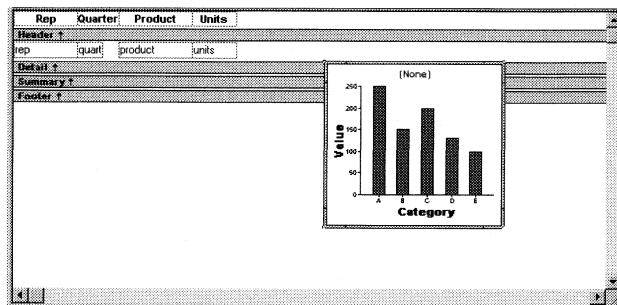
PowerBuilder displays the Graph Object property sheet with the Data property page on top:



- Specify which columns contain the data and click OK.

FOR INFO For more information, see "Associating data with a graph" on page 694.

You return to the painter workspace with a representation of the graph in place:



- Specify the graph's properties using the graph's property sheet.

Using the graph's property sheet

A graph has a property sheet in which you specify the data as well as the other properties of the graph.

❖ To display the Graph Object property sheet:

- ◆ Select Properties from the graph's popup menu.

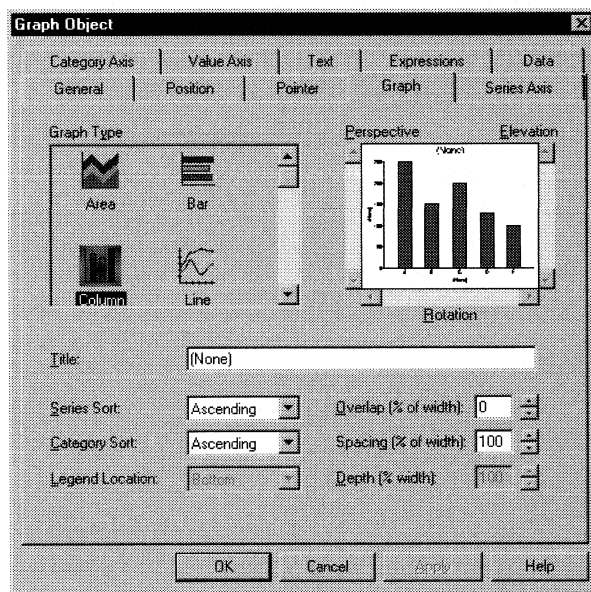
or

Select the graph and then select Design>Properties from the menu bar.

or

Select the graph and then click the Properties button.

The property sheet for a graph has 10 property pages in which you specify information about the graph. The following property sheet shows the Graph property page on top:



The following table lists the ten property pages on the Graph Object property sheet and describes what each property page specifies:

Property page	What it specifies
Category Axis	Labels, scale, information about major and minor divisions for the category axis
Data	Where to get the graph's data

Property page	What it specifies
Expressions	An expression that is evaluated at execution time to determine the value of a property of the graph
General	Various general graph properties, including border, graph colors, whether to size the graph to the full screen display, suppression in newspaper columns
Graph	Graph type, title, category and series sorting, legend location For 3D graphs, perspective, rotation, and elevation For bar graphs, overlap, spacing and depth of bars
Pointer	The pointer to use when the mouse is positioned over the graph
Position	The x,y location of the upper left corner of the graph, its width and height, sliding options, the layer in which the graph is to be positioned Whether the graph can be resized and moved during preview
Series Axis	Labels, scale, information about major and minor divisions for the series axis
Text	Text properties for text objects that display on the graph, including title, axis text, axis label, and legend Text properties include font, font style, font size, alignment, rotation, color, display expression, display format
Value Axis	Labels, scale, information about major and minor divisions for the value axis

Changing a graph's position and size

When you initially place a graph in a DataWindow object, it is in the foreground—it sits above the bands in the DataWindow object. Unless you change this setting, the graph displays in front of any retrieved data.

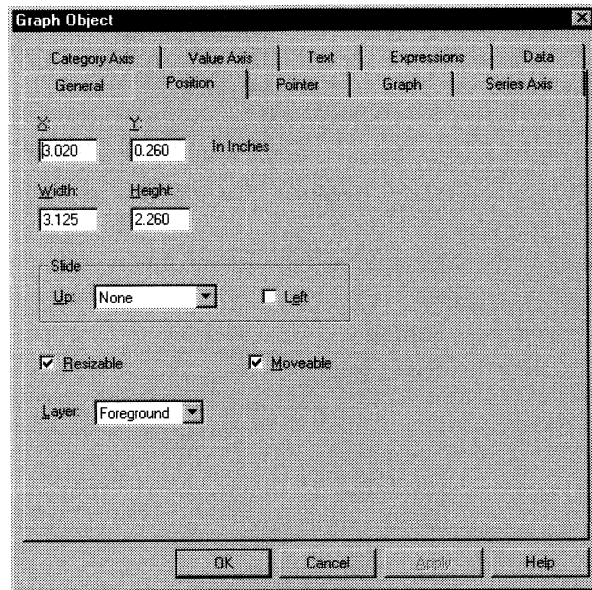
The initial graph is also movable and resizable, so users have complete flexibility as to the size and location of a graph at execution time.

You can change these properties if you want.

❖ To specify a graph's position and size:

- 1 Select Properties from the graph's popup menu and then select the Position tab or the General tab.

The Position property page contains most of the options:



- 2 Select the settings for the following options on the Position property page:

Setting	Meaning
Layer	<p><i>Background</i> — The graph displays behind other elements in the DataWindow object</p> <p><i>Band</i> — 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</p> <p><i>Foreground</i> — (Default) The graph displays above all other elements in the DataWindow object. Typically, if you choose this setting, you also make the graph moveable so it won't obscure data while users display the DataWindow object</p>
Moveable	The graph can be moved while displaying the DataWindow object

Setting	Meaning
Resizable	The graph can be resized while displaying the DataWindow object
Slide	The graph slides to the left or up to remove extra white space FOR INFO For more information, see "Sliding objects to remove blank space in a DataWindow object" on page 517.
X, Y	The location of the upper-left corner of the graph
Width, Height	The width and height of the graph

- 3 Select the settings for the following options on the General property page:

Setting	Meaning
Size to Display	The graph fills the DataWindow object and resizes when users resize the DataWindow object. This setting is used with the Graph presentation style
Suppress After First	Don't repeat graph after the first column in a DataWindow object using newspaper-style columns

Associating data with a graph

When using a graph in a DataWindow object, you associate axes of the graph with columns in the DataWindow object.

In fact, 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 don't have to be displayed.

About the examples

The process of specifying data for a graph is illustrated below using the Printer table in the Powersoft Demo Database.

❖ **To specify data for a graph:**

- 1 If you have just placed a graph in the DataWindow painter, the Graph Object property sheet is displayed. Otherwise, select Properties from the graph's popup menu. Select the Data tab.
- 2 Fill in the boxes as described below.
- 3 Click OK.

Specifying which rows to include in a graph

The Rows dropdown listbox allows you to specify which rows of data are graphed at any one time:

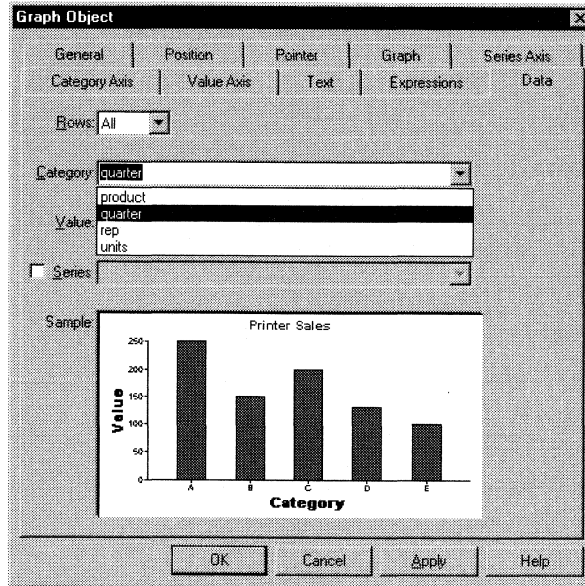
Setting	Meaning
All	Graphs the data from all the rows that have been retrieved but not filtered or deleted (that is, the rows in the primary buffer of the DataWindow object)
Page	Graphs only the data from the rows that are currently displayed on the page
Group n	Graphs only the data in the specified 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 may have several groups displayed at the same time, you should localize the graph in a group-related band in the workspace to ensure that it is clear which group the graph represents. Usually, the group header band is the most appropriate band.

Specifying the categories

Specify the column or expression whose values determine the categories. You can select a column name from the dropdown listbox or type an expression:



There will be an entry along the Category axis for each different value of the column or expression you specify here.

Using display values of data

If you are graphing columns that use code tables—data is stored with a data value but displayed to users with more meaningful display values—by default the graph will use the column's data values. To have the graph use a column's display values, use the `LookupDisplay` function when specifying Category or Series. `LookupDisplay` returns a string that matches the display value for a column:

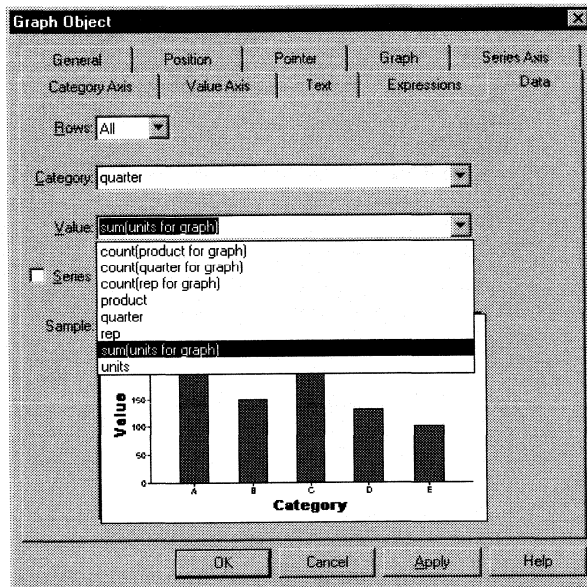
`LookupDisplay (column)`

FOR INFO For more about code tables, see "Defining a code table" on page 578. For more about `LookupDisplay`, see the *DataWindow Reference*.

Specifying the values

PowerBuilder populates the Value dropdown listbox in the Data property page. The listbox 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 dropdown listbox or type an expression. For example, if you want to graph the sum of units sold, you can specify:

```
sum(units for graph)
```

To graph 110 percent of the sum of units sold, you can specify:

```
sum(units*1.1 for graph)
```

Specifying the series

Graphs can have one or more series. You specify this in the Data property page.

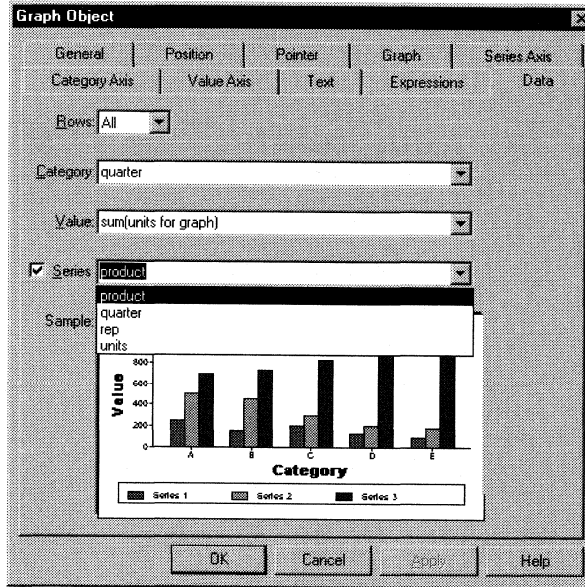
Single-series graphs

If you want only one series (that is, if you want to graph all retrieved rows as one series of values), leave the Series box empty.

Multiple-series graphs

If you want to graph more than one series, select the Series checkbox and specify the column that will provide the series values.

You can select column names from the dropdown listbox:



There will be a set of data points for each different value of the column you specify here. For example, if you specify in the Series box a column that has 10 values, then your graph will have 10 series: one set of data points for each different value of the column.

Using expressions

You can also specify expressions for Series. For example, you could specify the following for Series:

Units / 1000

In this case, if a table had unit values of 10,000, 20,000, and 30,000, the graph would show series values of 10, 20, and 30.

Specifying multiple entries

You can specify more than one of the retrieved columns to serve as series. Separate multiple entries by commas.

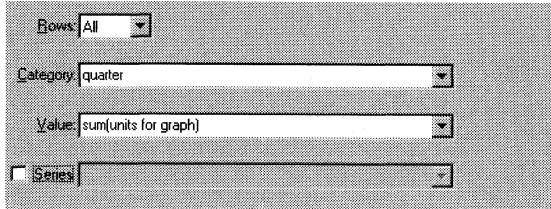
You need to specify the same number of entries in the Value box as you do in the Series box. The first value in the Value box corresponds to the first series identified in the Series box, the second value corresponds to the second series, and so on. The example about graphing actual and projected sales in "Examples" on page 699 illustrates this technique.

Examples

This section shows how to specify the data for several different graphs of the data in the Printer table in the Powersoft Demo Database. The table records quarterly unit sales of three printers by three sales representatives:

Rep	Quarter	Product	Units
Simpson	Q1	Stellar	12
Jones	Q1	Stellar	18
Perez	Q1	Stellar	15
Simpson	Q1	Cosmic	33
Jones	Q1	Cosmic	5
Perez	Q1	Cosmic	26
Simpson	Q1	Galactic	6
Jones	Q1	Galactic	2
Perez	Q1	Galactic	1
...
Simpson	Q4	Stellar	30
Jones	Q4	Stellar	24
Perez	Q4	Stellar	36
Simpson	Q4	Cosmic	60
Jones	Q4	Cosmic	52
Perez	Q4	Cosmic	48
Simpson	Q4	Galactic	3
Jones	Q4	Galactic	3
Perez	Q4	Galactic	6

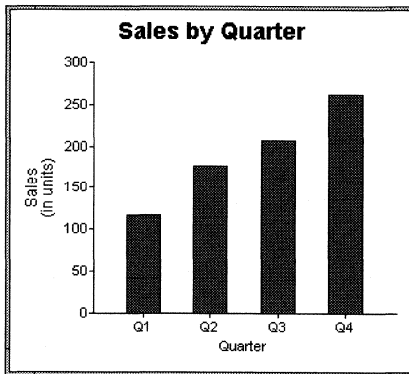
Graphing total sales Say you want to graph total sales of printers in each quarter. Retrieve all the columns into a DataWindow object and create a graph with the following data specification:



Rows: All
Category: quarter
Value: sum(units for graph)
Series:

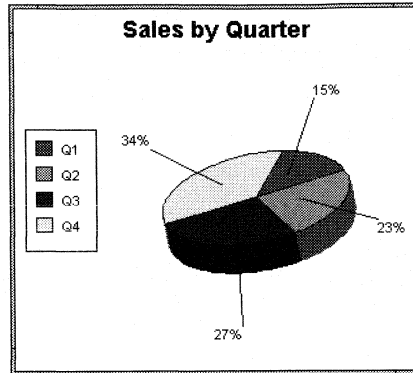
You want to graph sales by quarter, so the Quarter column serves as the category. Because the Quarter column has four values (Q1, Q2, Q3, and Q4), there will be four categories along the Category axis. You want only one series (total sales in each quarter), so you can leave the Series box empty as done in the preceding dialog box, or type a string literal to identify the series in a legend. You want to graph total sales in each quarter, so the Value box is specified as sum(units for graph).

Here is the resulting column graph. PowerBuilder automatically generates the category text based on the data in the table:



In the preceding graph, there is one set of data points (one series) across four quarters (the category values).

The following is a pie graph, which has exactly the same properties as the column graph above except for the type, which is 3D Pie:



In pie graphs, categories are shown in the legend.

Graphing unit sales of each printer Say you want to graph total quarterly sales of each printer. Create a graph with the following data specification:

Rows: All

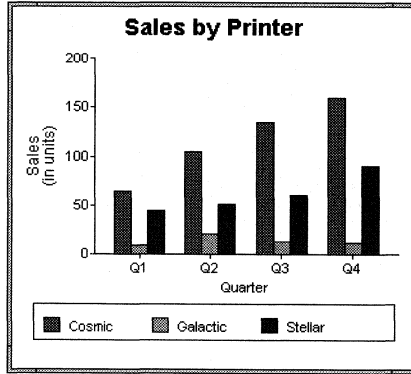
Category: quarter

Value: sum(units for graph)

Series: product

You want a different series for each printer, so the column Product serves as the series. Because the Product column has three values (Cosmic, Galactic, and Stellar), there will be three series in the graph. As in the first example, you want a value for each quarter, so the Quarter column serves as the category. And you want to graph total sales in each quarter, so the Value box is specified as sum(units for graph).

Here is the resulting graph. PowerBuilder automatically generates the category and series labels based on the data in the table. The series labels display in the graph's legend:

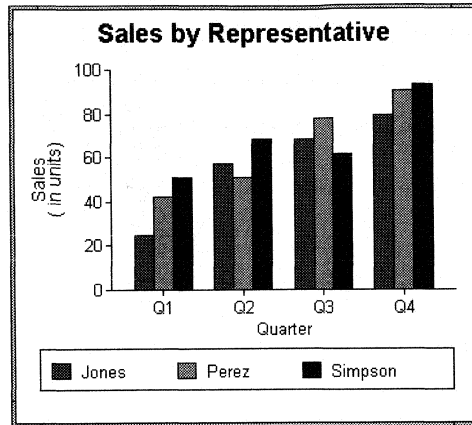


Graphing unit sales by representative Say you want to graph the quarterly sales made by each representative. Here you want a different series for each representative, so the column Rep serves as the series. Your category is still the Quarter column. And you still want to graph total sales in each quarter for each series, so the Value box is still sum(units for graph):

The screenshot shows a configuration dialog for a graph. The settings are as follows:

- Rows: All
- Category: quarter
- Value: sum(units for graph)
- Series: rep (checked)

Here is the resulting graph:



Graphing unit sales by representative and total sales Say you want to graph sales by representative, as shown above, plus total sales for each printer. Here is how to specify the data:

Rows: All

Category: quarter, "Total"

Value: sum(units for graph), sum(units for graph)

Series: rep, rep

Here you have two types of categories: the first is Quarter, which shows quarterly sales, as in the preceding graph. You also want a category for total sales. There is no corresponding column in the DataWindow object, so you can simply type the literal "Total" to identify the category. The Category box looks like this:

```
quarter, "Total"
```

You separate multiple entries with a comma.

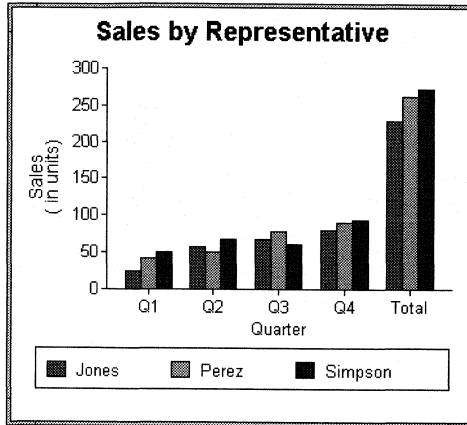
For each of these category types you want to graph the sum of units sold for each representative. So the Value entry is:

```
sum(units for graph), sum(units for graph)
```

And the Series entry is:

```
rep, rep
```

Here is the resulting graph:



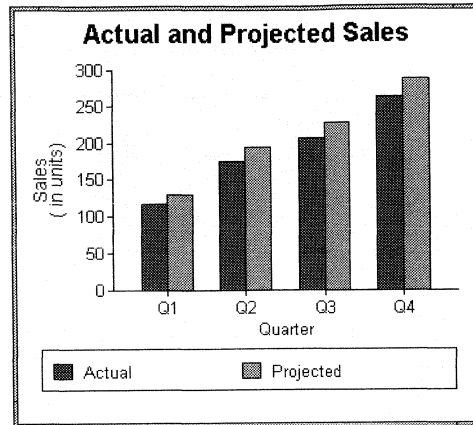
Notice that PowerBuilder uses the literal "Total" supplied in the Category box in the Graph Data window as a value in the Category axis.

Graphing actual and projected sales Say you want to graph total quarterly sales of all printers and also want to graph projected sales for next year. You figure that next year you will sell about 10 percent more printers. Here is how to specify the data:

The screenshot shows the configuration for a graph. The 'Rows' dropdown is set to 'All'. The 'Category' dropdown is set to 'quarter'. The 'Value' dropdown contains the expression 'sum(units for graph), sum(units*1.1 for graph)'. The 'Series' dropdown is checked and contains the literals 'Actual', 'Projected'.

You are using labels to identify two series, Actual and Projected. Note the single quotation marks around the literals. For Values, you enter the expressions that correspond to Actual and Projected sales. For Actual, you use the same expression as in the examples above, `sum(units for graph)`. For Projected sales, you multiply each unit sale by 1.1 to get the 10 percent increase. Therefore, the second expression is `sum(units*1.1 for graph)`.

Here is the resulting graph. PowerBuilder uses the literals you typed for the series as the series labels in the legend:



Using overlays

It is often useful to call special attention to one of the series in a graph, particularly in a bar or column graph. You can do that by defining the series as an **overlay**. An overlay series is graphed as a line on top of the other series in the graph. To define a series as an overlay, define the series in the Graph Object Data property page as follows:

- ◆ If specifying a column name to identify the series, specify this for the series:

"@overlay~t" + *ColumnName*

- ◆ If using a label to identify the series, specify this for the series:

"@overlay~tSeriesLabel "

Examples

Say you want to graph sales in each quarter and overlay the sales of each individual printer. Specify the graph's data like this:

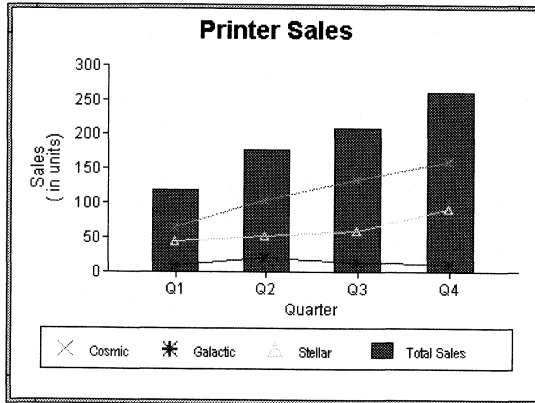
Rows: All

Category: quarter

Value: sum(units for graph)

Series: "Total Sales": "@overlay~t" + product

Here is the resulting graph:



To graph unit sales of printers by quarter and overlay the largest sale made in each quarter, specify the graph's data like this:

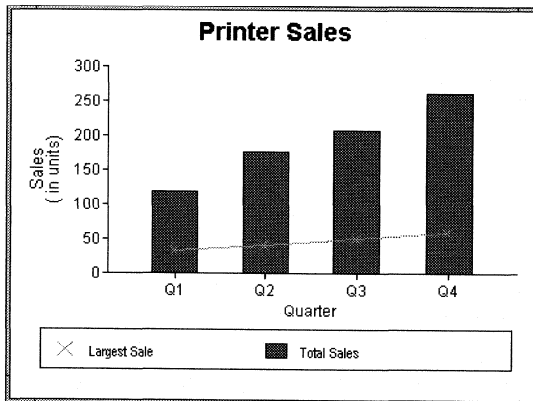
Rows: All

Category: quarter

Value: sum(units for graph), max(units for graph)

Series: "Total Sales", "@overlay" Largest Sale"

Here is the resulting graph:



Using the Graph presentation style

Instead of embedding a graph in a DataWindow object, you can use the Graph presentation style to create a DataWindow object that is only a graph—the underlying data is not displayed.

One advantage of the Graph presentation style is that the graph resizes automatically if users resize the DataWindow control associated with the graph DataWindow object during execution.

❖ To use the Graph presentation style:

- 1 Open the DataWindow painter and select New in the Select DataWindow dialog box.

The New DataWindow dialog box displays.

- 2 Select a data source and the Graph presentation style, then click OK.

You are prompted to specify the data.

- 3 Specify the data you want retrieved into the DataWindow object.

FOR INFO For more information, see Chapter 14, "Defining DataWindow Objects".

The DataWindow painter workspace displays, and you are prompted to specify the data for the graph in the graph's property sheet.

- 4 Enter the definitions for the series, categories, and values, as described in "Associating data with a graph" on page 694.

Note that the Rows box is protected. When using the Graph presentation style, the graph always graphs all rows; you cannot specify page or group.

- 5 Click OK.

A model of the graph displays in the entire workspace.

- 6 Specify the properties of the graph, as described in "Defining a graph's properties" next.

- 7 Save the DataWindow object in a library.

- 8 Associate the graph DataWindow object with a DataWindow control in a window or user object.

During execution, the graph fills the entire control and resizes when the control is resized.

Defining a graph's properties

This section describes properties of a graph that are used regardless of whether the graph is in a DataWindow object or in a window.

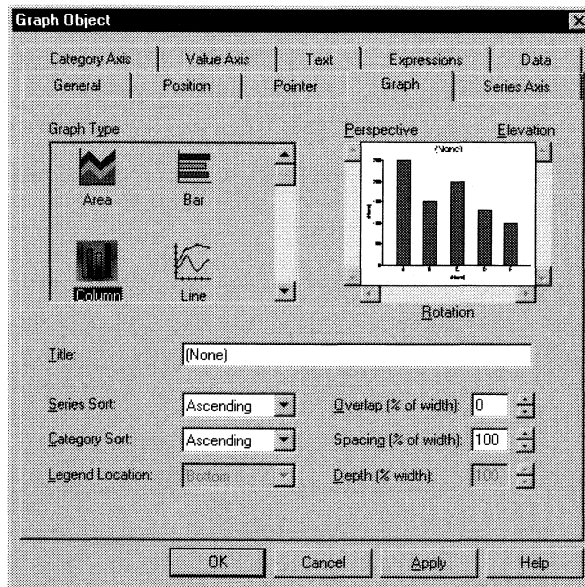
To define the properties of a graph, you use the graph's property sheet.

FOR INFO For general information about the ten tabbed property pages on the Graph Object property sheet, see "Using the graph's property sheet" on page 691.

Using the Graph property page of the graph's property sheet

One of the first things you will probably do with a graph is name it and define its basic properties. You can do that on the Graph property page of the graph's property sheet.

- ❖ **To specify the basic properties of a graph:**
 - ◆ Select Properties from the graph's popup menu and then select the Graph tab:



About the model graph

As you modify your graph's properties, PowerBuilder updates the model graph shown on the Graph property page so you can get an idea of the graph's basic layout:

- ◆ PowerBuilder uses the graph title and axis labels you specify
- ◆ PowerBuilder uses sample data (not data from your DataWindow object) to illustrate series, categories, and values

Naming a graph

You can modify graphs in scripts during execution. To reference a graph during execution, you use its name.

❖ **To name a graph:**

- 1 Select Properties from the graph's popup menu and then select the General tab.
- 2 Assign a meaningful name to the graph in the Name box.
- 3 Click Apply or OK.

Defining a graph's title

The title displays at the top of the graph.

❖ **To specify a graph's title:**

- 1 Select Properties from the graph's popup menu and then select the Graph tab.
- 2 Enter a title in the Title box.
- 3 Click Apply or OK.

Multiline titles

You can force a new line in a title by embedding ~n.

FOR INFO For information about specifying properties for the title text, see "Specifying text properties for titles, labels, axes, and legends" on page 711.

Specifying the type of graph

You can change the graph type anytime in the development environment (you can also change the type during execution by modifying a graph's `GraphType` property).

❖ **To specify the graph type:**

- 1 Select Properties from the graph's popup menu and then select the Graph tab.
- 2 Scroll the presentation of graph types to see all that are available.
- 3 Click the icon representing the graph type you want.
- 4 Click Apply or OK.

Using legends

A legend provides a key to your graph's series.

❖ **To include a legend for a series in a graph:**

- 1 Select Properties from the graph's popup menu and then select the Graph tab.
- 2 Specify where you want the legend to appear by selecting a value in the Legend Location dropdown listbox.
- 3 Click Apply or OK.

FOR INFO For information on specifying text properties for the legend, see "Specifying text properties for titles, labels, axes, and legends" on page 711.

Sorting data

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:**

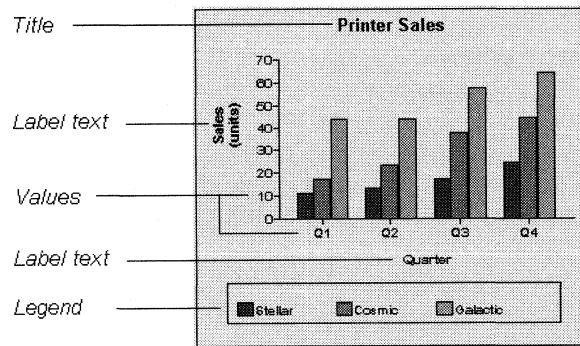
- 1 Select Properties from the graph's popup menu and then select the Graph tab.
- 2 Select Ascending (order), Descending (order), or Unsorted in the Series Sort box or the Category Sort box.
- 3 Click Apply or OK.

You can choose unsorted, sort in ascending order, or sort in descending order.

Specifying text properties for titles, labels, axes, and legends

A graph can have four text elements:

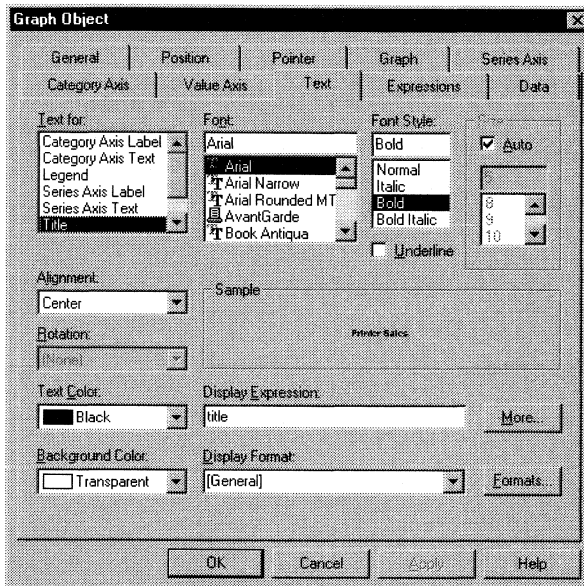
- ◆ Title
- ◆ Labels for the axes
- ◆ Text that shows the values along the axes
- ◆ Legend



You can specify properties for each text element.

- ❖ **To specify text properties for the title, labels, axes values, and legend of a graph:**
 - 1 Select Properties from the graph's popup menu and then select the Text tab.

- 2 Select a text element from the list in the Text For box:



- 3 Specify the font and its characteristics.
- 4 Click Apply or OK.

Using Auto Size

With Auto Size in effect, PowerBuilder resizes the text appropriately whenever the graph is resized. With Auto Size disabled, you specify the font size of a text element explicitly.

- ❖ **To have PowerBuilder automatically size a text element in a graph:**
 - 1 Select Properties from the graph's popup menu and then select the Text tab.
 - 2 Select a text element from the list in the Text For box.
 - 3 Select the Auto checkbox (this is the default).
 - 4 Click Apply or OK.
- ❖ **To specify a font size for a text element in a graph:**
 - 1 Select Properties from the graph's popup menu and then select the Text tab.

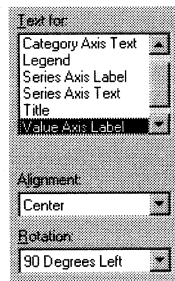
- 2 Select a text element from the list in the Text For box.
- 3 Deselect the Auto checkbox.
- 4 Select the Font size in the Size dropdown listbox.
- 5 Click Apply or OK.

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 Select Properties from the graph's popup menu and then select the Text tab.
- 2 Select a text element from the list in the Text For box.
- 3 Specify the rotation you want:



- 4 Click Apply or OK.

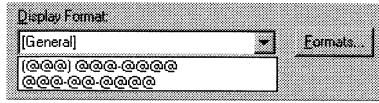
Changes you make here are shown in the model graph in the workspace and on the Graph property page of the graph's property sheet.

Using display formats

❖ To use a display format for a text element in a graph:

- 1 Select Properties from the graph's popup menu and then select the Text tab.
- 2 Select a text element from the list in the Text For box.

- 3 Choose an existing display format for the text from the Display Format dropdown listbox, or define a new display format by clicking the Formats button:



- 4 Click Apply or OK.

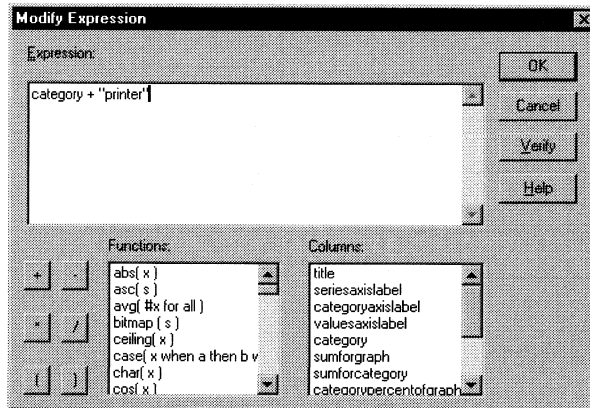
Modifying display expressions

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 Select Properties from the graph's popup menu and then select the Text tab.
- 2 Select a text element from the list in the Text For box.
- 3 Click the More button.

The Modify Expression dialog box displays:



- 4 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.

- 5 Click OK to return to the graph's property sheet.
- 6 Click Apply or OK.

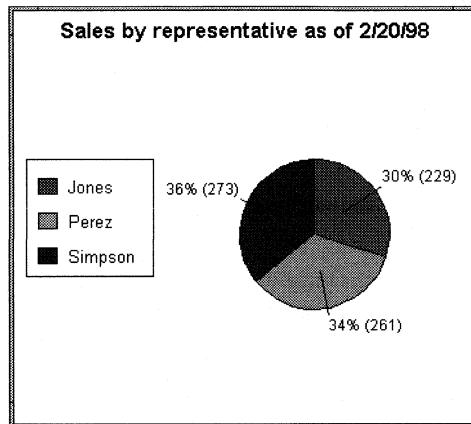
Example

Here's an example of using expressions to enhance a graph.

By default, when you generate a pie graph, PowerBuilder puts the title at the top and labels each slice of the pie with the percentage each slice is of the whole. Percentages are accurate to two decimal places.

The following graph has been enhanced as follows:

- ◆ The current date displays in the title
- ◆ The percentages are rounded to integers
- ◆ The raw data for each slice is shown in addition to the percentages



To accomplish this, the display expressions were modified for the title and pie graph labels:

Text 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 following properties:

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 popup menu and then select the Graph tab.
- 2 Specify a percentage for Overlap (% of width) and Spacing (% of width).
- 3 Click Apply or OK.

Specifying axis properties

Graphs have two or three axes. You specify the axes' properties from the Category Axis, Value Axis, or Series Axis property pages of the graph's property sheet.

❖ **To specify properties for an axis of a graph:**

- 1 Select Properties from the graph's popup menu.
- 2 Select the Category Axis tab, the Value Axis tab, or the Series Axis tab.
If you are not working with a 3D graph, the Series Axis options are disabled.
- 3 Specify the properties as described next.
- 4 Click Apply or OK.

Specifying text properties

You can specify the characteristics of text that displays for each axis. There are two kinds of text associated with an axis:

Type of text	Meaning
Text	Text that identifies the values for an axis
Label	Text that describes the axis. You specify the label text in a painter. You can use ~n to embed a new line within a label

FOR INFO For information on specifying properties for the text, see "Specifying text properties for titles, labels, axes, and legends" on page 711.

Specifying data types

The data graphed along the Value, Category, and Series axes has an assigned data type. The Series axis always has the data type String. The Value and Category axes can have the following data types:

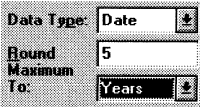
Axis	Possible data types
Value	Number, Date, DateTime, Time
Category (for scatter graph)	Number, Date, DateTime, Time
Category (other graph types)	String, Number, Date, DateTime, Time

For graphs in DataWindow objects, PowerBuilder automatically assigns the data types based on the data type of the corresponding column; you do not specify them.

For graphs in windows, you specify the data types yourself. Be sure you specify the appropriate data types, so when you populate the graph (using the AddData function), the data matches the data type.

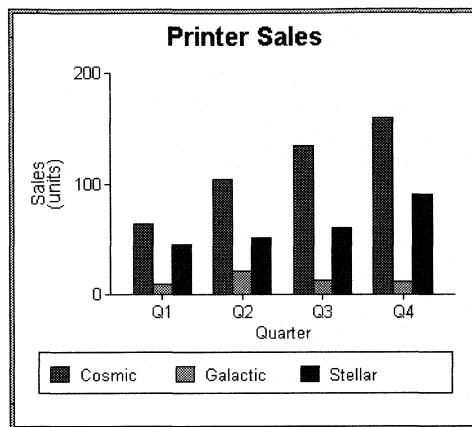
Scaling axes

You can specify several properties that define the scaling used along numeric axes:

Property	Meaning
Autoscale	If selected (the default), PowerBuilder automatically assigns a scaling for the numbers along the axis
Round Maximum To	<p>Specifies how to round the end points of the axis (note that this just rounds the range displayed along the axis; it doesn't round the data itself)</p> <p>You can specify a number and a unit. The unit is based on the data type; you can specify Default as the unit to have PowerBuilder decide for you</p> <p>For example, if the Value axis is a Date column, you can specify that you want to round the end points of the axis to the nearest five years. In this case, if the largest data value is the year 1993, the axis will extend up to 1995, which is 1993 rounded to the next highest five-year interval:</p> 
Minimum, Maximum	The smallest and largest numbers to appear on the axis (disabled if you have selected Autoscale)
Scale	Specifies linear or logarithmic scaling (common or natural)

Using major and minor divisions

You can divide axes into divisions. Each division is identified by a tick mark, which is a short line that intersects an axis. The following graph's Value axis is divided into two major divisions. One goes from 0 to 100. The other goes from 100 to 200:

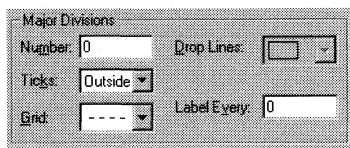


By default, PowerBuilder divides the axes automatically into major divisions.

❖ To define divisions for an axis of a graph:

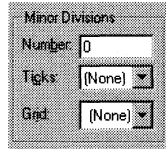
- 1 To divide an axis into a specific number of major divisions, type the number of divisions you want in the Number box in the Major Divisions group.

Leave the number 0 to have PowerBuilder automatically create divisions.



By default, PowerBuilder labels each tick mark in major divisions. If you don't want each tick mark labeled, enter a value in the Label Every box. For example, if you enter 2, PowerBuilder will label every second tick mark for the major divisions.

- To use minor divisions, which are divisions within each major division, type the appropriate number in the Number box in the Minor Divisions group. To not use minor divisions, leave the number 0:



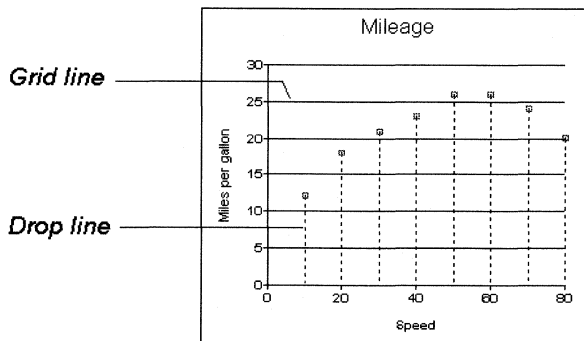
When using logarithmic axes

If you want minor divisions, specify 1; otherwise, specify 0.

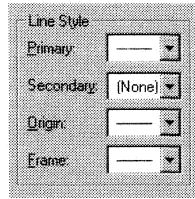
Specifying division lines

You can specify lines to represent the divisions:

Line	Meaning
Grid line	A line that extends from a tick mark across the graph. Grid lines make graphs easier to read
Drop line	A line that extends vertically from a data point to its axis (not available for all graph types)



Using line styles



You can define line styles for the following components of an axis:

Component	Meaning
Primary	The axis itself
Secondary	The axis parallel to and opposite the primary axis
Origin	A grid line that represents the value zero
Frame	The frame for the axis in 3D graphs (disabled for 2D graphs)

Specifying a border

You can specify the border that PowerBuilder places around a graph.

❖ To specify a border for a graph:

- 1 Select the graph.
- 2 Select the type of border to use from the Border dropdown toolbar.

Specifying a pointer

You can specify a pointer to use when the mouse is over a graph during execution.

❖ To specify a pointer for a graph:

- 1 Select Properties from the graph's popup menu and then select the Pointer tab.
- 2 Select a stock pointer from the list, or select a CUR file containing a pointer.
- 3 Click Apply or OK.

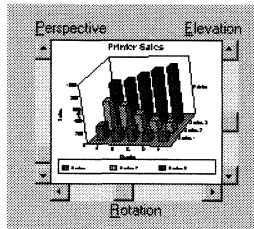
Specifying point of view in 3D graphs

If you are defining a 3D graph, you can specify the point of view that PowerBuilder uses when displaying the graph.

❖ **To specify a 3D graph's point of view:**

- 1 Select Properties from the graph's popup menu and then select the Graph tab.

The Graph property page includes a representation of the graph and scroll bars that let you change the point of view for a 3D graph:



- 2 Adjust the point of view along the three dimensions of the graph:
 - ◆ To change the perspective, move the scroll box in the left scrollbar.
 - ◆ To rotate the graph, move the scroll box in the bottom scrollbar.
 - ◆ To change the elevation, move the scroll box in the right scrollbar.
- 3 Define the depth of the graph (the percent the depth is of the width of the graph) by entering a number in the Depth (% width) box or using the scrollbars on the graph representation to increase or decrease the depth.
- 4 Click OK.

Using graphs in windows

In addition to using graphs in DataWindow objects, you can also place graphs directly in windows. You define 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.

FOR INFO For general graph properties, see "Defining a graph's properties" on page 708.

About user objects

You can also place graphs in user objects. Everything below about using graphs in windows also applies to using them in user objects.

Placing a graph in a window

❖ To place a graph in a window:

- 1 Open the Window painter and select the window that will contain the graph.
The Window painter workspace displays showing the window.
- 2 Click the Graph button in the Objects dropdown toolbar.
- 3 Click where you want the graph.
PowerBuilder displays a model of the graph in the window.
- 4 Specify properties for the graph, such as type, title, text properties, and axis properties.
FOR INFO See "Defining a graph's properties" on page 708.
- 5 Write one or more scripts to populate the graph with data.
FOR INFO See the chapter on manipulating graphs in *Application Techniques*.

Using the graph's property sheet

A graph's property sheet in the Window and User Object painters is similar to the property sheet in the DataWindow painter except that the property sheet in the Window and User Object painter:

- ◆ Does not have an Expressions or Data property page
- ◆ Does have a Drag and Drop property page, which you use to specify drag-and-drop properties for the graph control

FOR INFO For more information, see "Using the graph's property sheet" on page 691.

Working with Crosstabs

About this chapter

This chapter describes how to build crosstabs.

Contents

Topic	Page
About crosstabs	726
Creating crosstabs	730
Associating data with a crosstab	731
Previewing crosstabs	737
Enhancing crosstabs	738

About crosstabs

Cross tabulation is a useful technique for analyzing data. By presenting data in a spreadsheet-like grid, a crosstab lets users view summary data instead of a long series of rows and columns. For example, in a sales application you might want to summarize the quarterly unit sales of each product.

In PowerBuilder, you create crosstabs by using the Crosstab DataWindow presentation style. When data is retrieved into the DataWindow object, the crosstab processes all the data and presents the summary information that you have defined for it.

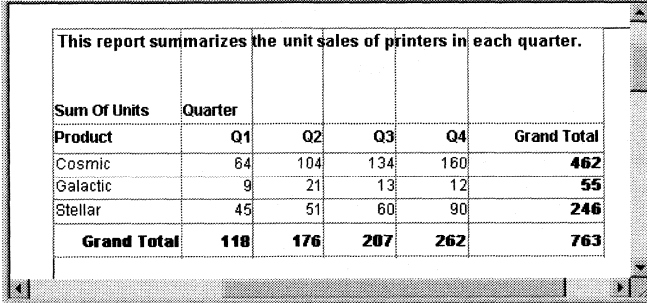
An example

Crosstabs are easiest to understand through an example. Consider the Printer table in the Powersoft Demo Database. It records quarterly unit sales of printers made by sales representatives in one year (this is the same data that was used to illustrate graphs in the preceding chapter):

Rep	Quarter	Product	Units
Simpson	Q1	Stellar	12
Jones	Q1	Stellar	18
Perez	Q1	Stellar	15
Simpson	Q1	Cosmic	33
Jones	Q1	Cosmic	5
Perez	Q1	Cosmic	26
Simpson	Q1	Galactic	6
Jones	Q1	Galactic	2
Perez	Q1	Galactic	1
.	.	.	.
.	.	.	.
.	.	.	.
Simpson	Q4	Stellar	30
Jones	Q4	Stellar	24
Perez	Q4	Stellar	36
Simpson	Q4	Cosmic	60
Jones	Q4	Cosmic	52

Rep	Quarter	Product	Units
Perez	Q4	Cosmic	48
Simpson	Q4	Galactic	3
Jones	Q4	Galactic	3
Perez	Q4	Galactic	6

This information can be summarized in a crosstab. Here is a crosstab that shows unit sales by printer for each quarter:



This report summarizes the unit sales of printers in each quarter.					
Sum Of Units	Quarter				
Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	60	90	246
Grand Total	118	176	207	262	763

The first-quarter sales of Cosmic printers displays in the first data cell (as you can see from the data in the Printer table shown before the crosstab, in Q1 Simpson sold 33 units, Jones sold 5 units, and Perez sold 26 units—totaling 64 units). PowerBuilder calculates each of the other data cells the same way.

To create this crosstab, all you have to do is tell PowerBuilder which database columns contain the raw data for the crosstab and PowerBuilder does all the data summarization automatically.

What crosstabs do

Crosstabs perform two-dimensional analysis:

- ◆ The first dimension is displayed as columns across the crosstab.

In the preceding crosstab, it is the quarter, whose values are in the Quarter column in the database table.

- ◆ The second dimension is displayed as rows down the crosstab.

In the preceding crosstab, it is the type of printer, whose values are in the Product column in the database table.

Each cell in a crosstab is the intersection of a column (the first dimension) and a row (the second dimension). The numbers that appear in the cells are calculations based on both dimensions. In the preceding crosstab, it is the sum of unit sales for the quarter in the corresponding column and printer in the corresponding row.

Crosstabs can also include summary statistics. The preceding crosstab totals the sales for each quarter in the last row and the total sales for each printer in the last column.

How crosstabs are implemented in PowerBuilder

Crosstabs in PowerBuilder are implemented as grid DataWindow objects. Because crosstabs are grid DataWindow objects, users can resize and reorder columns during execution (if you let them).

Two types of crosstabs

There are two types of crosstabs:

- ◆ Dynamic
- ◆ Static

Dynamic crosstabs

With **dynamic crosstabs**, PowerBuilder builds all the columns and rows in the crosstab dynamically when you run the crosstab. The number of columns and rows in the crosstab match the data that exists at execution time.

Using the preceding crosstab as an example, if a new printer was added to the database after the crosstab was saved, there would be an additional row in the crosstab when it is run. Similarly, if one of the quarter's results was deleted from the database after the crosstab was saved, there would be one less column in the crosstab when it is run.

By default, crosstabs you build are dynamic.

Static crosstabs

Static crosstabs are quite different.

With static crosstabs, PowerBuilder establishes the columns in the crosstab based on the data in the database when you *define* the crosstab (it does this by retrieving data from the database when you initially define the crosstab in the DataWindow painter workspace). No matter what values are in the database when you later run the crosstab, the crosstab will always have the same columns as when you defined it.

Using the preceding crosstab as an example, if there were four quarters in the database when you defined and saved the crosstab, there would always be four columns (Q1, Q2, Q3, and Q4) in the crosstab at execution time, even if the number of columns changed in the database.

Advantages of dynamic crosstabs

In most cases, you will want to use dynamic crosstabs, for the following reasons:

- ◆ You can define dynamic crosstabs very quickly because no database access is required at definition time.
- ◆ Dynamic crosstabs always use the current data to build the columns and rows in the crosstab. Static crosstabs show a snapshot of columns as they were when the crosstab was defined.
- ◆ Dynamic crosstabs are easy to modify: all properties for the dynamically built columns are replicated during execution automatically. With static crosstabs, you must work with one column at a time.

Creating crosstabs

❖ **To create a crosstab:**

- 1 Open the DataWindow painter and select New in the Select DataWindow dialog box.

The New DataWindow dialog box displays.

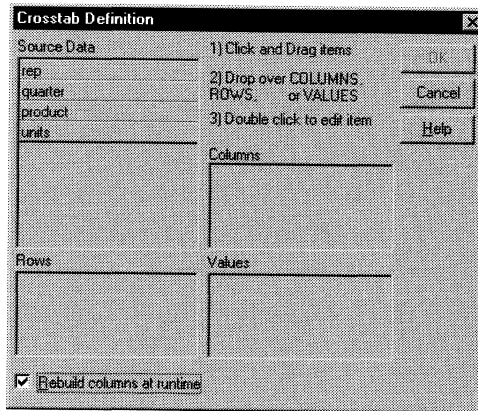
- 2 Select a data source and the Crosstab presentation style, then click OK.

You are prompted to specify the data.

- 3 Specify the data you want retrieved into the DataWindow object.

FOR INFO For more information, see Chapter 14, "Defining DataWindow Objects".

You are prompted to specify the data for the columns, rows, and cell values in the crosstab:



- 4 Enter the definitions for the columns, rows, and cell values in the crosstab.

FOR INFO See "Associating data with a crosstab" on page 731.

- 5 Click OK.

PowerBuilder places the crosstab in the workspace.

- 6 (Optional) Preview the crosstab to see how it looks.

FOR INFO See "Previewing crosstabs" on page 737.

- 7 (Optional) Specify other properties of the crosstab.

FOR INFO See "Enhancing crosstabs" on page 738.

- 8 Save the DataWindow object in a library.

Associating data with a crosstab

You associate crosstab columns, rows, and cell values with columns in a database table or other data source.

❖ To associate data with a crosstab:

- 1 If you are defining a new crosstab, the Crosstab Definition dialog box displays after you specify the data source.
- 2 Specify the database columns that will populate the columns, rows, and values in the crosstab, as described below.
- 3 To build a dynamic crosstab, make sure the Rebuild Columns At Runtime box is selected.

FOR INFO For information about static crosstabs, see "Creating static crosstabs" on page 747.

- 4 Click OK.

Specifying the information

To define the crosstab, you simply drag the column names from the Source Data box in the Crosstab Definition dialog box into the Columns, Rows, or Values box, as appropriate.

If you change your mind or want to edit the DataWindow object later, drag the column name out of the Columns, Row, or Values box and drop it. Then specify a different column.

The process is illustrated using the following dynamic crosstab. The columns in the database are Rep, Quarter, Product, and Units. The crosstab shows the number of printers sold by Quarter:

Sum Of Units	Quarter					
Product	Q1	Q2	Q3	Q4	Grand Total	
Cosmic	64	104	134	160	462	
Galactic	9	21	13	12	55	
Stellar	45	51	60	90	246	
Grand Total	118	176	207	262	763	

Specifying the columns

You use the Columns box to specify one or more of the retrieved columns to provide the columns in the crosstab. When users run the crosstab, there will be one column in the crosstab for each unique value of the database column(s) you specify here.

❖ **To specify the crosstab's columns:**

- ◆ Drag the database column from the Source Data box into the Columns box.

Using the printer example, to create a crosstab where the quarters form the columns, specify Quarter as the Columns value. Because there are four values in the table for Quarter (Q1, Q2, Q3, and Q4), there will be four columns in the crosstab.

Specifying the rows

You use the Rows box to specify one or more of the retrieved columns to provide the rows in the crosstab. When users run the crosstab, there will be one row in the crosstab for each unique value of the database column(s) you specify here.

❖ **To specify the crosstab's rows:**

- ◆ Drag the database column from the Source Data box into the Rows box.

Using the printer example, to create a crosstab where the printers form the rows, specify Product as the Rows value. Because there are three products (Cosmic, Galactic, and Stellar), during execution there will be three rows in the crosstab.

Display values are used

If you specify columns in the database that use code tables—data is stored with a data value but displayed with more meaningful display values—the crosstab uses the column's display values, not the data values.

FOR INFO For more about code tables, see Chapter 16, "Displaying and Validating Data".

Using expressions

Instead of simply specifying database columns, you can use any valid PowerScript expression to define the columns, rows, and values used in the crosstab. You can use any non-object-level PowerScript function in the expression.

For example, say a table contains a date column named `SaleDate`, and you want a column in the crosstab for each month. You could enter the following expression for the Columns definition:

```
Month(SaleDate)
```

The `Month` PowerScript function returns the integer value (1–12) for the specified month. Using this expression, you get columns labeled 1 through 12 in the crosstab. Each database row for January sales is evaluated in the column under 1, each database row for February sales is evaluated in the column under 2, and so on.

❖ To specify an expression for columns, rows, or values:

- 1 In the Crosstab Definition dialog box, double-click the item in the Columns, Rows, or Values box.

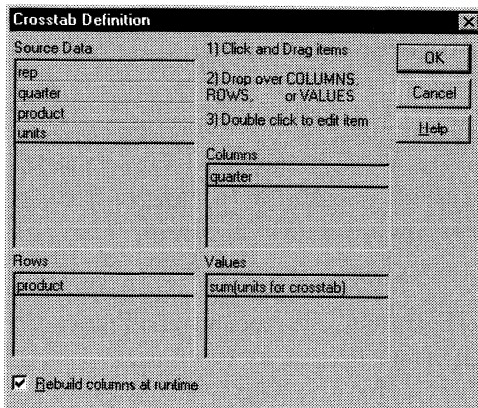
The Modify Expression dialog box displays.

- 2 Specify the expression and click OK.

What happens

After you have specified the data for the crosstab's columns, rows, and values, PowerBuilder displays the crosstab definition in the DataWindow painter workspace. The crosstab is implemented as a grid DataWindow object.

To create the dynamic crosstab shown earlier, you would enter these values:



In the workspace, the crosstab looks like this:

Sum Of Units	Quarter		
Header [1] ↑			
Product	@quarter	Grand Total	
Header [2] ↑			
product	units	crosstabsum(1)	
Detail ↑			
"Grand Total"	sum(units	sum(grand_sum_units for	
Summary ↑			
Footer ↑			

Notice that in the workspace PowerBuilder shows the Quarter entries using the symbolic notation **@quarter** (with dynamic crosstabs, the actual data values are not known at definition time). **@quarter** is resolved into the actual data values (in this case, Q1, Q2, Q3, and Q4) when the crosstab runs.

The crosstab is generated with summary statistics: the rows and columns are totaled for you.

At this point, you can run the crosstab by clicking the Preview button:

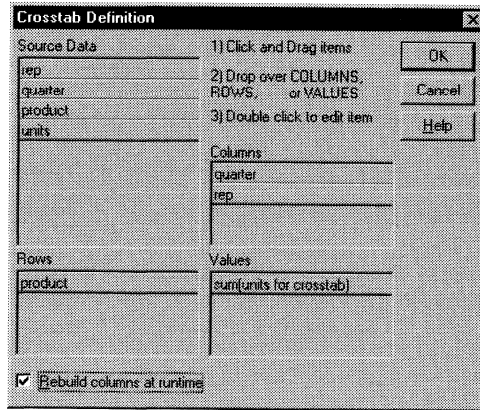
Sum Of Units	Quarter				
Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	60	90	246
Grand Total	118	176	207	262	763

- ◆ Because Quarter was selected as the Columns definition, there is one column in the crosstab for each unique quarter (Q1, Q2, Q3, and Q4)
- ◆ Because Product was selected as the Rows definition, there is one row in the crosstab for each unique product (Cosmic, Galactic, and Stellar)
- ◆ Because sum(units for crosstab) was selected as the Values definition, each cell contains the total unit sales for the corresponding quarter (the Columns definition) and product (the Rows definition)
- ◆ PowerBuilder displays the grand totals for each column and row in the crosstab

Specifying more than one row or column

Typically you will specify one database column as the Columns definition and one database column for the Rows definition, as in the printer crosstab. But you can specify as many columns (or expressions) as you want.

For example, consider the following, which specifies two database columns as the Columns definition:



Here we want columns in the crosstab for quarters and for sales reps.

PowerBuilder displays this in the workspace:

Sum Of Units	Quarter	Rep		
Header (1) :				
	@quarter	@quarter	Sum Of Units	
Header (2) :				
Product	@rep		Grand Total	
Header (3) :				
product	units	crosstabsum(1, 2, "@quarter", crosstabsum(1))		
Detail :				
"Grand Total"	sum(units sum(sum_units for all))	sum(grand_sum_units for		
Summary :				
Footer :				

This is what you get during execution:

Sum Of Units	Quarter		Rep		O1 Sum Of Units	O2 Sum Of Units			O2 Sum Of Units
	Q1	Q2	Jones	Perez		Simpson	Jones	Perez	
Cosmic	5	26	33	64	26	40			104
Galactic	2	1	6	9	6	12			21
Stellar	18	15	12	45	15	16			51
Grand Total	25	42	51	118	57	68			176

For each quarter, the crosstab shows sales of each printer by each sales representative.

Previewing crosstabs

Once you have defined the data for the crosstab, you can run it to see the data. You run it by previewing the DataWindow object.

❖ **To preview the crosstab:**

- 1 Click the Preview button.

or

Select Design>Preview from the menu bar.

You are now in preview. The bars that indicate the DataWindow object bands disappear. PowerBuilder retrieves the rows and performs the cross tabulation on the data.

- 2 Continue previewing your DataWindow object.

You can do the same things you do when previewing standard grid DataWindow objects. For example, you can resize columns, filter rows, sort rows, save data in an external file, and print the results.

FOR INFO For more on what you can do in preview, see Chapter 15, "Enhancing DataWindow Objects".

- 3 When you have finished previewing the crosstab, click the Preview button.

You return to the workspace. Changes you made (such as resizing columns) are retained.

Enhancing crosstabs

Once you have provided the data definitions, the crosstab is functional. But you will probably want to enhance it before using it. Because a crosstab is a grid DataWindow object, you can enhance a crosstab using the same techniques you use in other DataWindow objects. For example, you might want to:

- ◆ Sort or filter rows
- ◆ Change the column headers
- ◆ Specify fonts, colors, mouse pointers, and borders
- ◆ Specify column display formats

FOR INFO For more on these and the other standard enhancements you can make to DataWindow objects, see Chapter 15, "Enhancing DataWindow Objects".

The rest of this section covers topics either unique to crosstabs or especially important when working with crosstabs:

- ◆ "Specifying basic properties" next
- ◆ "Modifying the data associated with the crosstab" on page 739
- ◆ "Changing the names used for the columns and rows" on page 740
- ◆ "Defining summary statistics" on page 741
- ◆ "Cross-tabulating ranges of values" on page 744
- ◆ "Creating static crosstabs" on page 747
- ◆ "Using property conditional expressions" on page 748

Specifying basic properties

Crosstabs are implemented as grid DataWindow objects, so you can specify the following grid properties for a crosstab:

- ◆ When grid lines are displayed
 - ◆ How users can interact with the crosstab during execution
- ❖ **To specify the crosstab's basic properties:**
- 1 Position the mouse below the footer band in the workspace and display the popup menu.
 - 2 Select Properties and then the General tab.

- 3 Specify basic crosstab properties in the Grid box on the General property page:

Option	Result
Display	<p><i>On</i> — Grid lines always display</p> <p><i>Off</i> — Grid lines never display (columns cannot be resized during execution)</p> <p><i>Display Only</i> — Grid lines display only when the crosstab displays online</p> <p><i>Print Only</i> — Grid lines display only when the contents of the crosstab are printed</p>
Column Moving	Columns can be moved during execution
Mouse Selection	Data can be selected during execution (and, for example, copied to the clipboard)
Row Resize	Rows can be resized during execution

Modifying the data associated with the crosstab

When you initially define the crosstab, you associate the crosstab rows and columns with columns in a database table or other data source. You can change the associated data anytime in the Crosstab Definition dialog box.

❖ To open the Crosstab Definition dialog box:

- 1 Position the mouse below the footer band in the workspace and display the popup menu.
- 2 Select Crosstab from the popup menu.
The Crosstab Definition dialog box displays.

❖ To modify the data associated with a crosstab:

- 1 In the Crosstab Definition dialog box, fill in the boxes for Columns, Rows, and Values as described in "Associating data with a crosstab" on page 731.
- 2 Click OK.

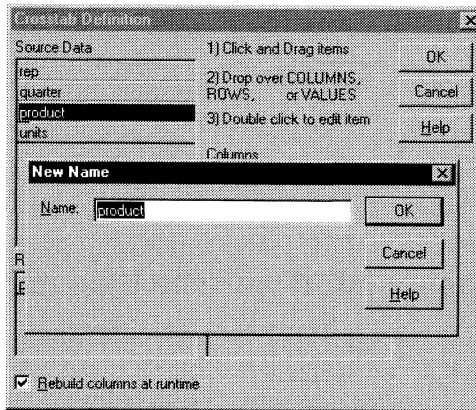
Changing the names used for the columns and rows

Sometimes names of columns in the database are not very user friendly and may not be meaningful. You can change the names that are used to label rows and columns in crosstabs so that the data is easier to understand.

❖ **To change the names used in crosstabs:**

- 1 In the Crosstab Definition dialog box, double-click the name of the column in the Source Data box.

The New Name dialog box displays:



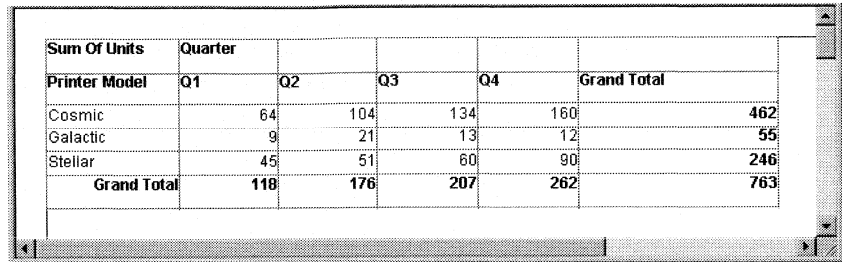
- 2 Specify the name you want to be used to label the corresponding column. You can have multiple-word labels by using underscores: underscores are replaced by spaces in the workspace and during execution.
- 3 Click OK.

PowerBuilder changes the column name in the Source Data box and anywhere else the column is used.

Example

For example, if you want the Product column to be labeled *Printer Model*, double-click Product in the Crosstab Definition dialog box and specify **printer_model** in the New Name dialog box.

When the crosstab runs, you will see this:



Sum Of Units	Quarter				
Printer Model	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	80	90	246
Grand Total	118	176	207	262	763

Defining summary statistics

When you generate a crosstab, the columns and rows are automatically totaled for you. You can include other statistical summaries in crosstabs as well. To do that, you place computed fields in the workspace.

❖ To define a column summary:

- 1 Enlarge the summary band to make room for the summaries.
- 2 Click the Compute button in the PainterBar.
or
Select Objects>Computed Field from the menu bar.
- 3 Click the cell in the summary band where you want the summary to display.
The Computed Object property sheet displays.
- 4 Define the computed field.

For example, if you want the average value for a column specify **avg(units for all)**, where Units is the column providing the values in the crosstab.

Here is a crosstab that has been enhanced to show averages and maximum values for each column. This is the workspace:

This report summarizes the unit sales of printers in each quarter.

Sum Of Units	Quarter				
Header[1]					
Product	@quarter	Grand Total			
Header[2]					
product	units	Crosstabsum(1)			
Detail					
"Grand Total"	sum(units	sum(grand_sun			
Average	avg(units				
Maximum	max(unit:				
Summary					
Footer					

And this is the crosstab during execution:

This report summarizes the unit sales of printers in each quarter.

Sum Of Units	Quarter					
Product	Q1	Q2	Q3	Q4	Grand Total	
Cosmic	64	104	134	160	462	
Galactic	9	21	13	12	55	
Stellar	45	51	60	90	246	
Grand Total	118	176	207	262	763	
Average	39	59	69	87		
Maximum	64	104	134	160		

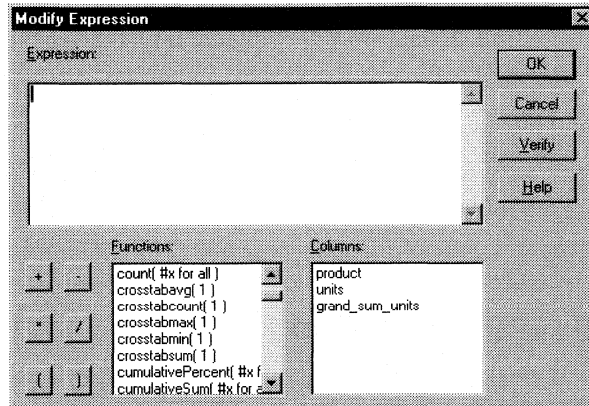
❖ **To define a row summary:**

- 1 Click the Compute button in the PainterBar.
or
Select Objects>Computed Field from the menu bar.
- 2 Click the empty cell to the right of the last column in the detail band.
The Computed Object property sheet displays.
- 3 Define the computed field. You should use one of the crosstab functions, described next. To display the Modify Expression dialog box, click More.

Using crosstab functions

There are five special functions you can use only in crosstabs: CrosstabAvg, CrosstabCount, CrosstabMax, CrosstabMin, and CrosstabSum.

These functions are listed in the Functions box when you define a computed field in a crosstab:



Each of these functions returns the corresponding statistic about a row in the crosstab (average, count, maximum value, minimum value, or sum). You place computed fields using these functions in the detail band in the workspace.

By default, PowerBuilder places CrosstabSum in the detail band, which returns the total for the corresponding row.

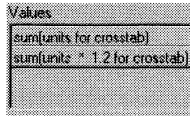
How to specify the functions

Each of these functions takes one numeric argument, which refers to the expression defined for Values in the Crosstab Definition dialog box. The first expression for Values is numbered 1, the second is numbered 2, and so on.

Generally, crosstabs have only one expression for Values, so the argument for the crosstab functions is 1. So, for example, if you defined `sum(units for crosstab)` as your Values expression, PowerBuilder places `CrosstabSum(1)` in the detail band.

But assume you want to cross-tabulate total unit sales and also a projection for future sales, assuming a 20 percent increase in sales (that is, sales that are 1.2 times the actual sales).

You would define two expressions for Values:



Here CrosstabSum(1) returns the total of sum(units for crosstab) for the corresponding row. CrosstabSum(2) returns the total for sum(units * 1.2 for crosstab).

For more information

For complete information about defining computed fields, see Chapter 15, "Enhancing DataWindow Objects".

For more about the crosstab functions, see the *DataWindow Reference*.

Cross-tabulating ranges of values

Often you want to build a crosstab where each row tabulates a *range* of values, instead of one discrete value. Similarly, you might want each column in the crosstab to correspond to a range of values.

For example, in cross-tabulating departmental salary information, you might want one row in the crosstab to count all employees making between \$30,000 and \$40,000, the next row to count all employees making between \$40,000 and \$50,000, and so on.

❖ To cross-tabulate ranges of values:

- 1 Determine the expression that results in the raw values being converted into one of a small set of fixed values.

Each of those values will form a row or column in the crosstab.

- 2 Specify the expression in the Columns or Rows box in the Crosstab Definition dialog box. You choose the box depending on whether you want the columns or rows to correspond to the range of values.
- 3 In the Values column, apply the appropriate aggregate function to the expression.

Example

This is best illustrated with an example.

You want to know how many employees in each department make between \$30,000 and \$40,000, how many make between \$40,000 and \$50,000, how many make between \$50,000 and \$60,000, and so on. To do this, you want a crosstab where each row corresponds to a \$10,000 range of salary.

The first step is to determine the expression that, given a salary, returns the next smaller salary that is a multiple of \$10,000. For example, given a salary of \$34,000, the expression would return \$30,000, and given a salary of \$47,000, the expression would return \$40,000. You can use the Int function to accomplish this, as follows:

```
int(salary/10000) * 10000
```

That expression divides the salary by 10,000 and takes the integer portion, then multiplies the result by 10,000. So for \$34,000, the expression returns \$30,000, as follows:

```
34000/10000 = 3.4  
int(3.4) = 3  
3 * 10000 = 30000
```

With that information you can build the crosstab. The following uses the Employee table in the Powersoft Demo Database:

- 1 Build a crosstab and retrieve the Dept_ID and Salary columns.
- 2 In the Crosstab Definition dialog box, drag the Dept_ID column to the Columns box.
- 3 Drag the Salary column to the Rows box and to the Values box and edit the expressions.

In the Rows box, use:

```
int(salary/10000) * 10000
```

In the Values box, use:

```
count(int(salary/10000) * 10000 for crosstab)
```

FOR INFO For more on providing expressions in a crosstab, see "Using expressions" on page 733.

- 4 Click OK.

This is the resulting workspace:

Number of employees by department and salary		Total number of employees making the salary	
\$30,000 includes up to \$39,999			
Header [1] ↑			
		Department Id	
Salary display(@			
Header [2] ↑			
row_column	val	crosstabsum(1)	
Detail ↑			
Total number of employees in the department		sum(val for	
Summary ↑			
Footer ↑			

And here is the crosstab during execution:

Number of employees by department and salary						Total number of employees making the salary	
\$30,000 includes up to \$39,999							
		Department Id					
Salary:	100	200	300	400	500		
\$20,000				2	5	7	
\$30,000	3	8	2	5	2	20	
\$40,000	6	5	2	5	1	19	
\$50,000	4	3	3	2	1	13	
\$60,000	4	1		2		7	
\$70,000	2	1	1			4	
\$80,000	2	1				3	
\$90,000	1					1	
\$130,000			1			1	
Total number of employees in the department	22	19	9	16	9		

You can see, for example, that two people in department 400 and five in department 500 make between \$20,000 and \$30,000.

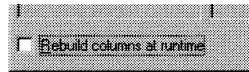
Creating static crosstabs

By default, crosstabs are dynamic: when you run them, PowerBuilder retrieves the data and dynamically builds the columns and rows based on the retrieved data. This is usually what you want. For example, if you have defined a crosstab that computes sales of printers and a new printer type is entered in the database after you defined the crosstab, you want the new printer to be in the crosstab. That is, you want PowerBuilder to dynamically build the rows and columns based on current data, not the data that existed when the crosstab was defined.

Occasionally, however, you might want a crosstab to be static. That is, you want its columns to be established when you define the crosstab. You do not want additional columns to display in the crosstab during execution; no matter what the data looks like, you don't want the number of columns to change. You only want the updated statistics for the predefined columns. Here is how to do that.

❖ To create a static crosstab:

- 1 In the Crosstab Definition dialog box, clear the Rebuild Columns At Runtime checkbox:



- 2 Define the data for the crosstab as usual.
- 3 Click OK.

What happens

With the checkbox cleared, instead of immediately building the crosstab's structure, PowerBuilder first retrieves the data from the database. Using the retrieved data, PowerBuilder then builds the crosstab structure and displays the workspace. It places all the values for the column specified in the Columns box in the workspace. These values become part of the crosstab's definition.

For example, in the following screen, the four values for Quarter (Q1, Q2, Q3, and Q4) are displayed in the workspace:

Sum Of Units	Quarter				
Header[1] ↑					
Product	Q1	Q2	Q3	Q4	Grand Total
Header[2] ↑					
product	units	units_1	units_2	units_3	crosstabsum(1)
Detail ↑					
"Grand Total"	sum(unitssum(unitssum(unitssum(unitssum(grand_sum_units for				
Summary ↑					
Footer ↑					

At execution time, no matter what values are in the database for the column, the crosstab will show only the values that were specified when the crosstab was defined. In the printer example, the crosstab will always have the four columns it had when it was first defined and previewed in the DataWindow painter.

Making changes

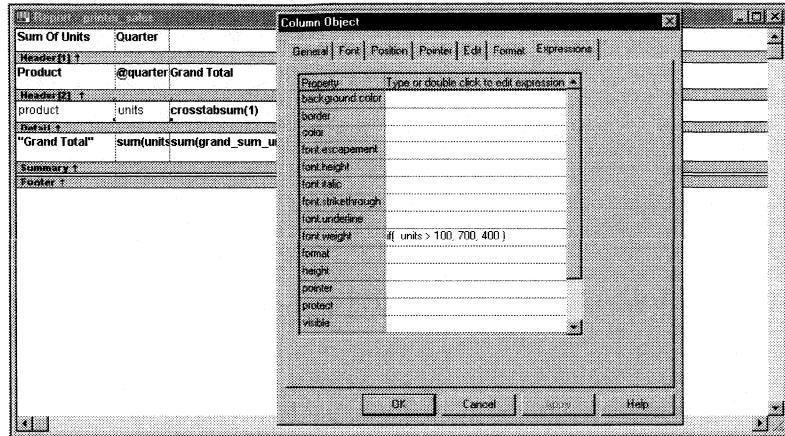
You can modify the properties of any of the columns in a static crosstab. You can modify the properties of each column individually, since each column is displayed in the workspace as part of the crosstab's definition. For example, in the printer crosstab you can directly modify the way values are presented in each individual quarter, since each quarter is represented in the workspace (the values are shown as units, units_1, units_2, and units_3).

Using property conditional expressions

As with other DataWindow objects, you can specify property conditional expressions to modify properties at execution time. You can use them with either dynamic or static crosstabs. With dynamic crosstabs, you specify an expression once for a column or value, and PowerBuilder assigns the appropriate properties when it builds the individual columns during execution. With static crosstabs, you have to specify an expression for each individual column or value because the columns are already specified at definition time.

Example

In the following crosstab, an expression has been specified for Units:



Here is the expression, which is for the Font.Weight attribute:

```
if (units > 100, 700, 400)
```

The expression specifies to use bold font (weight = 700) if Units is greater than 100. Otherwise, use normal font (weight = 400).

Here is the crosstab during execution:

Sum Of Units	Quarter				
Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	60	90	246
Grand Total	118	176	207	262	763

The large values are shown in bold.

FOR INFO For more about property conditional expressions, see Chapter 18, "Highlighting Information in DataWindow Objects".

Working with Rich Text

About this chapter

This chapter explains how to create DataWindow objects using the RichText presentation style.

Contents

Topic	Page
What is rich text?	752
Using the RichText presentation style	753
Using the RichTextEdit control	765
Formatting keys and toolbars	768

What is rich text?

Rich text format (RTF) is a standard for specifying formatting instructions and document content in a single ASCII document. An editor that supports rich text format interprets the formatting instructions and displays the text with formatting. If you look at rich text in a plain ASCII editor, you see complex instructions that are not very readable. The actual text of the document is buried amid all the formatting:

```
{\par}\pard\ql{\f2\fs18\cf0\up0\dn0 A RichText piece of text }
```

The same sample displayed without the commands looks like this:

A RichText piece of text

Elements of rich text

Rich text in PowerBuilder can have:

- ◆ Margins and tab settings for each paragraph
- ◆ Character formatting such as italic, bold, underline, or superscripts for each character
- ◆ Named input fields associated with database columns or other data
- ◆ Bitmaps
- ◆ A header and footer for the document

The user can use toolbars, editing keys, and a popup menu to specify formatting. A print preview lets users view a reduced image of the document to see how it fits on the page.

What is not supported

PowerBuilder supports version 1.2 of the RTF standard, except for the following:

- ◆ No formatted tables
- ◆ No drawing objects
- ◆ No double-underline

Using the RichText presentation style

The RichText presentation style allows you to combine input fields that represent database columns with formatted text. This presentation style is useful for display-only reports, especially mail-merge documents. However, if you want to use the RichText DataWindow object for data entry, you can specify validation rules and display formats for the input fields.

In the design workspace, you see the text along with placeholders called input fields:

```
{FNAME} {LNAME}
{COMPANY_NAME}
{ADDRESS}
{CITY}, {STATE} {ZIP}
```

```
Dear {FNAME}:
```

```
. . .
```

When you preview and print the report, PowerBuilder replaces the input fields with values from the database. In this sample, you see the same text as you saw in the workspace, but now the input fields are replaced with values from the database:

```
Beth Reiser
AMF Corp.
1033 Whippany Road
New York, NY 10154
```

```
Dear Beth:
```

```
. . .
```

Document template

The formatted text acts like a document template. There is only one copy of the text. As the user scrolls from row to row, the data for the current row is inserted in the input fields and the user sees the document with the current data. If the user edits the text, the changes are seen with every row of data.

Input fields

In the RichText presentation style, an input field is associated with a column or computed field. It gets its value from the retrieved data or from the computed field's expression.

If an input field is not a computed field and its name does not match a column, there is no way to specify data for the input field.

There can be more than one copy of an input field in the rich text. In the sample above, there are two instances of the field FNAME. Each instance of the field displays the same data.

Unavailable settings

Not all the settings available in other DataWindow styles are available. You cannot apply code tables and edit styles, such as a DropDownDataWindow or EditMask, to input fields. You cannot use slide left and slide up settings to reposition input fields automatically. However, you can set the LineRemove property during execution to achieve a similar effect.

Creating the DataWindow object

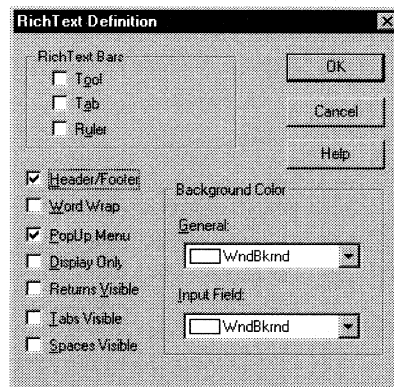
❖ To create a RichText DataWindow object:

- 1 In the DataWindow painter, create a new DataWindow object and select the RichText presentation style.
- 2 Select data for the DataWindow object as you do for any DataWindow object.

The columns become input fields.

- 3 Leave the SQL Select painter and return to the DataWindow painter workspace.

PowerBuilder displays the RichText Definition dialog box:



- 4 Specify settings for the DataWindow object in the RichText Definition dialog box, and click OK.

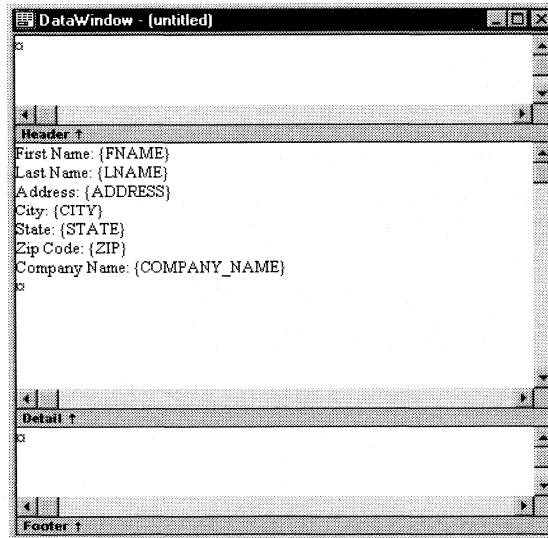
Available settings

This table describes the types of settings you can make for the RichText DataWindow object:

You can specify	With these settings
Tools available to the user	Rich text bars: Tool, Tab, and Ruler, and PopUp Menu
Whether various nonprinting characters are visible	Returns Visible, Tabs Visible, and Spaces Visible
Colors for the whole background and the background of input fields	Background Color: General and Input Field
Whether users are prevented from editing input fields and text	Display Only
Whether newly entered text will wrap within the display	Word Wrap
Whether there will be a header and footer for the printed DataWindow object	Header/Footer

Editing the content

After you click OK in the RichText Definition dialog box, you see input fields with their labels in the detail band of the DataWindow object:



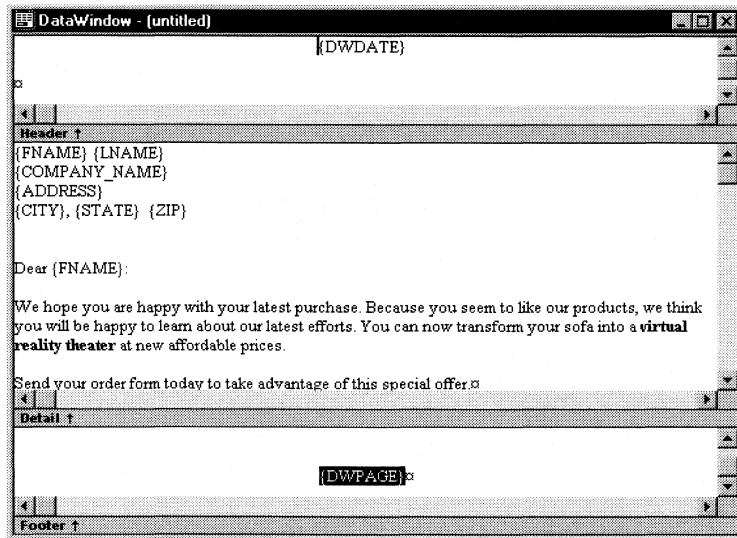
You can:

- ◆ Begin editing text in the detail, header, or footer bands, building a report around the input fields. You can delete, move, copy, and paste text and input fields as needed.
- ◆ Include a rich text file you have already prepared. If you include a rich text file created in PowerBuilder that contains input fields, those names should match the columns selected in the DataWindow object.

FOR INFO For information about creating rich text files, see *Application Techniques*.

- ◆ Add computed fields that will appear as input fields in the report and whose value comes from the computed field expression.

This sample shows how you might rearrange the input fields in a sales letter:



Editing text

You can add text by typing directly in the workspace. You don't have to create text objects as you do for other DataWindow object styles. The DataWindow painter's StyleBar lets you apply formatting to selected text. The RichText toolbars are *not* available in the painter.

If you want to use the RichText toolbars, you can use the PowerBuilder sample application PScript or some other tool to prepare rich text with input fields.

Preview mode and editing text

It may seem convenient to edit text in preview mode, because the toolbars are available. However, *any changes you make to the text when previewing are temporary*. They are discarded as soon as you return to the workspace.

Inserting a file

If you have a rich text file, you can include it in the DataWindow object. You can insert text from a file into the detail, header, or footer band.

❖ To insert a file:

- 1 Click in the text in any band to set the insertion point for the file.
- 2 Right-click in the workspace and select Insert File from the popup menu.
- 3 Select the file you want to insert in the file selection dialog box.

Only the body of the file is used. If the file has a header or footer, it is ignored.

Headers and footers

You decide whether your RichText DataWindow object has a header and footer by checking Header/Footer in the RichText Definition dialog box or Rich Text Object property sheet (described in "Formatting for RichText objects within the DataWindow object" next).

To display a page number or a date in the header or footer, you can insert the predefined computed fields **Page n of n** or **Today()**. You do not need to write scripts to set the values of these fields for each page, as you do for the RichTextEdit control.

Losing header and footer text

If you clear the Header/Footer checkbox, any text in the header and footer is discarded.

Formatting for RichText objects within the DataWindow object

Each type of object in a RichText DataWindow object has its own property sheet. When you select Properties from the popup menu, the property sheet you get depends on what is selected.

Most of the objects in a RichText DataWindow object correspond to familiar objects, like bitmaps, columns, and computed fields. You can also specify formatting for a temporary *selected text object*. In a RichText DataWindow object, the objects are:

- ◆ The whole document
- ◆ Selected text and paragraphs
- ◆ Input fields (associated with columns or computed fields)
- ◆ Pictures

This section describes how to select each type of object and access its property sheet. The user can access the property sheets too if you enable the Popup Menu option on the Rich Text Object's General property sheet.

The whole RichText object

Settings for the whole RichText DataWindow object include:

- ◆ Tools for the user
- ◆ Margins for printing
- ◆ Whether the document has a header and footer
- ◆ Display of nonprinting characters
- ◆ Whether pictures are displayed or represented by empty frames
- ◆ Standard DataWindow object settings such as units of measurement and the pointer

You specified values for some of these in the initial RichText Definition dialog box. Use the following procedure to change settings.

❖ To make general settings for the DataWindow object:

- 1 Make sure nothing is selected in the workspace by clicking to set the insertion point.
- 2 Right-click in the workspace and select Properties from the popup menu.
- 3 Click Help to get more information about a specific setting.

Selected text and paragraphs

You can specify detailed font formatting for selected text. The selected text can be one character or many paragraphs.

If an input field is part of the selection, the font settings apply to it too. A picture that is part of the selection ignores settings for the selected text object.

❖ To specify formatting for selected text:

- 1 Select the text you want to format.
- 2 Right-click in the workspace and select Properties from the popup menu.

The Selected Text Object property sheet displays. You can set:

- ◆ **Paragraph alignment** The alignment setting on the Selected Text page applies to all paragraphs in the selection.
- ◆ **Font formatting** Settings on the Font page apply to text in the selection, including input fields.

Paragraphs

There are also settings for selected paragraphs. You can display the Paragraph dialog box by pressing CTRL+SHIFT+S. The user can double-click the ruler bar or press the key combination to display the same dialog box.

Default font

The user can change the default font by double-clicking on the toolbar or pressing CTRL+SHIFT+D. You can't change the default font in the painter.

Input fields

An input field can be either a column or computed field. Before you retrieve data, its value is shown as two question marks (??).

The text can include many copies of a named input field. The same data will appear in each instance of the input field.

Column input fields

The columns you select for the DataWindow object become input fields in the rich text. Because the input field's name matches the column name, PowerBuilder displays the column's data in the input field.

If an input field exists in the text, you can copy and paste it to create another copy. If you need to recreate a column input field that you deleted, use this procedure.

❖ **To insert a column input field in the text:**

- 1 Select Objects>Column from the menu bar.
- 2 Click in the text where you want the column input field to appear.
PowerBuilder displays a list of the columns selected for the DataWindow object.
- 3 Select a column for the input field.

Properties for input fields

An input field is selected when the whole field is flashing. If there is a solid selection with a flashing insertion point, then the current object is selected text and you'll get a different property sheet.

Editing input field contents

When an input field is flashing, press the space bar to activate the input field and edit the data. In preview mode, you can update the database with your changes.

❖ **To set properties for an input field:**

- 1 Click on the input field so it is flashing.
- 2 Display the popup menu and select Properties.
- 3 On the Font page, specify text formatting.
- 4 On the Format page, specify a display format.
- 5 On the Validation page, specify a validation rule for the column.

If there are multiple copies of an input field, the validation and format settings apply to all the copies. Background color on the Font page applies to all input fields. Other settings on the Font page apply to individual instances.

The user cannot change the format or validation rule. During execution, these pages are not available in the property sheet.

Computed field input fields When you display the property sheet for a computed field, the settings are a little different. You can specify the input field name and its expression on the Compute page and there is no validation.

Data Value in preview For both columns and computed fields, you will see a value in the Data Value box when you preview the DataWindow object. The user will see a value in the Data Value box when the current row has a value. For columns, they can change the value.

Computed fields

Computed fields have an expression that specifies the value of the computed field. In rich text, they are represented as input fields too. You specify a name and an expression. The data value comes from evaluating the expression and cannot be edited.

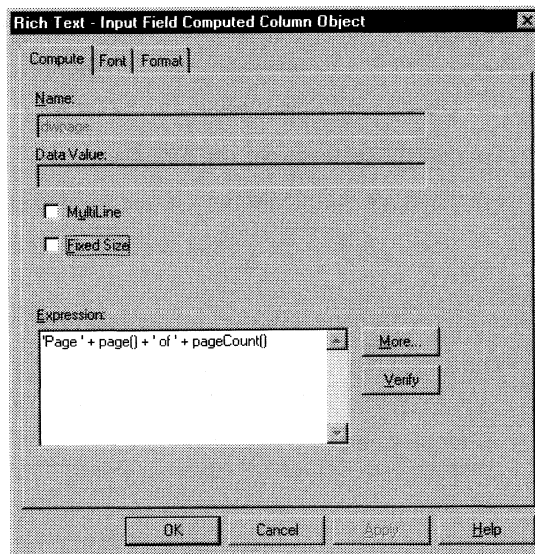
❖ To define a computed field:

- 1 From the Objects menu, select Computed Field or select one of the predefined computed fields at the bottom of the Objects menu.

Predefined computed fields

PowerBuilder provides several predefined computed fields, but in a RichText DataWindow object only the page number (Page n of n) and today's date (Today()) are available.

- 2 Click in the text where you want the computed field to appear.
PowerBuilder displays the property sheet for the computed field:



- 3 On the Compute page, name the computed field and specify its expression.
- 4 (Optional) On the Font page, specify text formatting.
- 5 (Optional) On the Format page, specify a display format.

If there are multiple copies of a computed field input field, the expression and format settings apply to all the copies. Font settings apply to individual instances.

FOR INFO For more about computed field expressions and display formats, see Chapter 15, "Enhancing DataWindow Objects".

Pictures

You can include bitmaps (BMP files) in rich text.

❖ To insert a picture in the rich text:

- 1 Select Objects>Picture from the menu bar.
- 2 Click in the text where you want the picture to appear.
PowerBuilder displays the Select Picture dialog box.
- 3 Select the file containing the picture.

A picture is selected when you can see its nine sizing handles. (If the picture is very small, the handles converge and the selected picture looks black.) When it is part of a text selection, it displays with inverted colors.

❖ To specify settings for the picture:

- 1 Click on the picture so you see its resizing handles.
- 2 Right-click in the workspace and select Properties from the popup menu.

The Bitmap Object property sheet displays.

You can change the size and position of the picture by dragging its handles. You can also change the size using the Bitmap Object dialog box:

Choosing this Picture Size Option	Has this effect
Original size	The bitmap displays at its original size. You can set the x and y offset to make additional space above and to the left of the picture
Resizable	The bitmap fills the picture dimensions
Resizable - Maintain X/Y Ratio	The bitmap fills the picture dimensions as well as it can while maintaining the aspect ratio of the picture

- ❖ **To change the picture's dimensions:**
 - ◆ Drag any handle on the edge of the selected picture.
or
Change the numbers in the Picture Dimension edit boxes on the Bitmap property page.
The picture will fill the frame according to the Picture Size Option you've selected.
- ❖ **To change the picture's position relative to the text baseline:**
 - ◆ Select the picture and drag the handle in the center of the picture up or down.
You can drag the picture above or below the text baseline. You can't drag the picture outside the paragraph. To do that, cut and paste the picture somewhere else in the text.

Previewing and printing

DataWindow object preview You can preview the RichText DataWindow object so you can see what it looks like with data.

- ❖ **To preview the DataWindow object:**
 - ◆ Click the Preview button on the toolbar.
or
Select Preview>Design from the menu bar.
- ❖ **To view the header and footer text:**
 - 1 From the Design menu, select Header/Footer.
 - 2 Select Header/Footer again to go back to the detail band.

Changes in preview mode **Data** While you are previewing the DataWindow object, you can use the scroll buttons in the Preview toolbar to move from row to row and you can change data in the input fields. If you choose the Save Changes button on the toolbar, you will update the data in the database.

Text Any changes you make to the rich text *will be discarded* when you return to the workspace. Any changes that you want to keep must be made in the workspace, not in preview.

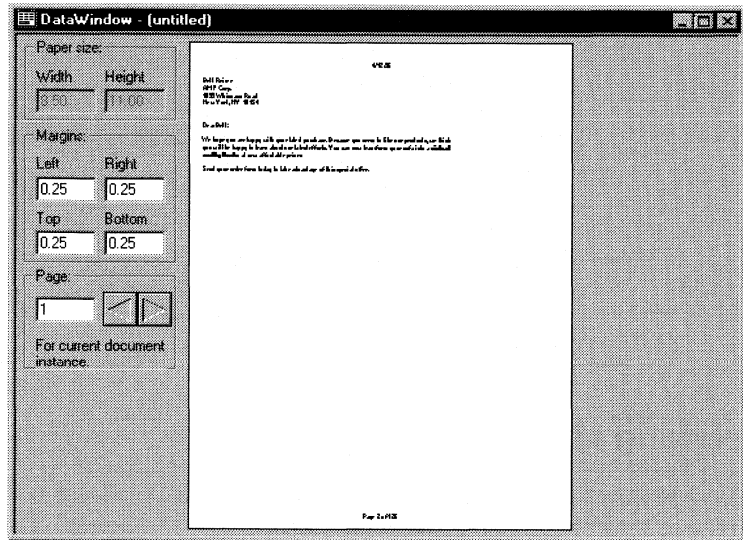
If the Display Only setting is checked, you cannot change text or data in preview mode.

Print Preview

Print Preview displays a reduced view of one row of data as it would be printed.

❖ **To see the DataWindow object in Print Preview:**

- ◆ While in preview mode, select File>Print Preview from the menu bar:



In Print Preview, you can test different margin settings and scroll through the pages of the document.

You *cannot* scroll to view other rows of data.

Any changes you make to settings in Print Preview are discarded when you return to the workspace.

Setting margins

To specify permanent margin settings for the RichText DataWindow object, use the Print Specifications page of the Rich Text Object property sheet.

Using the RichTextEdit control

You can add a RichTextEdit control to a window to enhance your application with word processing capabilities.

Users can enter text in a RichTextEdit control, format it, save it to a file, and print it.

Creating a RichTextEdit control

- ❖ **To add a RichTextEdit control to a window:**
 - ◆ In the Window painter, choose the RichTextEdit control tool and click the window.

Specifying the properties of a RichTextEdit control

You can modify the appearance of a RichTextEdit control by setting its properties. Some of the properties you can set are:

- ◆ The toolbars that appear in the control
- ◆ The visibility of nonprinting characters and graphics

You can also enable a popup menu, from which users can:

- ◆ Control the appearance of the control
- ◆ Import documents

Making a RichTextEdit control read-only

There are times when you may want to import a file into the RichTextEdit control and not give the user the opportunity to alter it. You can make a control read-only by setting the Enabled and Popup Menu properties.

- ❖ **To make a RichTextEdit control read-only:**
 - 1 Choose Properties from the control's popup menu and select the General tab.
The General property page displays.
 - 2 Make sure the Enabled checkbox is cleared.

- 3 Select the Document tab.
The Document property page displays.
- 4 Make sure the Popup Menu checkbox is cleared.

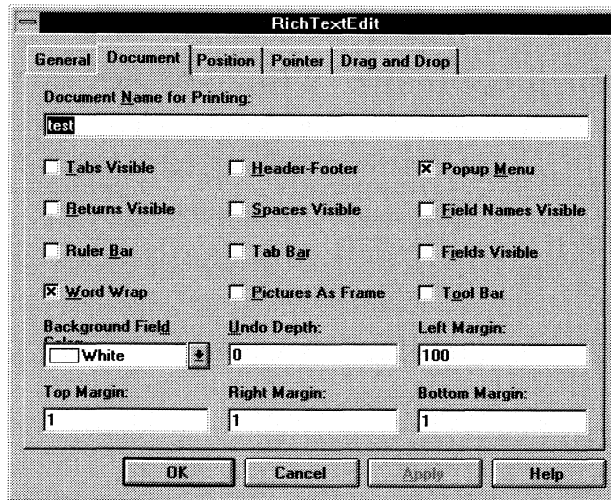
Controlling the appearance of a RichTextEdit control

Use the property sheets to alter the appearance of the RichTextEdit control.

❖ **To control the appearance of a RichTextEdit control:**

- 1 Choose Properties from the control's popup menu and select the Document tab.

The Document property page displays.



- 2 Choose the appropriate properties if you want to display nonprinting characters:
 - ◆ Tabs Visible to display tabs
 - ◆ Returns Visible to display returns
 - ◆ Spaces Visible to display spaces
 - ◆ Field Names Visible to display the names of fields rather than their data
 - ◆ Fields Visible to display field data

3 Choose the appropriate properties to display toolbars:

- ◆ Ruler bar
- ◆ Tab bar
- ◆ Tool bar

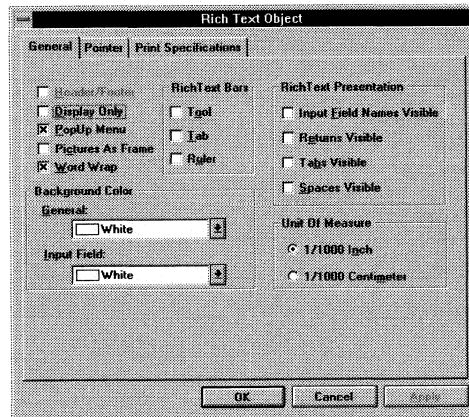
FOR INFO For information about other properties, click the Help button in the RichTextEdit property sheet.

Enabling the popup menu

If you enable the popup menu property, users can customize the appearance of the RichTextEdit control.

From the popup menu, users can:

- ◆ Perform editing tasks (cut, copy, paste, and clear)
- ◆ Insert a file into the RichTextEdit control
- ◆ Display and modify the Rich Text Object property sheet



The General property page on the user's Rich Text Object property sheet corresponds to the Document property page in the development environment.

FOR INFO For more information about the RichTextEdit control, see the chapter on implementing rich text in *Application Techniques*.

Formatting keys and toolbars

When the toolbar is visible, you can use its buttons to format text. The changes you make in preview are temporary.

The keystrokes listed in the following tables also assign formatting to selected text. You can use these in the workspace and save the changes permanently.

On Macintosh

On the Macintosh, use the COMMAND key instead of the CTRL key.

Using the clipboard	Key
Cut	CTRL+X (when there is a selection)
Paste	CTRL+V, SHIFT+INSERT
Copy	CTRL+C
Undo	CTRL+Z

Assigning font attributes	Key
Bold	CTRL+B
Italic	CTRL+I
Underline	CTRL+U
Subscript	CTRL+=
Superscript	CTRL+SHIFT+=
Strikeout	CTRL+K
Change font	CTRL+SHIFT+D or double-click on empty part of toolbar

Setting line spacing	Key
Single space	CTRL+1
Double space	CTRL+2
One and a half space	CTRL+5

Aligning text	Key
Justify	CTRL+J

Aligning text	Key
Center	CTRL+E
Left	CTRL+L
Right	CTRL+R
Set paragraph formatting	CTRL+SHIFT+S or double-click the ruler

Editing	Key
Insert a new paragraph	ENTER
Insert an empty line	CTRL+N
Delete character to right of insertion point	DELETE
Delete character to left of insertion point	BACKSPACE

Input fields	Key
Select the input field at the insertion point	ENTER
Activate the input field at the insertion point	SPACE
When input field is active, accept data and exit field	ENTER
When input field is active, exit field without changing data	ESC
Move to next input field	CTRL+TAB
Move to previous input field	SHIFT+CTRL+TAB

Miscellaneous	Key
Select All	CTRL+A
Print	CTRL+P
Undo	CTRL+Z
Toggle display of nonprinting characters	CTRL+*
Toggle preview mode	CTRL+F2

Navigating and selecting text	Navigating key	Selection key
Move or select a character to the right or left	RIGHT ARROW OR LEFT ARROW	SHIFT+RIGHT ARROW OR SHIFT+LEFT ARROW
Move or select a word to the right or left	CTRL+RIGHT ARROW OR CTRL+LEFT ARROW	CTRL+SHIFT+RIGHT ARROW OR CTRL+SHIFT+LEFT ARROW
Move or select a line up or down	UP ARROW OR DOWN ARROW	SHIFT+UP ARROW OR SHIFT+DOWN ARROW
Move or select to start of line	HOME	SHIFT+HOME
Move or select to end of line	END	SHIFT+END
Move or select to start of document	CTRL+HOME	CTRL+SHIFT+HOME
Move or select to end of document	CTRL+END	CTRL+SHIFT+END
Move to next input field	CTRL+TAB	
Move to previous input field	SHIFT+CTRL+TAB	

Using OLE in a DataWindow Object

About this chapter

This chapter discusses OLE objects in DataWindow objects. You can select OLE as the presentation style when you build a new DataWindow object, or you can mix OLE objects with columns and other DataWindow objects. The procedures in this chapter explain how to use OLE objects and specify their behavior.

The behavior of OLE objects in DataWindow objects is similar to the behavior of OLE controls in windows.

FOR INFO For more information about linked and embedded objects and automation, see the chapter on using OLE in an application in *Application Techniques*.

Platform note

OLE is supported on the Windows platform only.

Contents

Topic	Page
OLE support in DataWindow objects	772
OLE objects and OLE presentation style	774
Using OLE columns in a DataWindow object	785

OLE support in DataWindow objects

A DataWindow object can include an object that is a container for an OLE object. The container stores information about the application that created the object and it can launch the application to display or modify the OLE object.

The container can fill the whole DataWindow object, when you create a new DataWindow object using the OLE presentation style, or it can exist alongside other objects in a DataWindow object, when you add an OLE object to an existing DataWindow object. You can also read OLE data from a blob column in a database and display the objects in the DataWindow object. These mechanisms support both OLE 2.0 and OLE 1.0 servers.

You can use OLE objects in DataWindow objects in the following ways:

- ◆ **OLE object in a DataWindow object** The OLE object is displayed in its container object with the DataWindow data and other objects, such as bitmaps or text. You can associate it with data in a particular row, the rows on a page, or with all rows. You choose which columns in the DataWindow object are transferred to the OLE object. You can add an OLE container object to a DataWindow object that uses any presentation style that supports multiple DataWindow objects (this does not include the graph and RichTextEdit presentation styles).
- ◆ **OLE presentation style** The OLE presentation style is similar to an OLE object in a DataWindow object. The difference is that the OLE container is *the only* object in the DataWindow object. The underlying data is *not* presented in column objects and *there are no other* objects, such as bitmaps or text. The OLE object is *always* associated with all the rows in the DataWindow object.
- ◆ **OLE database blob column** OLE objects that are stored in the database in a blob column are displayed in each row of the DataWindow object.

You can also add OLE custom controls (OCXs) to DataWindow objects. OCXs range from simple visual displays, such as meters and clocks, to more complex controls that perform spellchecking or image processing.

About activation

When you're working in the DataWindow painter, you can start the server application for an OLE object by double-clicking the object. The OLE object is activated in place if the server supports in-place activation. Once the server application has started, you can use the tools provided by the server to edit the initial presentation of the object.

In-place or offsite activation

When an OLE object is activated in place, the menus and toolbars of the server application merge with or replace those of the application from which it was launched. If the server application does not support in-place activation, it is activated offsite, which means that the server application opens and the object becomes an open document in the server's window.

If the OLE object is associated with *all rows* retrieved and is in the foreground or background layer, not the band layer, users can activate the object. If the object is associated with a single row or page or is in the band layer, users can see the object but can't activate it. DataWindow Objects created using the OLE presentation style are *always* associated with all rows.

Unlike OLE objects, OCXs are always active. They do not contain objects that need to be opened or activated by double-clicking.

Editing OLE objects

When an OLE object is activated, you can edit the presentation of the data. Changes made to DataWindow data affect the OLE object. Changes made to the OLE object *do not* affect the data the DataWindow object retrieved.

Each OLE object stored in the database in a blob column can be activated and changed. When the DataWindow object updates the database, the changes are saved.

What's next

Whether you are inserting an OLE object into an existing DataWindow object or using the OLE presentation style, you use the same procedures to define, preview, and specify data for the OLE object. Because of their similarities, the next section discusses both OLE objects in existing DataWindow objects and the OLE presentation style. The last section discusses OLE database blob columns.

OLE objects and OLE presentation style

The OLE object in a DataWindow object and the OLE presentation style are very similar. Both are OLE container objects within the DataWindow object. They have these characteristics in common:

- ◆ **Icon or contents** The DataWindow object can display the OLE object as an icon or it can display an image of the contents when display of contents is supported by the server.
- ◆ **Data from the DataWindow object** You specify which DataWindow columns you want to transfer to the OLE object. The data that is sent to the OLE server replaces the OLE object template specified in the painter.

The OLE object in a DataWindow object and the OLE presentation style have these main differences:

- ◆ **Associating the object with rows** When the OLE object is added in the DataWindow painter, it can be associated with individual rows, groups of rows, or all rows. In the presentation style, the OLE object is associated with all rows.
- ◆ **Property sheet** The property sheet for the OLE object has no settings for the DataWindow object because the DataWindow object has its own property sheet. The property sheet for the OLE presentation style includes settings for the DataWindow object and the OLE container object.

Not all servers are appropriate

The features of the OLE server application determine whether it can provide useful information in a DataWindow object.

If the server doesn't support display of contents, it won't be useful for objects associated with rows. The user will only see the icon. Some servers support the display of contents, but the view is scaled too small to be readable even when the object is activated.

In this section

This section includes procedures for:

- ◆ Adding an OLE object to a DataWindow object
- ◆ Using the OLE presentation style
- ◆ Defining the OLE object
- ◆ Previewing the DataWindow object
- ◆ Specifying DataWindow data for the OLE object

Adding an OLE object to the DataWindow object

The procedures for adding OLE objects and OLE custom controls to a DataWindow object are similar. Both exist within the DataWindow object with other objects, such as columns, computed fields, and text objects.

You use the first procedure in this section whether you want to add an OLE object or an OCX to an existing DataWindow object. The procedure opens the Insert Object dialog box in which you define the OLE object.

❖ To place an OLE object in a DataWindow object:

- 1 Open the DataWindow painter and select the DataWindow object that will contain the OLE object.

The DataWindow painter workspace displays the DataWindow object.

- 2 From the toolbar, click the Object dropdown arrow and select the OLE button (not OLE Database Blob).

or

Select Objects>OLE Object from the menu bar.

- 3 Click where you want the OLE object.

PowerBuilder displays the Insert Object dialog box.

FOR INFO To use the Insert Object dialog box, see "Defining the OLE object" on page 776.

Using the OLE presentation style

Use the OLE presentation style to create a DataWindow object that consists of a single OLE object. The following procedure creates the new DataWindow object and opens the Insert Object dialog box.

❖ **To create a new DataWindow object using the OLE presentation style:**

- 1 Open the DataWindow painter and select New in the Select DataWindow dialog box.

The New DataWindow dialog box displays.

- 2 Select a data source and the OLE presentation style, then click OK.

You are prompted to specify the data.

- 3 Specify the data you want retrieved into the DataWindow object.

FOR INFO For more information about selecting data, see Chapter 14, "Defining DataWindow Objects".

- 4 Click OK.

PowerBuilder displays the Insert Object dialog box in which you define the OLE object.

FOR INFO To use the Insert Object dialog box, see "Defining the OLE object" next.

Defining the OLE object

You define the OLE object in the Insert Object dialog box. It has three tab pages:

If you want to	Select
Embed an OLE server object in the DataWindow object	Create New
Link or embed the contents of an existing file as an OLE object so that it can be activated using the application that created it	Create From File
Insert an OCX in the DataWindow object	Insert Control

This section contains procedures for each of these selections.

Create New

Use the following procedure if you want to embed a new OLE server object.

❖ **To embed a new OLE server object using the Create New tab:**

- 1 Select the Create New tab.
- 2 In the Object Type box, highlight the OLE server you want to use.
You can click Browse to get information about the server from the registry.
- 3 Optionally display the OLE object as an icon by doing one of the following:
 - ◆ Check Display as Icon to display the server application's default icon in the control.
 - ◆ Check Display as Icon and then select Change Icon to supply a nondefault icon and icon label.

- 4 Click OK.

The OLE object is inserted in your document and the OLE server is activated (in place if the server supports in-place activation). You can edit the object now. However, this object only provides the initial presentation of the OLE object. The presentation is replaced by data from the DataWindow object when you preview the DataWindow object and at execution time.

- 5 Click outside the hatched border of the OLE object to deactivate it.

The DataWindow painter menus replace the OLE server's menus and PowerBuilder displays the property sheet for the OLE object with the Data property page selected.

- 6 On the Data property page, specify how the OLE object will use the DataWindow object's data.

You can drag the columns you want the OLE object to use to the Target Data box. You can also edit columns and control the grouping of data. If you want to, you can set or change these specifications later.

FOR INFO For more information, see "Specifying data for the OLE object" on page 781.

- 7 If you are inserting an OLE object in an existing DataWindow object and you want to associate the object with the current row, click the Position tab and change the value in the Layer box to Band.

You cannot change this value to Band if you are using the OLE presentation style.

- 8 Click OK when you have finished.

The OLE object is either empty or contains the initial presentation.

Create From File

Use the following procedure if you want to link or embed the contents of an existing file as an OLE object so that it can be activated using the application that created it. Most of the steps in this procedure are the same as those for embedding a new OLE server object.

❖ **To link or embed an existing object using the Create From File tab:**

- 1 Select the Create From File tab.
- 2 Specify the filename in the File Name box. If you do not know the name of the file, click the Browse button and select a file in the dialog box.
- 3 To create a link to the file, rather than embed a copy of the object in the control, select the Link checkbox.
- 4 Go to step 3 in the procedure for the Create New tab (above) to continue.

Insert Control

Use the following procedure if you want to insert an OLE custom control (OCX) in the DataWindow object.

❖ **To insert an OCX using the Insert Control tab:**

- 1 In the Control Type box, highlight the OCX you want to use.
To get information about the selected OCX, you can click Browse. OCXs are self-documenting. PowerBuilder gets the property, event, and function information from the OCX itself via the registry.
- 2 If the OCX you want has not been registered, click Register New.
You will be prompted for the file that contains the registration information for the OCX.
- 3 Click OK.
PowerBuilder displays the property sheet for the control.
- 4 Change the control properties if desired and click OK.
PowerBuilder displays the property sheet for the OLE object with the Data property page selected.

- 5 On the Data property page, specify how the OLE object will use the DataWindow object's data.

If you have inserted an OCX that does not display data, such as the Clock control, you don't need to transfer data to it.

FOR INFO For more information, see "Specifying data for the OLE object" on page 781.

- 6 Click OK when you have finished.

Previewing the DataWindow object

Previewing the DataWindow object lets you see how the OLE object displays the data from the DataWindow object.

❖ To preview the DataWindow object with the OLE object:

- 1 Click the Preview toolbar button.

or

Select Design>Preview from the menu bar.

The DataWindow object retrieves rows from the database and replaces the initial presentation of the OLE object with an image of the data that the OLE server provides.

Tip

If you have previewed already, you may need to click the Retrieve button to retrieve the rows again. This resets the data being transferred to the OLE object.

- 2 If you associated the OLE object with all rows, activate the OLE object by double-clicking on it.

The OLE object displays with a hatched border and the server menus appear on the menu bar. Although you can edit the presentation or the data in the server, your changes do not affect the DataWindow object's data.

You can't always activate the OLE object

If the OLE object is associated with individual rows in the detail band or with the page, you (and the user) cannot activate it; you can only view it.

- 3 Click outside the hatched border to deactivate the OLE object.

- 4 Return to design mode by clicking the Preview toolbar button.
or
Select Display>Design from the menu bar.

Activating and editing the OLE object

In the painter

PowerBuilder stores an initial presentation of the OLE object that it displays before data is retrieved and in newly inserted rows. When you activate the OLE object in the painter, you are editing the initial presentation of the OLE object. Any changes you make and save affect only this initial presentation. After rows are retrieved and data transferred to the OLE object, an object built using the data replaces the initial presentation.

In preview or at execution time

PowerBuilder displays the initial presentation of the OLE object while it is retrieving rows and then replaces it with the retrieved data. After you activate the OLE object in preview or at execution time, you can edit the presentation of the data. However, you cannot save these changes. The object is recreated whenever data from retrieved rows is transferred to the OLE object.

You can save the object with its data by saving the DataWindow object as a Powersoft report (PSR).

FOR INFO For more information, see "About activation" on page 773.

❖ To activate the OLE object in the container in the painter:

- ◆ Double-click the object.

or

Select Open from the container's popup menu.

When you double-click an OCX, it displays its property sheet (which is different from the DataWindow property sheet for the container). OCXs are always active.

Changing the object in the control

In the DataWindow painter, you can change or remove the OLE object in the OLE container object.

❖ To delete the OLE object in the container:

- ◆ Select Delete from the container's popup menu.

The container object is now empty and cannot be activated.

❖ To change the OLE object in the container:

- 1 Select Insert from the container's popup menu.
PowerBuilder displays the Insert Object dialog box.
- 2 Choose one of the tabs and specify the type of object you want to insert, as you did when you defined the object.
- 3 Click OK.

Specifying data for the OLE object

You specify data for the OLE object on the Data property page in the property sheet for the container.

When you insert a new object, just after you deactivate the object, PowerBuilder displays the property sheet with the Data property page selected. If you want to change the settings later, use the following procedure.

❖ To change the data settings:

- ◆ Select Properties from the container's popup menu and select the Data tab.

What the data is for

When an OLE object is part of a DataWindow object, you can specify that some or all of the data the DataWindow object retrieves be transferred to the OLE object too. You can specify expressions instead of the actual columns so that the data is grouped, aggregated, or processed in some way before being transferred.

The way the OLE object uses the data depends on the server. For example, data transferred to Microsoft Excel is displayed as a spreadsheet. Data transferred to Microsoft Graph populates its datasheet, which becomes the data being graphed. Some OCXs don't display data, so you would not transfer any data to them. For an OCX such as Visual Speller, you would use automation to process text when the user requests it.

Data property page

Three boxes on the Data property page list data columns or expressions. This table explains what they are for:

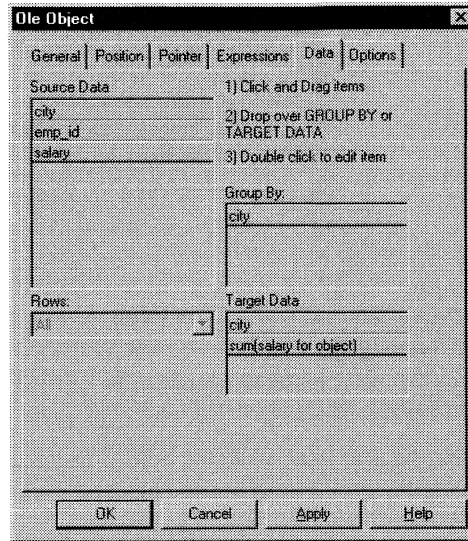
Data property page item	Contents and purpose
Source Data	Lists the columns retrieved in the DataWindow object You can't change the source data
Group By	Specifies how PowerBuilder groups the data it transfers to the OLE object Aggregation functions in the target data expressions use the groupings specified here To populate the Group By box, drag columns to it from Source Data or Target Data. The same source column can appear in both Group By and Target Data
Target Data	Specifies the data that you want to transfer to the OLE object To populate the Target Data box, drag columns to it from Source Data or Group By If there is something in the Group By box, PowerBuilder automatically converts each new column you drag to the Target Data box to an aggregate expression, unless the new column is also in the Group By box The order of the columns and expressions is important to the OLE server. You need to know how the server will use the data to choose the order An aggregation function uses the range <i>for object</i> . The expression will have a value for each group (or combination of groups) in the Group By box

On the Data property page, you can also:

- ◆ Change the order of the grouping or the target data by dragging an item up or down within its box.
- ◆ Edit an item by double-clicking on it in the Target Data or Group By box. In the Modify Expression dialog box, you can edit the expression or use the Functions or Columns boxes and the operator buttons to select elements of the expression.

Example of a completed Data property page

This example of the Data property page specifies two columns to transfer to Microsoft Graph: city and salary. Graph expects the first column to be the categories and the second column to be the data values. The second column is an aggregate because the city column is in the Group By box:



Specifying a value for Rows

The last setting on the Data property page specifies how the OLE object is associated with rows in the DataWindow object. The selection (all rows, current row, or page) usually corresponds with the band where you placed the OLE object, as explained in this table. If you used the OLE presentation style to create the DataWindow object, you cannot change this setting: the OLE object is always associated with all the rows in the DataWindow object.

Range of rows	When to use it
All	<p>When the OLE object is in the summary, header, or footer band, or the foreground or background layer. For the OLE presentation style, All is the only choice</p> <p>Rows must be All and Layer must be Foreground or Background if you want the user to be able to activate the object</p> <p>Target data for all rows is transferred to the object</p>

Range of rows	When to use it
Current Row	When the OLE object is in the detail band There is an instance of the OLE object for every row. Target data for a single row is transferred to each object Because OCXs must be in the foreground or background layer, they cannot be associated with individual rows in the detail band
Page	When the OLE object is in the group header or trailer, foreground, or background Target data for the rows on the current page is transferred to the OLE object

Range of rows and activating the object

When the range of rows is Current Row or Page, the user *cannot* activate the OLE object. The user can see contents of the object in the form of an image presented by the server but cannot activate it.

If you want the user to activate the object, Rows must be set to All and Layer on the Position property page must be Foreground or Background.

Additional settings on the property sheet

The Options property page for the OLE object's property sheet has some additional settings. They are similar to the settings you can make for the OLE control in a window. This table describes the settings you can make:

Options property page setting	Effect
Client Name	A name for the OLE object that some server applications use in the title bar of their window Corresponds to the ClientName DataWindow property
Contents	Whether the object in the OLE container is linked or embedded. The default is Any, which allows either method

Options property page setting	Effect
Activation	<p>How the OLE object is activated. Choices are:</p> <ul style="list-style-type: none"> ◆ Double click When the user double-clicks on the object, the server application is activated ◆ Manual The object can only be activated programmatically <p>The object can always be activated programmatically, regardless of the Activation setting</p>
Display Type	<p>What the OLE container displays. You can choose:</p> <ul style="list-style-type: none"> ◆ Manual Display a representation of the object, reduced to fit within the container ◆ Icon Display the icon associated with the data. This is usually an icon provided by the server application
Link Update	<p>When the object in the OLE container is linked, the method for updating link information. Choices are:</p> <ul style="list-style-type: none"> ◆ Automatic If the link is broken and PowerBuilder cannot find the linked file, it displays a dialog box in which the user can specify the file ◆ Manual If the link is broken, the object cannot be activated <p>You can let the user re-establish the link in a script using the UpdateLinksDialog function</p>

Using OLE columns in a DataWindow object

You can create OLE columns in a DataWindow object. An OLE column allows you to:

- ◆ Store blob (binary large-object) data, such as Excel worksheets or Word for Windows documents, in the database
- ◆ Retrieve blob data from a database into a DataWindow object
- ◆ Use an OLE server application, such as Microsoft Excel or Word for Windows, to modify the data
- ◆ Store the modified data back in the database

You can modify the document in the server, then update the data in the DataWindow object. When the database is updated, the OLE column, which contains the modified document, is stored in the database.

Database support for OLE columns

If your database supports a blob data type, then you can implement OLE columns in a DataWindow object. The name of the data type that supports blob data varies.

FOR INFO For information on which data types your DBMS supports, see your DBMS documentation.

Creating an OLE column

This section describes how to create an OLE column in a DataWindow object. The steps are illustrated using a table named ole in the Powersoft Demo Database. It contains three columns: id, object, and description:

- ◆ The id column is an integer and serves as the table's key
- ◆ The object column is a blob data type and contains OLE objects associated with several OLE servers
- ◆ The description column has a brief description of the object in each row

Creating the database table

For this sample procedure, you can use the ole table in the Powersoft Demo Database or you can create your own table. Follow this procedure to create your own table.

❖ To create the database table:

- 1 In the Database painter, create a table to hold the blob (binary large-object) data. The table must have at least two columns: a key column and a column with the blob data type.

The actual data type you choose depends on your DBMS. For example, in SQL Anywhere, choose long binary as the data type for the blob column. In SQL Server, choose Image.

FOR INFO For information about data types, see your DBMS documentation.

- 2 Define the blob columns as allowing NULLs (this allows you to store a row that doesn't contain a blob).

Adding a blob column to the DataWindow object

This procedure describes how to add a blob column to a DataWindow object.

❖ **To add a blob column to a new DataWindow object:**

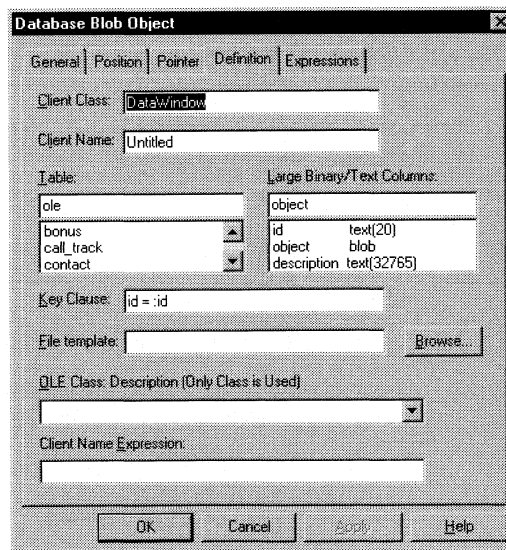
- 1 Open the DataWindow painter and create a new DataWindow object.
- 2 Specify the table containing the blob as the data source for the DataWindow object.

Be sure to include the key column, but you cannot include the blob column in the data source. If you try, a message tells you that its data type is not supported. You will add the blob column later in the DataWindow painter workspace. (If you use Quick Select, the blob column is not listed in the dialog box.)

In the example, you must choose the id column. Choosing the description column is optional.

- 3 Add the blob column to the DataWindow object by selecting OLE Database Blob from the Objects menu and clicking the place in the DataWindow object where you want the blob object.

The Blob Object property sheet displays with the Definition property page selected:



Setting properties for the blob column

This procedure describes the properties you need to set for the blob column.

❖ **To set properties for a blob column:**

- 1 (Optional) Enter the client class in the Client Class box. The default is DataWindow.

This value is used in some OLE server applications to build the title that displays at the top of the server window.

- 2 (Optional) Enter the client name in the Client Name box. The default is Untitled.

This value is used in some OLE server applications to build the title that displays in the title bar of the server window.

- 3 In the Table box, select the database table that contains the blob database column you want to place in the DataWindow object.

The names of the columns in the selected table display in the Large Binary/Text Columns listbox.

- 4 In the Large Binary/Text Columns box, select the column that contains the blob data type from the list.

- 5 If necessary, change the default key clause in the Key Clause box.

PowerBuilder uses the key clause to build the WHERE clause of the SELECT statement used to retrieve and update the blob column in the database. It can be any valid WHERE clause.

Use colon variables to specify DataWindow columns. For example, if you enter this key clause:

```
id = :id
```

the WHERE clause will be:

```
WHERE id = :id
```

- 6 Identify the OLE server application by doing one of the following:

- ◆ If you always want to open the same file in the OLE server application, enter the name of the file in the File Template box.

For example, to specify a particular Word for Windows document, enter the name of the DOC file. If the file is not on the current path, enter the fully qualified name.

Use the Browse button to find the file

If you do not know the name of the file you want to use, click the Browse button to display a list of available files. Select the file you want from the resulting window.

- ◆ If you do not want to open the same file each time, select an OLE server application from the OLE Class: Description dropdown listbox.

If your OLE server application does not appear in this list, run the Windows RegEdit utility to add it.

FOR INFO For more information about RegEdit, see the Windows online Help REGEDIT.HLP and REGEDITV.HLP.

When the server doesn't match the OLE blob data

If you specify a server that doesn't match the OLE blob object or if your database contains objects belonging to different servers, the OLE mechanism can usually handle the situation. It looks for the server specified in the object and starts it instead of the server you specified.

- 7 Enter text or an expression that evaluates to a string in the Client Name Expression box.

The server may use this expression in the title of the window in the OLE server application. The expression you specify can identify the current row in the DataWindow object.

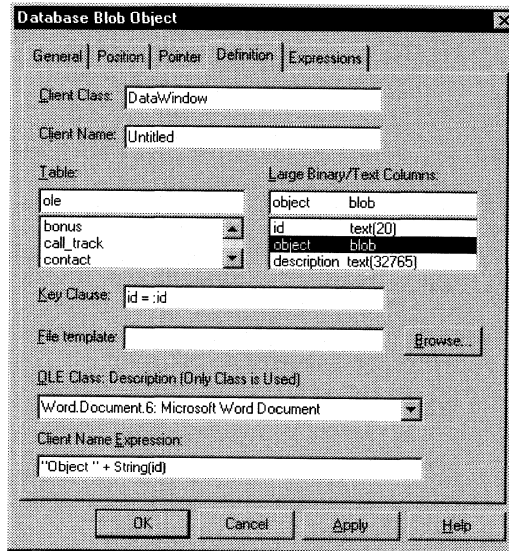
Use an expression to make sure the name is unique

Use an expression to make sure the name is unique. For example, you might enter the following expression to identify a document (where ID is the integer key column):

```
"Document " + String(id)
```

- 8 (Optional) Select the General tab and enter a name for the object in the Name box. You don't need to name the object, but doing so allows you to refer to it in scripts.
- 9 Click OK.
PowerBuilder closes the Database Blob Object dialog box and displays the DataWindow painter workspace. The blob column is represented by a box labeled Blob.
- 10 Save the DataWindow object.

Here is the completed Database Blob Object dialog box for the example:



Making the blob column visible for OLE 1.0 servers

For OLE 1.0 servers, the blob column is invisible in the DataWindow object until you activate the OLE server. To make it easy to find the blob column, you can place a drawing object behind the blob object. When the DataWindow object displays, the drawing object will indicate the location of the blob column until the user double-clicks the column to open the server application.

The drawing object is not necessary for OLE 2.0 servers, which provide an icon for the object.

Previewing an OLE column

Before using the DataWindow object in an application, you should preview it to see how it works.

❖ **To preview an OLE column:**

- 1 Click the Preview button
or
Select Preview from the Design menu.
- 2 Click the Insert Row button.
PowerBuilder adds a blank row.

Previewing the ole table in the Powersoft Demo Database

If you are previewing the ole table, there are already several rows in the table. You do not have to insert another.

3 In the blank row, enter a value in the key column.

4 Double-click the column that contains the blob data type.

The OLE server application starts and displays the file you specified in the File Template box or an empty workspace if you only specified the OLE server name.

5 Review the file in the OLE server application and make changes if you want.

When you use an OLE column to access an OLE server application, the server application adds an item to its File menu that allows you to update the data in the server application and in the client (the DataWindow object). The text of the menu item depends on the OLE server application. In most applications, it is Update.

6 Select the menu item in the OLE server that updates the OLE client with the modifications.

In the example, you would select Update from the File menu in Word for Windows.

The OLE server application sends the updated information to the DataWindow object.

7 Close the file in the server application (typically by selecting Close from the File menu).

8 Return to the DataWindow painter.

The updated blob is represented in the DataWindow object using an icon for the OLE server application.

9 To save the blob data in the database, click the Save Changes button.

The new row, including the key value and the blob, are stored in the database.

Later, after you retrieve the rows from the database, you can view and edit the blob by double-clicking it, which invokes the OLE server application and opens the stored document. If you make changes and then update the database, all the modified OLE columns are stored in the database.

About this chapter

This chapter describes how to use the Data Pipeline painter to create data pipelines, which let you reproduce database data in various ways.

Contents

Topic	Page
About data pipelines	794
Creating a data pipeline	798
Modifying the data pipeline definition	801
Correcting pipeline errors	811
Saving a pipeline	813
Using an existing pipeline	814
Pipeline examples	815

About data pipelines

The Data Pipeline painter gives you the ability to quickly reproduce data within a database, across databases, or even across DBMSs. To do that, you create a data pipeline which, when executed, **pipes** the data as specified in the definition of the data pipeline.

What you can do

With the Data Pipeline painter, you can perform some tasks that would otherwise be very time consuming. For example, you can:

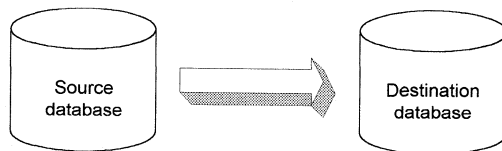
- ◆ Pipe data (and extended attributes) from one or more tables to a table in the same DBMS or a different DBMS
- ◆ Pipe an entire database, a table at a time, to another DBMS (and if needed, pipe the database's repository tables)
- ◆ Create a table with the same design as an existing table but with no data
- ◆ Pipe corporate data from a database server to a SQL Anywhere database on your computer so you can work on the data and report on it without needing access to the network
- ◆ Upload local data that changes daily to a corporate database
- ◆ Create a new table when a change (such as allowing or disallowing NULLs or changing primary key or index assignments) is disallowed in the Table painter

Piping data in applications

You can also create applications that pipe data. For more information, see *Application Techniques*.

Source and destination databases

You can use the Data Pipeline painter to pipe data from *one or more* tables in a **source database** to *one* table in a **destination database**:



You can pipe all data or selected data in one or more tables. For example, you can pipe a few columns of data from one table or data selected from a multitable join. You can also pipe a view.

When you pipe data, the data in the source database remains in the source database and is reproduced in a new or existing table in the destination database.

Although the source and destination can be the same database, they are usually different ones, and they can even have different DBMSs. For example, you can pipe data from a SQL Server database to a SQL Anywhere database on your computer.

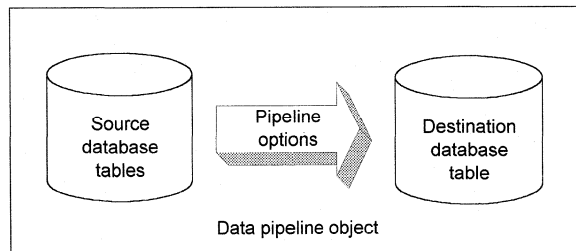
Defining a data pipeline

When you use the Data Pipeline painter to create a pipeline, you define:

- ◆ Source database
- ◆ Destination database
- ◆ Source tables to access and the data to retrieve from them
- ◆ Pipeline operation
- ◆ Destination table

Once you define it

After you create a pipeline, you can execute it immediately. If you want, you can also save it as a named object to use and reuse:



Saving a pipeline enables you to pipe the data that may have changed since the last pipeline execution or to pipe the data to other databases later.

Data type support

Each DBMS has certain data types it supports. When you pipe data from one DBMS to a different DBMS, PowerBuilder makes its best guess for the destination data types. You can correct PowerBuilder's best guess in your pipeline definition as needed.

The Data Pipeline painter supports the piping of columns of any data type, including columns with blob data.

FOR INFO For information about piping a column that has a blob data type, see "Piping blob data" on page 808.

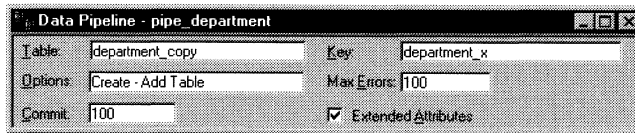
Piping extended attributes

The first time PowerBuilder or InfoMaker connects to a database, it creates five Powersoft system tables, called the **Powersoft repository**. These system tables initially contain default extended attribute information for tables and columns. In PowerBuilder or InfoMaker, you can create extended attribute definitions, such as column headers and labels, edit styles, display formats, and validation rules.

FOR INFO For more information about the repository, see the Appendix, "The Powersoft Repository".

Piping extended attributes automatically

When you pipe data, you can specify that you want to pipe the extended attributes associated with the columns you are piping. You do this by selecting the Extended Attributes checkbox in the Data Pipeline painter workspace:



When the Extended Attributes checkbox is selected, the extended attributes associated with the source database's selected columns automatically go into the repository of the destination database, with the following exception.

When extended attributes are not automatically piped

When you pipe data that has an edit style, display format, or validation rule associated with it, the style, rule, or format will *not* be piped if one with the same name exists in the repository of the destination database. In this situation, the data uses the style, rule, or format already present in the destination database.

For example, for the Phone column in the Employee table, the display format with the name Phone_format would be piped unless a display format with the name Phone_format already exists in the destination database. If such a display format exists, the Phone column would use the Phone_format display format in the destination database.

Piping the Powersoft repository

Selecting the Extended Attributes checkbox *never* results in the piping of named display formats, edit styles, and validation rules that are stored in the repository but independent of data. If you want such extended attribute definitions from one database to exist in another database, you can pipe the appropriate Powersoft repository table or a selected row or rows from the table.

If you want to reproduce an entire database, you can pipe all database tables and repository tables, one table at a time.

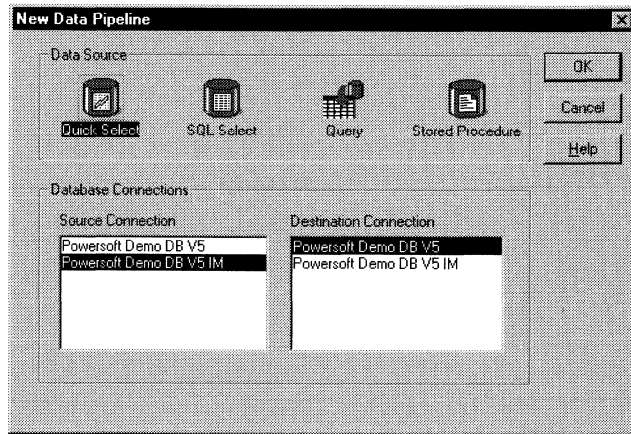
Creating a data pipeline

You have a number of choices when creating a data pipeline. This section walks you through them.

❖ **To create a data pipeline:**

- 1 Click the Pipeline button in the PowerBar.
The Select Data Pipeline dialog box displays.
- 2 Click New.

The New Data Pipeline dialog box displays:



The Source Connection and Destination Connection boxes display database profiles that have been defined. The last database you connected to is selected as the source. The first database on the destination list is selected as the destination.

If you don't see the connections you need

To create a pipeline, the databases you want to use for your source and destination must each have a database profile defined. If you don't see profiles for the databases you want to use, cancel the New Data Pipeline dialog box, then define those profiles.

FOR INFO For information about defining profiles, see "Changing the destination and source databases" on page 809.

3 Select a data source.

The data source determines how PowerBuilder retrieves data when you execute a pipeline:

Data source	Use it if
Quick Select	The data is from tables that are connected through a key and you only need to sort and limit data
SQL Select	You want more control over the SQL SELECT statement generated for the data source or your data is from tables that are not connected through a key
Query	The data has been defined as a query
Stored Procedure	The data is defined in a stored procedure

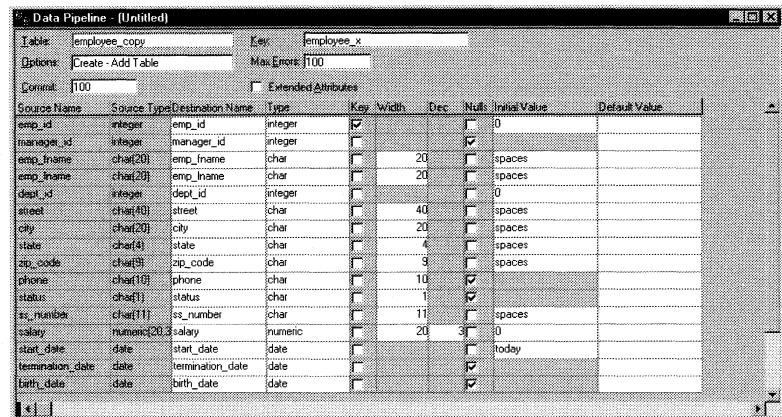
4 Select the source and destination connections and click OK.

5 Define the data to pipe.

How you do this depends on what data source you chose in step 3.

FOR INFO For complete information about using each data source and defining the data, see Chapter 14, "Defining DataWindow Objects".

When you finish defining the data to pipe, the Data Pipeline painter workspace displays the pipeline definition, which includes a pipeline operation, a checkbox for specifying whether to pipe extended attributes, and source and destination items:



The pipeline definition is PowerBuilder's best guess based on the source data you specified.

- 6 Modify the pipeline definition as needed.

FOR INFO For information, see "Modifying the data pipeline definition" next.

- 7 (Optional) Modify the source data as needed:

- ◆ Click the Edit SQL button.

or

Select Design>Edit Data Source from the menu bar.

FOR INFO For information about working in the Select painter, see Chapter 14, "Defining DataWindow Objects".

When you return to the Data Pipeline painter workspace, PowerBuilder reminds you that the pipeline definition will change. Click OK to accept the definition change.

- 8 If you want to try the pipeline now, click the Execute button.

or

Select Design>Execute from the menu bar.

PowerBuilder retrieves the source data and executes the pipeline. If you specified retrieval arguments in the Select painter, PowerBuilder first prompts you to supply them.

During execution, the number of rows read and written, the elapsed execution time, and the number of errors display in MicroHelp. You can stop execution yourself or PowerBuilder may stop execution when errors occur.

FOR INFO For information about execution and how rows are committed to the destination table, see "When execution stops" on page 805.

- 9 Save the pipeline definition if appropriate.

FOR INFO For information, see "Saving a pipeline" on page 813.

Seeing the results of piping data

You can see the results of piping data by connecting to the destination database and opening the destination table.

Modifying the data pipeline definition

After you create a pipeline definition, you can modify it in a variety of ways. This depends on what pipeline operation you select, the destination DBMS, and what you are trying to accomplish by executing the pipeline.

Items you can modify in the workspace

At the top of the workspace These items apply to the destination table:

The screenshot shows a dialog box titled "Data Pipeline - (Untitled)". It contains several input fields and a checkbox. The "Table" field contains "employee_copy", the "Key" field contains "employee_x", the "Options" dropdown is set to "Create - Add Table", the "Max Errors" field contains "100", and the "Commit" field contains "100". There is an unchecked checkbox labeled "Extended Attributes".

Item	Description	Default	How to edit
Table	Name of the destination table	If source and destination are different, name of first table specified in the source data or name of the stored procedure; if the same, <i>_copy</i> is appended	For Create or Replace, enter a name For Refresh, Append, or Update, select a name from the dropdown listbox
Options	Pipeline operation: Create, Replace, Refresh, Append, or Update	Create - Add Table	Select an option from the dropdown listbox
Commit	Number of rows piped to the destination database before PowerBuilder commits the rows to the database	100 rows	Select a number, <i>All</i> , or <i>None</i> from the dropdown listbox
Key	Key name for the table in the destination database	If the source is only one table, the table name is followed by <i>_x</i>	For Create or Replace, enter a name. Not editable for other pipeline operations
Max Errors	Number of errors allowed before the pipeline stops	100 errors	Select a number or <i>No Limit</i> from the dropdown listbox
Extended Attributes	For Create and Replace, a checkbox that specifies whether or not the extended attributes of the selected source columns are piped to the repository of the destination database. Does not display for Refresh, Append, or Update	Not checked	Click the checkbox

At the bottom left of the workspace These items show the source column names and data types. They are not editable, because you specified them as the data source:

Source Name	Source Type	Destination Name	Type	Key	Width	Dec	Nulls	Initial Value	Default Value
emp_id	integer	emp_id	integer	<input checked="" type="checkbox"/>			<input type="checkbox"/>	0	
manager_id	integer	manager_id	integer	<input checked="" type="checkbox"/>			<input type="checkbox"/>		
emp_name	char(20)	emp_name	char	<input type="checkbox"/>	20		<input type="checkbox"/>	spaces	
emp_name	char(20)	emp_name	char	<input type="checkbox"/>	20		<input type="checkbox"/>	spaces	
dept_id	integer	dept_id	integer	<input type="checkbox"/>			<input type="checkbox"/>	0	
street	char(40)	street	char	<input type="checkbox"/>	40		<input type="checkbox"/>	spaces	
city	char(20)	city	char	<input type="checkbox"/>	20		<input type="checkbox"/>	spaces	
state	char(4)	state	char	<input type="checkbox"/>	4		<input type="checkbox"/>	spaces	
zip_code	char(9)	zip_code	char	<input type="checkbox"/>	9		<input type="checkbox"/>	spaces	
phone	char(10)	phone	char	<input type="checkbox"/>	10		<input type="checkbox"/>		
status	char(1)	status	char	<input type="checkbox"/>	1		<input checked="" type="checkbox"/>		
ss_number	char(11)	ss_number	char	<input type="checkbox"/>	11		<input type="checkbox"/>	spaces	
salary	numeric(20)	salary	numeric	<input type="checkbox"/>	20	3	<input type="checkbox"/>	0	
start_date	date	start_date	date	<input type="checkbox"/>			<input type="checkbox"/>	today	
termination_date	date	termination_date	date	<input type="checkbox"/>			<input type="checkbox"/>		
birth_date	date	birth_date	date	<input type="checkbox"/>			<input type="checkbox"/>		

At the bottom right of the workspace These items apply to the destination table's columns and key. They are mostly editable only for the Create and Replace pipeline operations:

Item	Description	Default	How to edit
Destination Name	Column name	Source column name	Enter a name
Type	Column data type	If the DBMS is unchanged, source column data type. If the DBMS is different, a best-guess data type	Select a type from the dropdown listbox
Key	Whether the column is a key column (check means yes)	Source table's key columns (if the source is only one table and all key columns are selected)	Select or clear checkboxes
Width	Column width	Source column width	Enter a number
Dec	Decimal places for the column	Source column decimal places	Enter a number
Nulls	Whether NULL is allowed for the column (check means yes)	Source column value	Select or clear checkboxes
Initial Value	Column initial value	Source column initial value (if no initial value, character columns default to <i>spaces</i> and numeric columns default to 0)	Select an initial value from the dropdown listbox

Item	Description	Default	How to edit
Default Value	Column default value	Source column default value that's assigned in the database	Select a default value from the dropdown listbox or enter a default value. Keyword values depend on destination DBMS

Choosing a pipeline operation

When PowerBuilder pipes data, what happens in the destination database depends on which pipeline operation you choose in the Options dropdown listbox:

When you choose this pipeline operation	This happens in the destination database
Create - Add Table	A new table is created and rows selected from the source tables are inserted If a table with the specified name already exists in the destination database, a message displays and you must select another option or change the table name
Replace - Drop/Add Table	An existing table with the specified table name is dropped, a new table is created, and rows selected from the source tables are inserted If no table exists with the specified name, a table is created
Refresh - Delete/Insert Rows	All rows of data in an existing table are deleted, and rows selected from the source tables are inserted
Append - Insert Rows	All rows of data in an existing table are preserved, and new rows selected from the source tables are inserted
Update - Update/Insert Rows	Rows in an existing table that match the key criteria values in the rows selected from the source tables are updated, and rows that don't match the key criteria values are inserted

Dependency of modifications on pipeline operation

The modifications you can make in the workspace depend on the pipeline operation you have chosen.

When using
Create - Add Table or
Replace - Drop/Add
Table

When you select the Create option (the default) or the Replace option, you can change these items:

You can	Comment
Change the destination table definition	Follow the rules of the destination DBMS
Have both a key name and key columns or neither	Specify key columns by selecting one or more checkboxes to define a unique identifier for rows
Allow or disallow NULL for a column	If NULL is allowed (checkbox selected), no initial value is allowed If NULL is not allowed, an initial value is required. The words <i>spaces</i> (a string filled with spaces) and <i>today</i> (today's date) are initial value keywords
Modify the Commit and Max Errors values	
Specify an initial value and a default value	

If you have specified key columns and a key name and if the destination DBMS supports primary keys, the Data Pipeline painter creates a primary key for the destination table. If the destination DBMS does not support primary keys, a unique index is created.

For Oracle databases

PowerBuilder generates a unique index for Oracle databases.

If you try to use the Create option, but a table with the specified name already exists in the destination database, PowerBuilder tells you and you must select another option or change the table name.

When you use the Replace option, PowerBuilder warns you that you are deleting a table, and you can choose another option if needed.

When using Refresh
- Delete/Insert Rows
or Append - Insert
Rows

For the Refresh and Append options, the destination table already exists. You can:

- ◆ Select an existing table from the dropdown listbox in the Table box
- ◆ Modify the Commit and Max Errors values
- ◆ Change the initial value for a column

When using Update -
Update/Insert Rows

For the Update option, the destination table already exists. You can:

- ◆ Select an existing table from the dropdown listbox in the Table box
- ◆ Modify the Commit and Max Errors values
- ◆ Change the Key columns in the destination table's key (primary key or unique index, depending on what the DBMS supports), but key columns must be selected

The key determines the UPDATE statement's WHERE clause.

- ◆ Change the initial value for a column

Bind variables and the Update option

If the destination database supports bind variables, the Update option takes advantage of them to optimize pipeline execution.

When execution stops

Execution of a pipeline can stop for any of these reasons:

- ◆ You click the Cancel button
During the execution of a pipeline, the Execute button in the PainterBar changes to the Cancel button.
- ◆ The error limit is reached

If there are rows that can't be piped to the destination table for some reason, those error rows display once execution stops. You can correct error rows or return to the workspace to change the pipeline definition and then execute it again.

FOR INFO For information, see "Correcting pipeline errors" on page 811.

Whether rows are committed

When rows are piped to the destination table, they are first inserted and then either committed or rolled back. Whether rows are committed depends on:

- ◆ The Commit and Max Errors values
- ◆ When errors occur during execution
- ◆ Whether you click the Cancel button or PowerBuilder stops execution

When you stop execution

If the Commit value is	Then when you click Cancel
A number n	Each row that was piped is committed
<i>All or None</i>	Each row that was piped is rolled back

For example, if you click the Cancel button when the 24th row is piped and if the Commit value is 20, then:

- ◆ 20 rows are piped and committed
- ◆ 3 rows are piped and committed
- ◆ Piping stops

If the Commit value were *All or None*, 23 rows would be rolled back.

When PowerBuilder stops execution

If the Commit value is	And the Max Errors value is	Then when PowerBuilder stops execution because the error limit is reached
A number n	<i>No limit</i> or a number m	Rows are piped and committed n rows at a time until the Max Errors value is reached
<i>All or None</i>	<i>No limit</i>	Each row that pipes without error is committed
<i>All or None</i>	A number n	If the number of errors is less than n , all rows are committed If the number of errors is equal to n , each row that was piped is rolled back. No changes are made

For example, if an error occurs when the 24th row is piped and the Commit value is 10 and the Max Errors value is 1, then:

- ◆ 10 rows are piped and committed
- ◆ 10 rows are piped and committed
- ◆ 3 rows are piped and committed
- ◆ Piping stops

If the Commit value were *All* or *None*, 23 rows would be rolled back.

About transactions

A transaction is a logical unit of work done by a DBMS, within which either all the work in the unit must get done or none of the work in the unit must get done. If the destination DBMS doesn't support transactions or isn't connected in transaction mode, each row that is inserted or updated is committed.

About the All and None commit values

In the Data Pipeline painter, the Commit values *All* and *None* have the same meaning.

In PowerBuilder applications, you may execute more than one pipeline at a time in databases that support transaction processing. Such applications need either all piped rows to be committed or none to be committed (all piped rows rolled back). The *None* commit value is most useful for the application execution environment.

A PowerBuilder application can issue commits and rollbacks in pipeline scripts.

Piping blob data

Blob data is data having a data type that specifies the data as *binary large-objects* such as a Microsoft Word documents or Excel spreadsheets. A data pipeline can pipe columns containing blob data.

The name of the data type that supports blob data varies by DBMS, for example:

DBMS	Data types that support blob data
SQL Anywhere	LONG BINARY LONG VARCHAR (if more than 32 KB)
Sybase SQL Server	IMAGE TEXT
Microsoft SQL Server	IMAGE TEXT
Oracle	RAW LONG RAW
Informix	BYTE TEXT

FOR INFO For information about the data type that supports blob data in your DBMS, see your DBMS documentation.

Adding blob columns to a pipeline definition

When you select data to pipe, you cannot select a blob column as part of the data source because blobs can't be handled in a SELECT statement. After the pipeline definition is created, you add blob columns, one at a time, to the definition.

❖ **To add a blob column to a pipeline definition:**

- 1 Select Design>Database Blob from the menu bar.

If the Database Blob menu item is grayed

The Database Blob menu item is grayed if the pipeline definition doesn't contain a unique key for at least one source table, or if the pipeline operation is Refresh, Append, or Update and the destination table has no blob columns or you've associated all destination blob columns with source blob columns.

The Database Binary/Text Large Object property sheet displays.

The Table box has a dropdown list of tables in the source database that have a primary key and contain blob columns.

- 2 In the Table box, select the table that contains the blob column you want to add to the pipeline definition.
For example, in the Powersoft Demo Database, the ole table contains a blob column named Object with the large binary data type.
- 3 In the Large Binary/Text Column box, select a column that has a blob data type.
- 4 In the Destination Column box, change the name of the destination column for the blob if you want.
If you want to add the column and see changes you make without closing the dialog box, click Apply after each change.
- 5 When you have specified the blob source and destination as needed, click OK

❖ **To edit the source or destination name of the blob column in the pipeline definition:**

- ◆ Display the blob column's popup menu and select Properties.

❖ **To delete a blob column from the pipeline definition:**

- ◆ Display the blob column's popup menu and select Clear.

Executing a pipeline with blob columns

After you've completed the pipeline definition by adding one or more blob columns, you can execute the pipeline. When you do, rows are piped a block at a time, depending on the Commit value. For a given block, Row 1 is inserted, then Row 1 is updated with Blob 1, then Row 1 is updated with Blob 2, and so on. Then Row 2 is inserted, and so on until the block is complete.

If a row is not successfully piped, the blob is not piped. Blob errors display, but the blob itself does not display. When you correct a row in error and execute the pipeline, the pipeline pipes the blob.

Changing the destination and source databases

Changing the destination

When you create a pipeline, you can change the destination database. And if you want to pipe the same data to more than one destination, you can change the destination database again and re-execute.

❖ **To change the destination database:**

- ◆ Click the Destination button in the PainterBar.
or
Select File>Destination Connect from the menu bar.

Changing the source Normally you don't change the source database, because your pipeline definition is dependent on it. But if you need to (maybe because you are no longer connected to that source) you can.

❖ **To change the source database:**

- ◆ Select File>Source Connect from the menu bar.

Working with database profiles

At any time in the Data Pipeline painter, you can edit an existing database profile or create a new one.

❖ **To edit or create a database profile:**

- ◆ Click the Database Profile button in the PainterBar and then click the Edit button or the New button.

FOR INFO For information about how to edit or define a database profile, see *Connecting to Your Database*.

Correcting pipeline errors

If the pipeline can't pipe certain rows to the destination table for some reason, PowerBuilder displays the error rows:

Error Message	emp_id	manager_id	emp_name	emp_name	dept_id
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 102	501		Fran	Whitney	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 105	501		Matthew	Cobb	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 129	902		Philip	Chan	200
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 148	1293		Julio	Jordan	300
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 160	501		Robert	Breault	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 184	1576		Melissa	Espinosa	400
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 191	703		Jeanette	Bertrand	500
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 195	902		Marc	Dill	200
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 207	1576		Jane	Francis	400
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 243	501		Natasha	Shishov	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 247	501		Kurt	Discoll	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 249	501		Rodrigo	Guevara	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 266	501		Ram	Gowda	100
SQLSTATE = 23000 [Sybase][ODBC Driver][Integer] 278	501		Terry	Melkisetan	100

Source Name	Source Type	Destination Name	Type	Key	Width	Dec	Null	Initial Value	Default Value
emp_id	integer	emp_id	integer	<input checked="" type="checkbox"/>			<input type="checkbox"/>	0	

What you see

PowerBuilder shows:

- ◆ Name of the table in the destination database
- ◆ Pipeline operation you chose in the Option box
- ◆ Error messages to identify the problem with each row
- ◆ Data values in the error rows
- ◆ Source and destination column information

What you can do

You can correct the error rows by changing one or more of their column values so the destination table will accept them; or you can ignore the error rows and return to the Data Pipeline painter workspace. If you return to the workspace, you cannot redisplay the error rows without re-executing the pipeline.

Before you return to the workspace

You may want to print the list of errors or save them in a file. Select File>Print or File>Save As from the menu bar.

- ❖ **To return to the Data Pipeline painter workspace without correcting errors:**
 - ◆ Click the Design button.

❖ **To correct pipeline errors:**

- 1 Change data values for the appropriate columns in the error rows.
- 2 Click the Update DB button.

or

Select Design>Update Database from the menu bar.

PowerBuilder pipes rows in which errors were corrected to the destination table and displays any remaining errors.

- 3 Repeat steps 1 and 2 until all errors are corrected.

The Data Pipeline painter workspace displays.

Viewing an error message

Sometimes you cannot see an entire error message because the column isn't wide enough:

Error Message	emp_id	emp_fname	emp_lname	dept_name	salary
SQLSTATE = 23000 [WATCOM]ODBC Driver [WAT102		Fran	Whitney	R & D	45700.000
SQLSTATE = 23000 [WATCOM]ODBC Driver [WAT103		Mark	Morris	R & D	20000.000
SQLSTATE = 23000 [WATCOM]ODBC Driver [WAT104		Trolan	Ovenbey	R & D	20000.000

❖ **To view an error message**

- ◆ Move the pointer to the error message and press the RIGHT ARROW key to scroll through it.

or

Drag the Error Message column border to the width needed:

Error Message	emp_id
SQLSTATE = 23000 [WATCOM]ODBC Driver [WATCOM SQL] integrity constraint violation: primary key for table 'employee' is not unique	102
SQLSTATE = 23000 [WATCOM]ODBC Driver [WATCOM SQL] integrity constraint violation: primary key for table 'employee' is not unique	103
SQLSTATE = 23000 [WATCOM]ODBC Driver [WATCOM SQL] integrity constraint violation: primary key for table 'employee' is not unique	104

Making the error messages shorter

For ODBC data sources, you can set the DBParm MsgTerse parameter in the destination database profile to make the error messages shorter. If you type:

```
MsgTerse = 'Yes'
```

then the SQLSTATE error number won't appear.

FOR INFO For more information, see *Connecting to Your Database*.

Saving a pipeline

When you have generated a pipeline definition in the Data Pipeline painter workspace, you should save the pipeline. You can then reuse it later.

❖ **To save a pipeline:**

- ◆ Click the Save button
- or*
- ◆ Select File>Save from the menu bar.

For a new pipeline

When you save a pipeline for the first time:

- ◆ Specify a name (any valid identifier up to 40 characters).
- ◆ Specify the library to save it in.

Using an existing pipeline

If you save a pipeline, you can modify and execute it any time. You can also pipe data that may have changed since the last pipeline execution or pipe data to other databases.

❖ **To use an existing pipeline:**

- 1 Click the Pipeline button in the PowerBar.

The Select Data Pipeline dialog box displays. Pipelines in the current library (PBL) are listed. If you do not see the pipeline you want, select another library.

- 2 Select the pipeline you want to execute.
- 3 Click the Design button.

The Data Pipeline painter workspace displays.

- 4 If you want to change the pipeline operation, select a new option.
- 5 Modify the pipeline definition as needed.
- 6 Execute and/or save the pipeline.

Pipeline examples

Updating data in a destination table

You may want to pipe data and then update the data often.

❖ **To update a destination table:**

- 1 Click the Pipeline button, select an existing pipeline that you executed before, and click OK.

The pipeline definition displays. Since this pipeline has been executed before, the table exists in the destination database.

- 2 Select the Update option in the pipeline definition.
- 3 Execute the pipeline.

The destination table is updated with current data from the source database.

Reproducing a table definition with no data

You can force a pipeline to create a table definition and not pipe data. It's easiest to do this using SQL Select as the data source.

❖ **To reproduce a table definition with no data:**

- 1 Click the Pipeline button, click New, select SQL Select as the data source and specify the source and destination databases, and click OK.
- 2 In the Select painter, open the table you want to reproduce and select all columns.
- 3 In the Where tab, type an expression that will never evaluate to true, such as $1 = 2$.
- 4 Click the SQL Select button to create the pipeline definition.
- 5 Select the Extended Attributes checkbox.
- 6 Click the Execute button to execute the pipeline.

The table definition is piped to the destination database, but no rows of data are piped. You can open the new table in the Database painter and then click the Grid, Table, or Freeform button to view the data. As specified, there is no data.

If you don't use SQL Select If you use a data source other than SQL Select, you can reproduce a table definition by creating the pipeline definition, editing the data source (which opens the Select painter), and then typing the expression that will never evaluate to true in the Where tab.

Piping a table to many databases

In the Data Pipeline painter workspace, you can execute a pipeline many times with a different destination database each time.

❖ **To pipe a table to many databases:**

- 1 Select File>Destination Connect from the menu bar to change the destination to the database you want.
- 2 Execute the pipeline.
- 3 Repeat steps 1 and 2 for each database you want.

Running Your Application

This part describes the ways in which your application can be run. The first chapter describes how to run your application from within PowerBuilder: in debug mode, where you can set breakpoints and examine the state of your application as it executes, and in regular mode, where the application runs until you stop it or an error occurs, and where you can collect trace information so that you can analyze performance and evaluate your application's structure. The second chapter describes how to build your application for distribution to users.

Debugging and Running Applications

About this chapter

This chapter describes how to debug and run an application in PowerBuilder. It also lists the errors that can occur at execution time.

Contents

Topic	Page
Overview of debugging and running applications	820
Debugging an application	821
Running an application	844
Tracing and profiling an application	851

Overview of debugging and running applications

After you build all or part of an application and compile and save its objects, you can run the application. The PowerBuilder development environment provides two ways to run an application: in debug mode and in regular mode.

Debug mode

In **debug mode**, you can insert breakpoints (stops) in scripts and functions, single-step through code, and display the contents of variables to locate logic errors and mistakes that will result in errors during execution.

Regular mode

In **regular mode**, the application responds to user interaction and runs until the user stops it or until an execution-time error occurs. This is the mode you and your users will use to run the completed application. You can also collect trace information while you run your application in regular mode.

This chapter describes:

- ◆ Running applications in debug mode
- ◆ Running applications in regular mode

Debugging an application

Sometimes an application doesn't behave the way you think it will. Perhaps a variable is not being assigned the value you expect, or a script doesn't do what you want it to. In these situations, you can examine your application by running it in debug mode.

When you run the application in debug mode, PowerBuilder stops execution before it executes a line containing a **breakpoint** (stop). You can then step through the application and examine its state.

❖ To debug an application:

- 1 Open the Debug window.
- 2 Set breakpoints at places in the application where you have a problem.
- 3 Run the application in debug mode.
- 4 When execution is suspended at a breakpoint, look at the values of variables, the properties of objects in memory, and the call stack, or change the values of variables.
- 5 Step through the code line by line if you want.
- 6 As needed, add or modify breakpoints as you run the application.
- 7 When you uncover a problem, fix your code.

Using the Debug window

The Debug window contains several **views**. Each view shows a different kind of information about the current state of your application or the debugging session. The following table summarizes what each view shows and what you can do from that view:

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

View	What it shows	What you can do
Call Stack	The sequence of function calls leading up to the function that was executing at the time of the breakpoint, shown as the script and line number from which the function was called	Examine the context of the application at any line in the call stack
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 descendant 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

Opening the Debug window

If the application you want to debug is not the current application, open the application you want to debug before you open the Debug window.

❖ **To open the Debug window:**

- 1 Click the Debug button in the PowerBar or PowerPanel.

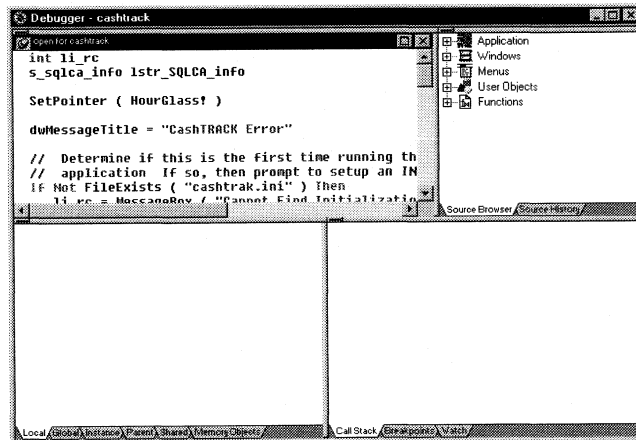
If you are in a painter with unsaved work, you are asked whether you want to save the object.

- 2 If necessary, save your work.

The PowerBuilder Debug window opens.

Managing the Debug window

Views are displayed in **panes**. When you first open the Debug window, there are two rows of panes. Most panes are overlapped and have tabs you can click to pop the pane to the top of the stack. Each pane shows a different view. All the available views are included in the default layout, including five different Variables views, one for each variable type.



Each pane has these features:

- ◆ A title bar you can display transiently 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

You can change the layout of the Debug window to suit your needs, and you can save different layouts for different debugging tasks. This section tells you how to show title bars, add additional panes, resize, move, overlap, or delete existing panes, and save customized layouts.

Displaying title bars

When you are debugging, the kind of information displayed in each pane and your familiarity with the layout of the Debug window usually makes title bars unnecessary, but you can display a title bar for any pane either transiently or permanently. The default layout displays the title bar for the Source view permanently because it shows the name of the script being displayed.

❖ **To display a title bar for a Debug window pane:**

- 1 Place the cursor on the splitter bar at the top of the pane.
The cursor changes to an arrow with a bar and the title bar displays.
- 2 (Optional) Click the pushpin at the left of the title bar.
The title bar remains visible when you move the cursor. Click the pushpin again to hide the title bar.

Moving a pane

You can move a pane to any location in the Debug window.

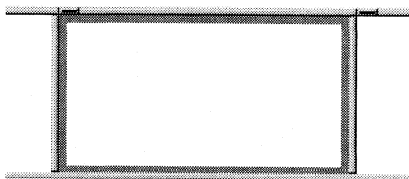
❖ **To move a Debug window pane:**

- 1 Place the cursor on the handle in the top-left corner of the pane and hold down the left mouse button.

Moving tabbed panes

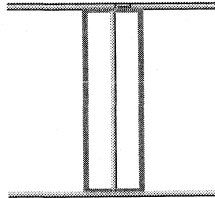
If you move tabbed panes using the handle in the top-left corner of the top pane, *all* the panes in the stack move. To move one of the panes out of the stack, drag the tab for the pane you want to move instead of the handle.

A gray outline appears in the pane.



- 2 Drag the outline to the new location.

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



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

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

Resizing panes

You can resize panes by dragging the splitter bars between panes. To maximize a pane, display the title bar and click the Maximize button.

Adding and removing panes

You may want to add additional views to the Debug window. For example, you may want more than one Source view. If there are some views you rarely use, you can move them into an overlapped stack or delete them.

❖ To add a new pane to the Debug window:

- 1 Select the view you want in the pane from the View menu.

The new pane displays as a new row.

- 2 Move the pane where you want it.

FOR INFO For how to move panes, see "Moving a pane" on page 824.

❖ **To remove a pane from the Debug window:**

- ◆ Select Close from the pane's popup menu.

or

Display the pane's title bar if it is not already visible and click the Close button.

Closing tabbed panes

Clicking the Close button in a stack of tabbed panes closes *all* the panes.

Selecting Close from the popup menu closes only the uppermost pane.

Changing Variable views

The default Debug window layout contains a separate pane for each variable type. You can combine two or more Variables views in a single pane.

❖ **To display multiple Variable views in a single pane:**

- 1 Display the popup menu for a pane that contains a Variables view you want to change.
- 2 Click the names of the variable types you want to display.

A checkmark displays next to selected variable types. The popup menu closes each time you select a variable type or clear a checkmark so you will need to reopen the menu to select additional variable types.

When you select or clear variable types, the tab for the pane changes to show the variable types displayed on that pane.

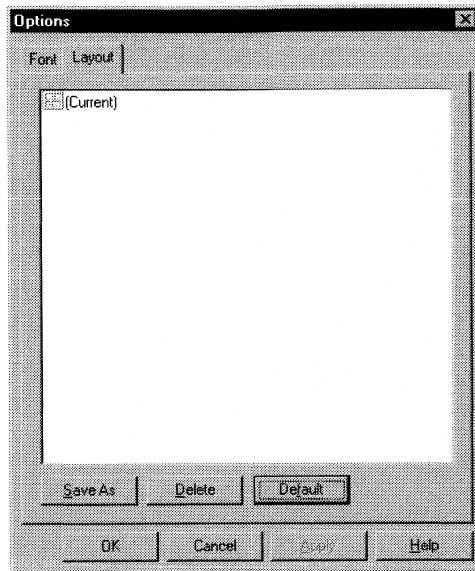


Saving a layout

When you have rearranged panes in the Debug window during a debugging session, PowerBuilder saves the layout in your PowerBuilder initialization file and restores it the next time you open the Debug window. You can also save customized layouts yourself so that you can switch from one layout to another for different kinds of debugging activities.

❖ **To save customized layouts for the Debug window:**

- 1 Select Debug > Options and select the Layout tab.



- 2 Click the SaveAs button.

The last name in the list, labeled Current, becomes editable.

- 3 Type an appropriate name and click OK.

Restoring the default layout

You can restore the default layout at any time by clicking the Default button on the Layout tab.

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 Debug window, any breakpoints you set are written to your PowerBuilder initialization file and are available when you reopen the Debug window.

Setting a simple breakpoint

❖ To set a breakpoint on a line in a script:

- 1 Display the script in a Source view and place your cursor where you want to set the breakpoint.

FOR INFO For how to change the script shown in the Source view, see "Using the Source view" on page 837.

- 2 Double-click the line.

or

Select Add Breakpoint from the popup menu.

PowerBuilder sets a breakpoint and a red circle displays at the beginning of the line. If you select a line that doesn't contain executable code, PowerBuilder sets the breakpoint at the beginning of the next executable statement.

Setting special breakpoints

Breakpoints can be triggered when a statement has been executed a specified number of times (an occasional breakpoint), when a specified expression is true (a conditional breakpoint), or when the value of a variable changes.

Opening the Edit Breakpoints dialog box

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.

❖ To open the Edit Breakpoints dialog box:

- ◆ Click the Edit Stop button on the PainterBar.

or

Select Breakpoints from the popup menu in the Source, Variables, Watch, or Breakpoints view.

or

Select Edit > Breakpoints from the menu bar.

or

Double-click a line in the Breakpoints view.

Setting occasional and conditional breakpoints

If you want to check the progress of a loop without interrupting execution in every iteration, you can set an occasional breakpoint that is triggered only after a specified number of iterations. To specify that execution stops only when conditions you specify are met, set a conditional breakpoint. You can also set both occasional and conditional breakpoints at the same location.

- ◆ **If you specify an occurrence** Each time PowerBuilder passes through the specified location it increments a counter by one. When the counter reaches the number specified, it triggers the breakpoint and resets the counter to zero.
- ◆ **If you specify a condition** Each time PowerBuilder passes through the specified location it evaluates the expression. When the expression is true, it triggers the breakpoint.
- ◆ **If you specify both an occurrence and a condition** Each time PowerBuilder passes through the specified location it evaluates the expression. When the expression is true, it increments the counter. When the counter reaches the number specified, it triggers the breakpoint and resets the counter to zero.

For example, if you specify an occurrence of 3 and the condition **notisNull(val)**, PowerBuilder checks whether val is NULL each time the statement is reached. The breakpoint is triggered on the third occurrence of a non-NULL val, then again on the sixth occurrence, and so forth.

❖ To set an occasional or conditional breakpoint:

- 1 On the Location tab in the Edit Breakpoints dialog box, specify the script and line number where you want the breakpoint.

You can select an existing location or select New to enter a new location.

Set a simple breakpoint first

You must specify a line that contains executable code. Set a simple breakpoint on the line before opening the Edit Breakpoints dialog box to ensure the format and line number are correct.

- 2 Specify an integer occurrence, a condition, or both.

The condition must be a valid boolean PowerScript expression. PowerBuilder displays the breakpoint expression in the Edit Breakpoints dialog box and in the Breakpoints view. When PowerBuilder reaches the location where the breakpoint is set, it evaluates the breakpoint expression and triggers the breakpoint only when the expression is true.

Setting a breakpoint when a variable changes

You can interrupt execution every time the value of a variable changes. The variable must be in scope when you set the breakpoint.

❖ To set a breakpoint when a variable changes:

- ◆ Select the variable in the Variables view or Watch view and select Break on Change from the popup menu.

or

Drag the variable from the Variables view or Watch view to the Breakpoints view.

or

Select New on the Variable tab in the Edit Breakpoints dialog box and specify the name of a variable in the Variable textbox.

The new breakpoint displays in the Breakpoints view and in the Edit Breakpoints dialog box if it is open. PowerBuilder 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:

- ◆ Click the red circle next to the breakpoint in the Source view, Breakpoints view, or Edit Breakpoints dialog box.

or

Select Disable Breakpoint from the popup menu in the Source view.

or

Select Disable from the popup 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 popup menus or by clicking the white circle.

Disabling all
breakpoints

To disable all breakpoints, select Disable All from the popup menu in the Breakpoints view.

❖ To clear a breakpoint:

- ◆ Double-click the line containing the breakpoint in the Source view.

or

Select Clear Breakpoint from the popup menu in the Source view.

or

Select Clear from the popup menu in the Breakpoints view.

or

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 popup menu in the Breakpoints view.

Running in debug mode

Once you have set your breakpoints, you can run the application in debug mode. The application executes normally until it hits a statement containing a breakpoint. After examining your application, you can single-step through it, continue execution until execution reaches another breakpoint, or stop the debugging run so that you can start a new debugging run or close the Debug window.

❖ **To run an application in debug mode:**

- 1 If necessary, open the Debug window by clicking the Debug button.
- 2 Click the Start button in the PainterBar.

or

Select Debug > Start from the menu bar.

The application starts and runs until it hits a breakpoint. You return to the Debug window, with the line containing the breakpoint displayed. The yellow arrow cursor indicates that this line contains the next statement to be executed. You can now examine the application using Debug window views and tools.

FOR INFO For more information, see "Examining an application at a breakpoint" on page 833 and "Stepping through an application" on page 839.

❖ **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:**

- ◆ Select Debug > Stop from the menu bar.

PowerBuilder resets the state of the application and all the Debug window views to their state at the beginning of the debugging run. You can now begin another run in debug mode, or close the Debug window.

Cleaning up

When you terminate a debugging run or close the Debug window without terminating the run, PowerBuilder executes the application's close event and destroys any objects, such as autoinstantiated local variables, that it would have destroyed if the application had continued to run and exited normally.

Examining an application at a breakpoint

When an application is suspended at a breakpoint, use the Variables, Watch, Call Stack, and Objects in Memory views to examine its state.

About icons used in debugging views

The Variables, Watch, and Objects in Memory views use many of the icons used in the PowerBuilder Browser as well as some additional icons: I represents an Instance; F, a field; A, an array; and E, an expression.

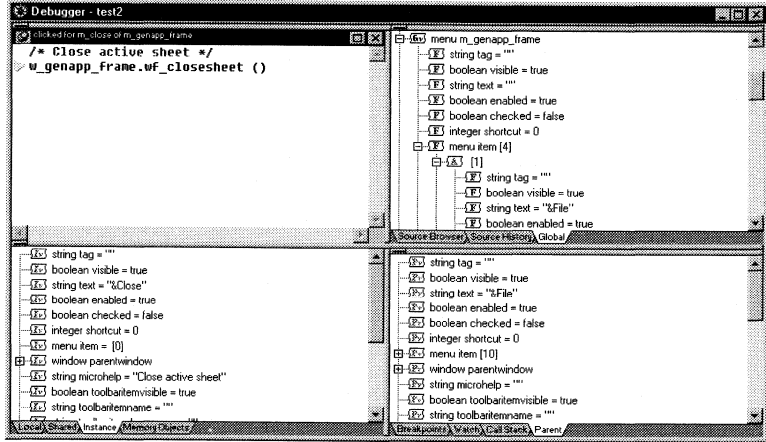
Examining variable values

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.

Variable type	What the Variables view shows
Local	Values of variables that are local to the current script or function
Global	Values of all global variables defined for the application and properties of all objects (such as windows) that are open
Instance	Properties of the current object instance (the object to which the current script belongs) and values of instance variables defined for the current object
Parent	Properties of the parent of the current instance
Shared	Objects, such as application, window, and menu objects, that have been opened and the shared variables associated with them

About Instance and Parent variables

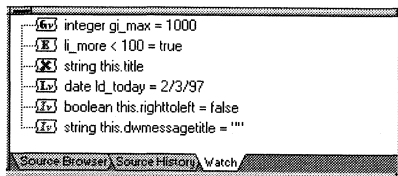
In the following illustration, an application has stopped at a breakpoint in the script for the Clicked event for the Close menu item on a frame's File menu. The Instance tab shows the properties of the current instance of the Close menu item and the Parent tab shows the properties of its parent, an instance of the File menu. Navigating through the hierarchy on the Global tab shows the same objects.



Watching variables and expressions

The Watch view lets you monitor the values of selected variables and expressions as the application runs.

If the variable or expression is in scope, the Watch view shows its value. Empty quotes indicate that the variable is in scope but has not been initialized. An X in the Watch view indicates that the variable or expression is not in scope.



Setting variables and expressions in the Watch view

You can select variables you want to watch as the application runs by copying them from a Variables view. You can also set a watch on any PowerScript expression. When you close the Debug window, any watch variables and expressions you set are written to your PowerBuilder initialization file.

❖ **To add a variable to the Watch view:**

- 1 Select the variable in the Variables view.
- 2 Drag it to the Watch view.
or
Click the Add Watch button on the PainterBar.
or
Select Debug > Add Watch from the menu bar.
PowerBuilder adds the variable to the watch list.

❖ **To add an expression to the Watch view:**

- 1 Select Insert from the popup menu.
- 2 Type any valid PowerScript expression in the New Expression dialog box and click OK.
PowerBuilder adds the expression to the watch list.

❖ **To edit 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 popup 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 Select Clear from the popup menu.
or
Click the Remove Watch button on the PainterBar.
or
Select Debug > Remove Watch from the menu bar.

- ❖ **To clear all variables and expressions from the Watch view:**
 - ◆ Select Clear All from the popup menu

Monitoring the call stack

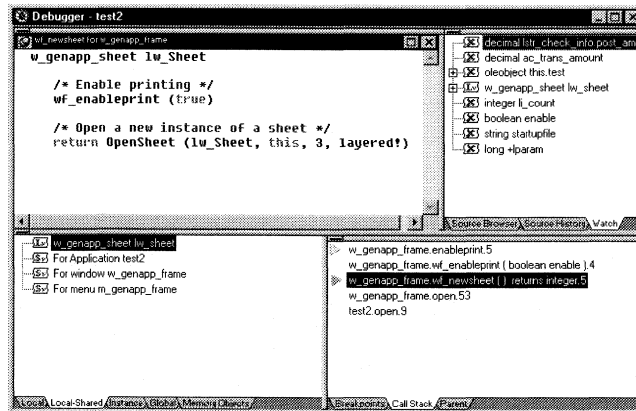
The Call Stack view shows the sequence of function calls leading up to the script or function that was executing at the time of the breakpoint. Each line displays as the script and line number from which the call was made. The yellow arrow shows the script and line where execution was suspended.

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:**

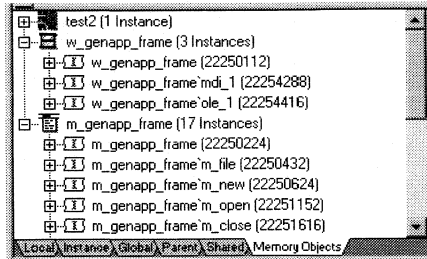
- 1 Select a line in the Call Stack view.
- 2 Double-click the line.
or
Select Show Context from the popup menu.
or
Drag the line into the Source view.

A green arrow indicates the script that you selected. The Source view shows the script and line number you selected, and the Variables and Watch views show the variables and expressions in scope in that context.



Examining objects in memory

The Objects in Memory view shows an expandable list of objects currently in memory. Double-click the name of an object or click the plus sign next to it to view the names and memory locations of instances of each object and property values of each instance.



Using the Source view

The Source view displays the full text of a script. As you run or step through the application, the Source view is updated to show the current script with a yellow arrow indicating the next statement to be executed.

Multiple Source views

You can open more than one source view. If there are multiple source views open, only the first one opened is updated to show the current script when the context of an application changes.

Changing the Source view

From the popup 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:

- ◆ Drag the name of a script to the Source view from the Call Stack, Source Browser, or Source History views.

or

Select a line and then select Open Source from the popup menu in the Breakpoints, Source Browser, or Source History views.

or

Select Edit > Select Script from the menu bar.

❖ **To find a specified string in the Source view:**

- 1 Select Find from the popup menu.
or
Select Edit > Find from the menu bar.
The Find Text dialog box opens.
- 2 Type the string in the Find textbox 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 Double-click the script.
or
Select the script and select Open Source from the popup menu.
or
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 lists all the scripts that have been opened in the current debugging session. Use the same techniques as in the Source Browser to display a selected script in the Source view.

Source History limit

The Source History list shows up to 100 scripts and is *not* cleared at the end of each debugging run. It is cleared when you close the Debug window, or you can clear the list from the popup 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 your code, the Debug window 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 that you don't want to single step through, 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 may reach sections of code that you aren't interested in examining closely. The code may contain a large loop, or it may be well-tested code that you are confident is free of errors. You can use Run To Cursor to select a statement further down in a script or in a subsequent script where you want execution to stop.

❖ **To step through multiple statements:**

- 1 Click on the line in the script where you want to resume single stepping.
- 2 Click the Run To Cursor button in the PainterBar.
or
Select Debug > Run To Cursor from the menu bar.

PowerBuilder executes all intermediate statements and the yellow arrow cursor displays at the line where you set the cursor.

Bypassing statements

You can use Set Next Statement to bypass a section of code that contains a bug, or to test part of an application using specific values for variables. Execution continues from the statement where you place the cursor. Be cautious when you use Set Next Statement, because results may be unpredictable if, for example, you skip code that initializes a variable.

❖ **To set the next statement to be executed:**

- 1 Click on the line in the script where you want to continue execution.
- 2 Click the Set Next Statement button in the PainterBar.
or
Select Debug > Set Next Statement from the menu bar.
- 3 If necessary, change the values of variables.
- 4 Continue execution using Continue, Step In, or Step Over.

If you select Continue, PowerBuilder begins execution at the statement set and continues to the next breakpoint. If you select Step In or Step Over, PowerBuilder sets the next statement and displays the yellow arrow cursor at the line where you set the cursor.

Changing a variable's value

As you step through the application, you can change the values of variables that are in scope. You may want to do this to examine different flows through the application, to simulate a condition that is difficult to reach in normal testing, or if you are skipping code that sets a variable's value.

Enumerated variables

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 popup menu, select Edit Variable.
- 3 Type a value for the variable or select the Null checkbox 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, close the Debug window before you fix it. After you have fixed the problem, you can reopen the Debug window 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. Suppose GlobalWindow is a global window of type window, you could add the following line to the end of the preceding script:

```
GlobalWindow = mywin
```

You can look at and modify the opened window through the global variable. When you have finished debugging the window, you can remove the global variable and the statement assigning the local to the global.

Just-in-time debugging

If you are running your application in regular mode, using the Run button, and you notice that the application is behaving incorrectly, just-in-time debugging lets you switch to debug mode without terminating the application.

When you open the Debug window while running an application, the application *does not* stop executing. The Source, Variables, Call Stack, and Objects in Memory views are all empty because the debugger does not have any context. To suspend execution and examine the context in a problem area, open an appropriate script and set breakpoints, then initiate the action that calls the script.

If just-in-time debugging is enabled and a system error occurs while an application is running in regular mode, the Debug window 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 System Options from the PowerPanel.
- 2 Check the Just In Time Debugging checkbox and click OK.

❖ **To debug an application while running in regular mode:**

- 1 Click on the PowerBuilder icon and click the Debug button in the dialog box that displays.

On Macintosh

On the Macintosh, press OPTION-ESC to open the Debug window.

- 2 Open a script in the Source view and set breakpoints.

The application is suspended when it hits a breakpoint and the Source, Variable, Call Stack, and Objects in Memory views show the current context. You can now debug the application.

Running an application

When the application seems fine, you are ready to run it in regular mode. In regular mode, the application responds to user interaction and continues to run until the user exits the application or an execution-time error occurs. You can rely on the default execution-time error reporting by PowerBuilder or write a script that specifies your own error processing. You can also generate a diagnostic trace of your application's execution.

FOR INFO For how to analyze your application's logic and performance, see "Tracing and profiling an application" on page 851.

Running the application

❖ To run the current application:

- ◆ Click the Run button in the PowerBar or PowerPanel.
or
Select File>Run from the menu bar.

PowerBuilder becomes minimized (its icon displays at the bottom of the screen along with the icons of other minimized applications), and your application executes.

❖ To stop a running application:

- ◆ Either end the application normally, or double-click the PowerBuilder icon to stop it immediately.

Handling errors during execution

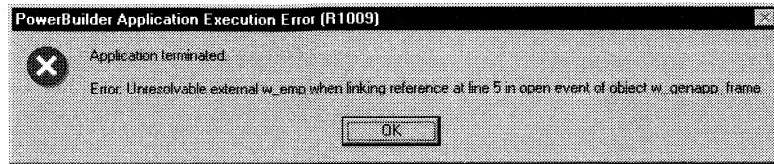
A serious error during execution (such as attempting to access a window that has not been opened) will trigger the SystemError event in the application object.

If there is no
SystemError script

If you do not write a SystemError script to handle these errors, PowerBuilder displays a message box containing the following information:

- ◆ The number and text of the error message
- ◆ The line number, event, and object in which the error occurred

There is also an OK button that closes the message box and stops the application.



If there is a
SystemError script

If there is a script for the SystemError event, PowerBuilder executes the script and does not display the message box. It is a good idea to build an application-level script for the SystemError event to trap and process any execution-time errors, as described in "Using the Error object" next.

Using the Error object

In the script for the SystemError event, you can access the built-in Error object to learn which error occurred and where it occurred. The Error object contains the following properties:

Property	Data type	Description
Number	Integer	Identifies the PowerBuilder error
Text	String	Contains the text of the error message
WindowMenu	String	Contains the name of the window or menu in which the error occurred
Object	String	Contains the name of the object in which the error occurred. If the error occurred in a window or menu, Object will be the same as WindowMenu
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 specify that you want to use your user object inherited from Error as the global Error object in your application.

FOR INFO For more information, see "Building a standard class user object" on page 303.

Execution-time error numbers

The following table lists the execution-time error numbers returned in the Number property of the Error object and the meaning of each number:

Number	Meaning
1	Divide by zero
2	Null object reference
3	Array boundary exceeded
4	Enumerated value is out of range for function
5	Negative value encountered in function
6	Invalid DataWindow row/column specified
7	Unresolvable external when linking reference
8	Reference of array with null subscript
9	DLL function not found in current application
10	Unsupported argument type in DLL function
11	Object file is out of date 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 <i>name</i>
16	Maximum string size exceeded

Number	Meaning
17	DataWindow referenced in DataWindow object does not exist
18	Function doesn't return value
19	Cannot convert <i>name</i> in Any variable to <i>name</i>
20	Database command has not been successfully prepared
21	Bad runtime function reference
22	Unknown object type
23	Cannot assign object of type <i>name</i> to variable of type <i>name</i>
24	Function call doesn't match its definition
25	Double or Real expression has overflowed
26	Field <i>name</i> assignment not supported
27	Cannot take a negative to a noninteger power
28	VBX Error: <i>name</i>
29	Nonarray expected in ANY variable
30	External object does not support data type <i>name</i>
31	External object data type <i>name</i> not supported
32	Name not found calling external object function <i>name</i>
33	Invalid parameter type calling external object function <i>name</i>
34	Incorrect number of parameters calling external object function <i>name</i>
35	Error calling external object function <i>name</i>
36	Name not found accessing external object property <i>name</i>
37	Type mismatch accessing external object property <i>name</i>
38	Incorrect number of subscripts accessing external object property <i>name</i>
39	Error accessing external object property <i>name</i>
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 <i>name</i>

Number	Meaning
43	Parent object doesn't 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
50	Distributed Service Error
51	Bad argument list for function/event
52	Distributed Communications Error
53	Requested server not active
54	Server not accepting requests
55	Request terminated abnormally
56	Response to request incomplete
57	Not connected
58	Object instance does not exist
59	Invalid column range
60	Invalid row range
61	Invalid conversion of <i>number</i> 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:\npass by value/reference mismatch
66	Invalid subscript for array index operation
67	NULL object reference cannot be assigned or passed to an autoinstantiate

Number	Meaning
68	NULL object reference cannot be passed to external DLL function name
69	Function <i>name</i> cannot be called from a secured runtime session
70	External DLL function <i>name</i> cannot be called from a secured runtime session
71	General protection fault occurred
72	<i>name</i> failed with an operating system error code of <i>number</i>
73	Reference parameters cannot be passed to an asynchronous shared/remote object method
74	Reference parameters cannot be passed to a shared object method
75	The server has forced the client to disconnect
76	Passing NULL as a parameter to external function <i>name</i>
77	Object passed to shared/remote object method is not a nonvisual user object
78	Listen can only be done in Enterprise version of PowerBuilder
79	The argument to <i>name</i> must be an array
80	The server has timed out the client connection
81	Function argument file creator must be a four character string
82	Function argument file type must be a four character string
83	Attempt to invoke a function or event that is not accessible
84	Wrong <i>number</i> of arguments passed to function/event call
85	Error in reference argument passed in function/event call
86	Ambiguous function/event reference
87	The connection to the server has been lost
88	Cannot ask for ClassDefinition Information on open painter: name
89	5.0 style proxy objects are not supported. Copy the new style proxy that was generated at migration time
90	Cannot assign array of type name to variable of type array of name

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.

FOR INFO For information about the PopulateError and SignalError functions, see the *PowerScript Reference*.

Tracing and profiling an application

When you run an application, you can generate an execution **trace** file. You use the trace file to create a **profile** of your application.

The profile shows you which functions and events were called by which other functions and events, how often they were called, when garbage collection occurred, when objects were created and destroyed, and how long each activity took to complete. This information will help you identify areas that you should rewrite to improve performance and find errors in the application's logic.

Enterprise only feature

The tracing and profiling features are only available in the Enterprise edition of PowerBuilder. All editions of PowerBuilder let you generate a simple text trace file without timer values by checking Enable PBDebug Tracing in the System Options dialog box.

FOR INFO For more about PBDebug, see "Generating a trace file without timing information" on page 870.

When you can trace an application

You can create a trace file when you run an application in the PowerBuilder environment, and when you run an executable outside PowerBuilder. For machine-code executable files, the trace file is only generated if you check the Trace Information checkbox when you build the executable.

When you run an application with tracing turned on, PowerBuilder records a timer value in a data file every time a specific activity occurs. You control when logging begins and ends and which activities are recorded.

Creating profiles

Once you have generated a trace file, you can create several different profiles or views of the application by extracting different types of information from the trace file.

PowerBuilder provides a tool called the Application Profiler that creates profiles (views) of the application for you, but you can also create your own analysis tools.

Examining these views tells you where the application is spending the most time. You can also find routines that are being called too often or that are being called from classes or routines that you didn't expect to call them. You may find routines that aren't being called at all.

The Application Profiler

The Application Profiler provides an easy way to create and print profile reports. It provides three views:

- ◆ **Class View** shows information about the objects that were used in the application run.

For each object, the class view shows all the routines called from each class with the number of times each routine was called (hit) as well as timing information for each call. The following illustration shows part of a Class View. Embedded SQL commands are shown as being called from a pseudo class called ESQL.

Calling Routine	Hits	Self	% Self	Self + Called	% Self + Called
w_ct_modify_describe.open()	1	0.30	0.02	894.19	60.47
Routine					
<ESQL>.DB Connect	2	63.63	4.30	63.63	4.30
datawindow.retrieve()	2	700.79	47.40	732.29	49.53
datawindow.rowcount()	1	0.04	0.00	0.04	0.00
datawindow.settransobject()	2	0.06	0.00	0.06	0.00
messagebox()	1	97.78	6.61	97.78	6.61
w_ct_modify_describe.identify_ah	1	0.07	0.00	0.08	0.01

- ◆ **Routine View** shows information about all the routines (functions and events) that were used in the application run.

For each routine, the Routine View shows the routine that called it and the routines it called, with detailed timing information for each routine.

Called By	Hits	Self	% Self	Self + Called	% Self + Called
open()	1	0.30	0.00	894.19	100.00

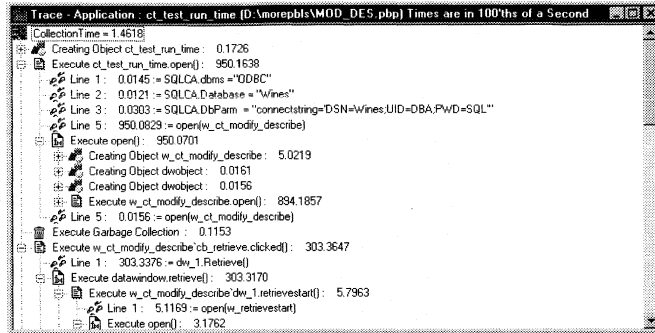
Routine	Hits	Self	% Self	Self + Called	% Self + Called
<System>	1	0.00	0.00	478.62	100.00
datawindow.retrieve()	4	186.97	80.21	236.58	83.56
open()	4	67.27	4.95	956.64	64.70
ct_test_run_time.open()	1	0.09	0.01	950.16	64.26
w_ct_modify_describe.open()	1	0.30	0.02	894.19	60.47
w_ct_modify_describe.cb_retrieve.clicked()	2	0.10	0.01	503.39	34.04
messagebox()	1	97.78	6.61	97.78	6.61

Calls	Hits	Total	% Total
datawindow.retrieve()	2	732.29	81.89
messagebox()	1	97.78	10.94
<ESQL>.DB Connect	2	63.63	7.12
w_ct_modify_describe.identify_another_transo	1	0.08	0.01

Routine Detail	
Called:	1
Self:	0.3021
% Self:	0.0200
Self Min:	0.3021
Self Max:	0.3021
Self + Called:	894.1957
% Self + Called:	60.4740
Self + Called Min:	894.1957
Self + Called Max:	894.1957

- ◆ **A Trace View** shows the elapsed time taken by each activity in chronological order.

The following illustration shows a Trace Tree view with several nodes expanded. The number to the right of each item is the execution time for that item.



FOR INFO If you plan to use the Application Profiler, see "Collecting trace information" on page 853 to find out how to create a trace file, and then use the online Help for the Application Profiler to find out how to use it.

Using the Application Profiler as a model

Even if you want to develop your own analysis tools, using the Application Profiler is a good way to learn about profiling in PowerBuilder. The Application Profiler is provided as a sample application with PowerBuilder Enterprise so that you can use it as a model for your own analysis tools.

FOR INFO When you're ready to develop your own tools, see "Analyzing trace information" on page 860 for an overview of the approaches you can take.

Collecting trace information

There are three ways to collect trace information. You can use:

- ◆ The Profiling tab on the System Options dialog box
- ◆ The `w_starttrace` window
- ◆ Trace objects and functions

Use the Profiling tab if you want to trace an entire application run in the development environment.

FOR INFO See "Tracing an entire application in PowerBuilder" on page 856.

Use the `w_starttrace` 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.

FOR INFO See "Using the `w_starttrace` window" on page 856 and "Collecting trace information using PowerShell functions" on page 857.

Whichever method you use, you can specify:

- ◆ The name and location of the trace file and optional labels for blocks of trace data
- ◆ The kind of timer used in the trace file
- ◆ The activities you want recorded in the trace file

Trace file names and labels

The default name of the trace file is the name of the application with the extension `PBP`. The trace file is saved in the directory where the `PBL` or executable file resides and overwrites any existing file of the same name. If you run several different tests on the same application, you will want to 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 for each part, called a **trace block**.

Timer kinds

There are three kinds of timer: clock, process, and thread. If your analysis does not require timing information, you can omit timing information from the trace file to improve performance.

On Windows NT and Windows 95 If you don't 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 startup time. The clock timer measures time in microseconds. Depending on the speed of your machine's central processing unit, the clock timer can offer a resolution of less than one microsecond. A timer's resolution is the smallest unit of time the timer can measure.

You can also use process or thread timers, which measure time in microseconds with reference to when the process or thread being executed started. You should always use the thread timer for distributed applications. Both process and thread timers exclude the time taken by any other running processes or threads so that they give you a more accurate measurement of how long the process or thread is taking to execute, but both have a lower resolution than the clock timer.

On Windows 3.1, UNIX, and Macintosh Specifying a different timer kind has no effect on other platforms—the same timer is always used. On UNIX, times are always recorded using the thread timer.

Collection time

The timer values in the trace file exclude the time taken to collect the trace data.

Trace activities

You can choose to record in the trace file the time at which any of the following activities occurs. If you're using the System Options dialog box or the `w_starttrace` window, you select the checkboxes 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.

Trace Activities checkbox	What is recorded	TraceActivity value
Routine Entry/Exit	Routine entry or exit	ActRoutine!
Routine Line Hits	Execution of any line in any routine	ActLine!
Embedded SQL	Use of an embedded SQL verb	ActESQL!
Object Creation/ Destruction	Object creation or destruction	ActObjectCreate!, ActObjectDestroy!
User Defined Activities	A user-defined activity that records an informational message	ActUser!
System Errors	A system error or warning	ActError!
Garbage Collection	Garbage collection	ActGarbageCollect!
Not available	Routine entry and exit, embedded SQL verbs, object creation and destruction, and garbage collection	ActProfile!
Not available	All except ActLine!	ActTrace!

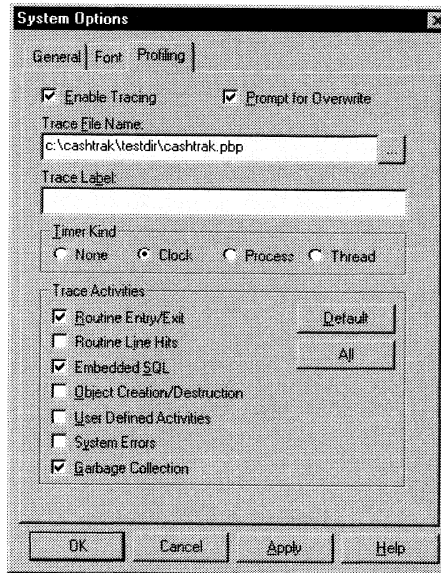
When you begin and end tracing, an activity of type `ActBegin!` is automatically recorded in the trace file. User-defined activities, which you use to log informational messages to the trace file, are the only trace activities enabled by default.

Tracing an entire application in PowerBuilder

Use the Profiling tab on the System Options dialog box if you want to collect trace data for an entire application run in the PowerBuilder development environment.

❖ **To trace an entire application in PowerBuilder:**

- 1 Click the Options button in the PowerPanel and select the Profiling tab.



- 2 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 the w_starttrace window

The PBL provided with the Application Profiler sample application includes a window that is similar to the Profiling tab on the System Options dialog box.

The Start button on the window opens a trace file, enables the trace activities you requested, and starts tracing. The Stop button stops tracing and closes the trace file. The View Output button opens the Application Profiler.

If you add this window to your application, you can easily trace specific parts of the application as you develop it. If you want to, you can include it in a deployed application so that users can generate a trace file.

Supporting files needed

To create a profile from a trace file, PowerBuilder must also access the PBL, PBD, or executable file used to create the trace file, and the PBL, PBD, or executable file must be in the same location as when the trace file was created.

❖ **To use the `w_starttrace` window in an application:**

- 1 In the Library painter, copy `w_starttrace` from the sample PROFILE.PBL to your application PBL.
- 2 Add a script that opens the `w_starttrace` window.

For example, you could add a command button to the application's main window and code `open(w_starttrace)` in the button's Clicked event.

Collecting trace information using PowerScript functions

You use the following PowerScript system functions to collect information in a trace file. Each of these functions returns a value of type `ErrorReturn`, an enumerated data type.

Use this PowerScript function	To do this
<code>TraceOpen</code>	Open a named trace file and set the timer kind
<code>TraceEnableActivity</code>	Enable logging of the specified activity
<code>TraceBegin</code>	Start logging all enabled activities. You can pass an optional label for the trace block
<code>TraceError</code>	Log a severity level and error message to the trace file
<code>TraceUser</code>	Log a reference number and informational message to the trace file
<code>TraceEnd</code>	Stop logging all enabled activities
<code>TraceDisableActivity</code>	Disable logging of the specified activity
<code>TraceClose</code>	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've finished tracing.

TraceEnableActivity and TraceDisableActivity can only be called 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 only be called 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 CASE statement. 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 Trace view in the Application Profiler to pinpoint problems quickly.

FOR INFO For detailed information about these functions, see the *PowerScript Reference*.

Example: trace data collection

In this example, the user selects a timer kind from a dropdown listbox and enters a name for the trace file in a single-line edit. The example only uses the ErrorReturn return value from the TraceOpen call to return an error message if the call fails, but typically you would do this for every trace 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

PowerBuilder provides three ways to analyze trace information:

- ◆ 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 Application Profiler uses the first two ways. The Application Profiler's Class and Routine views are based on a call graph model, and the Trace view is based on a trace tree model.

FOR INFO For how to use the Application Profiler, see the online Help provided with it.

The rest of this section describes how to analyze trace information using PowerBuilder objects and functions.

Analyzing performance

You use the following PowerScript functions and PowerBuilder objects to analyze the performance of an application.

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

Use this function	With this object	To do this
ClassList	Profiling	Get a list of classes in the model
SystemRoutine	Profiling	Get the name of the routine node that represents the root of the model
IncomingCallList	ProfileRoutine	Get a list of routines that called a specific routine
OutgoingCallList	ProfileRoutine and ProfileLine	Get a list of routines called by a specific routine or from a specific line
LineList	ProfileRoutine	Get a list of lines in the routine in line order
DestroyModel	Profiling	Destroy the current performance analysis model and all the objects associated with it

Each of these functions returns a value of the enumerated data type `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.

FOR INFO For more information about these functions and objects, see the *PowerScript Reference* and *Objects and Controls*.

Using the `BuildModel` function

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 per cent of the activities have been processed. The progress of the build is shown in a window called w_progress that has a cancel button.

FOR INFO For a more complete version of the example, see the Application Profiler sample application.

```
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

Once you've built a call graph model of the application, you can extract detailed information from it.

For routines and lines, you can extract the following timing information from the ProfileRoutine and ProfileLine objects:

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

Property	What it means
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: 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.

FOR INFO For more detailed examples of this and similar functions, see the Application Profiler sample application.

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

```

Building a trace tree model

You use the following PowerScript functions and PowerBuilder objects to build a nested trace tree model of an application.

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 `EntryList` and `GetChildrenList` 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:

<code>TraceTreeError</code>	<code>TraceTreeESQL</code>	<code>TraceTreeGarbageCollect</code>
<code>TraceTreeLine</code>	<code>TraceTreeObject</code>	<code>TraceTreeRoutine</code>
<code>TraceTreeUser</code>		

FOR INFO For more information about these functions and objects, see the *PowerScript Reference* and *Objects and Controls*.

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 INFO For more about using `BuildModel`, see "Using the `BuildModel` function" on page 861.

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 data type, 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. (For a more detailed example, see the Application Profiler sample application.)

```
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. Here's 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 ltctrtr_routine
        ltctrtr_routine = atctn_node
        IF ltctrtr_routine.Classname = '' THEN &
            ls_routinename = ltctrtr_routine.ClassName + "."
        END IF
        ls_routinename += ltctrtr_routine.Name
        ltctrtr_routine.GetChildrenList(ltctn_list)
        ll_node_cnt = UpperBound(ltctn_list)

        ls_label = "Execute " + ls_routinename + ' :' + &
            space(ii_offset) + String(l_timescale * &
```

```

        (ltctrtrt_routine.ExitTimerValue - &
         ltctrtrt_routine.EnterTimerValue), '0.000000')
astr_node.Children = (ll_node_cnt > 0)
astr_node.Label = ls_label
astr_node.Time = ltctrtrt_routine.EnterTimerValue
RETURN
CASE ActLine!
TraceTreeLine tctltn_treeLine
tctltn_treeLine = atctn_node
ls_label = LINEPREFIX + &
         String(tctltn_treeLine.LineNumber )
astr_node.time = tctltn_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 following PowerScript functions and PowerBuilder objects to access the data in the trace file directly so that you can develop your own analysis tools.

Use this function	With this object	To do this
Open	TraceFile	Opens the trace file to be analyzed
NextActivity	TraceFile	Returns the next activity in the trace file. The value returned is of type TraceActivityNode
Reset	TraceFile	Resets the next activity to the beginning of the trace file
Close	TraceFile	Closes the open trace file

With the exception of `NextActivity`, each of these functions returns a value of type `ErrorReturn`. Each `TraceActivityNode` object includes information about the category of the activity, the timer value when the activity occurred, and the activity type.

Timer values

The category of the activity is either `In` or `Out` for activities that have separate beginning and ending points, such as routines, garbage collection, and tracing itself. Each such activity has two timer values associated with it: the time when it began and the time when it completed.

Activities that have only one associated timer value are in the category `Atomic`. `ActLine!`, `ActUser!`, and `ActError!` are all atomic activities.

Inherited objects

The following objects inherit from `TraceActivityNode` and contain data about the associated activity type:

<code>TraceBeginEnd</code>	<code>TraceError</code>	<code>TraceESQL</code>
<code>TraceGarbageCollect</code>	<code>TraceLine</code>	<code>TraceObject</code>
<code>TraceRoutine</code>	<code>TraceUser</code>	

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 INFO For more information about these functions and objects, see the *PowerScript Reference* and *Objects and Controls*. For a list of activity types, see "Trace activities" on page 855.

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 using the `TraceActivity` enumerated data type, 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 `ltrace_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()

```

Here's part of `of_dumpActivityNode`:

```

string lstr_result

lstr_result = String(Truncate(atcan_node. &
    TimerValue, 6)) + " "
CHOOSE CASE atcan_node.ActivityType
CASE ActRoutine!
    TraceRoutine ltcrtr_routine
    ltcrtr_routine = atcan_node
    IF ltcrtr_routine.IsEvent THEN
        lstr_result += "Event: "
    ELSE
        lstr_result += "Function: "
    END IF
    lstr_result += ltcrtr_routine.ClassName + "." + &
    ltcrtr_routine.name + "(" + &
    ltcrtr_routine.LibraryName + ")" + " " &

```

```
        + String(ltcrt_routine.ObjectId) + "~r~n"
CASE ActLine!
    TraceLine ltcln_line
    ltcln_line = atcan_node
    lstr_result += "Line: " + &
        String(ltcln_line.LineNumber) + "~r~n"
CASE ActESQL!
    TraceESQL ltcsql_esql
    ltcsql_esql = atcan_node
    lstr_result += "ESQL: " + ltcsql_esql.Name &
        + "~r~n"

// CASE statements and code omitted
...
CASE ActBegin!
    IF atcan_node.Category = TraceIn! THEN
        lstr_result += "Begin Tracing~r~n"
    ELSE
        lstr_result += "End Tracing~r~n"
    END IF
CASE ActGarbageCollect!
    lstr_result += "Garbage Collection~r~n"
CASE else
    lstr_result += "Unknown Activity~r~n"
END CHOOSE

RETURN lstr_result
```

Generating a trace file without timing information

If you want to generate an activity log with no timing information in a text file, you can turn on PBDebug tracing in the System Options dialog box. The PBDebug trace file contains a log showing which object functions and instructions and system DLL functions were executed in chronological order.

❖ **To generate a simple trace file:**

- 1 Select Enable PBDebug Tracing in the System Options dialog box.
- 2 (Optional) Specify a path name for the PBDebug output file.
- 3 Restart PowerBuilder and run your application

If you don't specify an output file path, PowerBuilder creates an output file in the same directory as the PowerBuilder executable file. The output file has the same name as the PowerBuilder executable with the extension DBG. If you don't have write permission to this directory, you must specify a value for DebugOutFile.

Managing trace files

Each time you run your application with PBDebug on, PowerBuilder appends the new trace output to the same trace file. A long session with PBDebug turned on can produce a large trace file, so you must ensure you have adequate disk space.

Each time the PowerBuilder development environment is started with PBDebug on, the trace file is overwritten. If you want to save the data in the existing trace file, rename the output file between sessions.

Turning PBDebug off

Running your application with PBDebug on will slow down execution. Be sure to clear the Enable PBDebug Tracing checkbox on the System Options dialog box if you don't need this trace information.

FOR INFO For information on creating the same kind of diagnostic trace file when you run your compiled application outside PowerBuilder, see "Tracing execution" on page 901.

About this chapter

This chapter describes how to create an executable version of your application.

Contents

Topic	Page
Overview of creating an executable	874
Defining a project	875
Using dynamic libraries	883
Building a project	885
Distributing resources	891

Overview of creating an executable

You want your users to be able to execute your application the same way they execute other applications on their computer. To do that, you create an executable version of your application.

Packaging your application

You have two basic ways to package your application:

- ◆ As one standalone executable file that contains all the objects in the application
- ◆ As an executable file and one or more dynamic libraries that contain objects that are linked at execution time

Providing other resources

You may also need to provide some additional resources that your application uses, such as bitmaps and icons. There are two ways to provide resources:

- ◆ Distribute them separately
- ◆ Include them in a PowerBuilder resource file and build an executable or dynamic library using the resource file

This chapter describes how to use PowerBuilder to create an executable version of your application.

How to proceed

Read the chapter on packaging your application for deployment in *Application Techniques* to get an understanding of the best way for you to package your application. Then follow the procedures in this chapter to implement your strategy.

Defining a project

This section describes how to create an executable application using the Project painter.

About the Project painter

The Project painter allows you to streamline the generation of executable files and dynamic libraries. You use the Project painter to create and maintain PowerBuilder project objects. When you build a project object, you specify the following components of your application:

- ◆ Executable filename
- ◆ Which of the libraries you want to distribute as dynamic libraries
- ◆ Which PowerBuilder resource files (if any) should be used to build the executable file and the dynamic libraries
- ◆ Which build options you want to use in your project
- ◆ Which code generation options you want to use

On Macintosh

On the Macintosh, you can also specify a Macintosh resource filename, application signature, and minimum and preferred heap sizes.

Once you have defined the project, you can rebuild your application from the Project painter by simply clicking the Build button.

Building a project object for your application can greatly reduce the amount of time spent creating an executable, dynamic libraries, and PowerBuilder resource files for every build you want to generate.

Using a version control system

If you are using a version control system to manage your PowerBuilder objects, you can also use the Project painter to restore previous versions of your libraries.

FOR INFO For more information, see *Version Control Interfaces*.

On UNIX

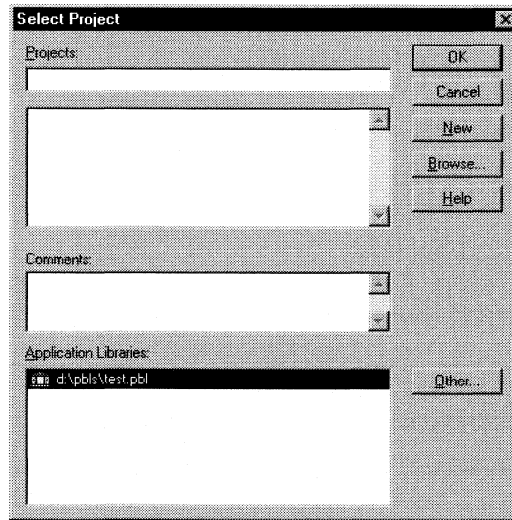
PowerBuilder does not currently support any interfaces to third-party version control products (such as PCVS) on UNIX systems.

Defining a project object

❖ **To define a project object:**

- 1 Open the Project painter by clicking the Project button in the PowerBar or PowerPanel.

The Select Project dialog box displays.



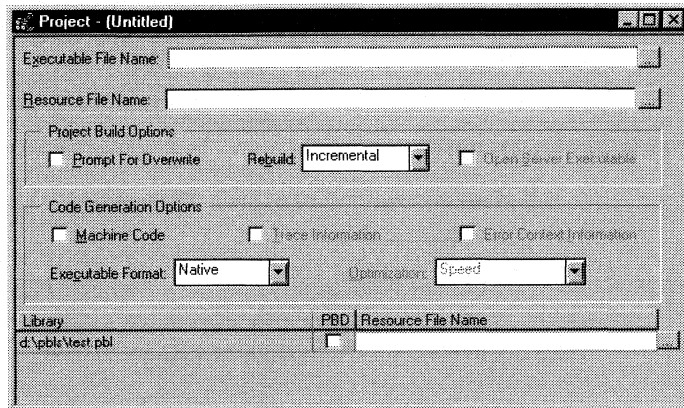
- 2 Click New to create a new project object.

The New Project dialog box displays.

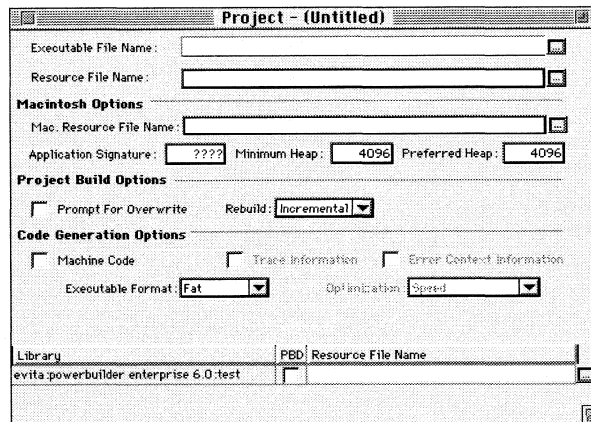
- 3 Select Application and click OK.

The Project painter workspace displays.

The following figure shows the Project painter on the Windows platform.



There are some minor differences in the Project painter on other platforms. On the Macintosh, you can specify several additional options.



- 4 Specify the options you prefer.

FOR INFO For information about each option, see "Project painter options" next.

- 5 When you have finished defining the project object, save the object by selecting File>Save from the menu bar.
PowerBuilder saves the project as an independent object in the specified library. Like other objects, projects are displayed in the Library painter.
- 6 Close the Project painter by selecting Close from the File menu.

Project painter options

This section describes each of the options you can specify in the Project painter.

Executable File Name The name you specify for your executable must have the extension EXE on the Windows platform. The executable will be saved to your current directory. Click the button next to the textbox to navigate to a different directory.

Resource File Name You need to specify a PowerBuilder resource file (PBR) for your executable if you dynamically reference resources (such as bitmaps and icons) in your scripts *and* you want the resources included in the executable file instead of having to distribute the resources separately.

You can type the name of a resource file in the Resource File Name box or click the button next to the textbox to browse your directories for the resource file you want to include.

FOR INFO For more about PowerBuilder resource files, see "Distributing resources" on page 891.

Macintosh options

The following options are available only on the Macintosh.

Mac Resource File Name If you want to supply your own versions of the resources that your PowerBuilder application needs, create a file of resources and name that file in the Mac Resource File Name box. Select Edit>Paste Macintosh Resource File Name to use the file-finding dialog box.

FOR INFO For information about what you need to put in the resource file, see "Macintosh resources" on page 895.

Application Signature	The four-character signature you specify in the Application Signature box will be stored in the BNDL resource of your application, overriding any signature it already contains. Signatures for applications that will be deployed outside your company should be registered with Apple to avoid conflicts with other applications.
Minimum and Preferred Heap	You can specify minimum and preferred memory requirements for your application in the Minimum Heap and Preferred Heap boxes. Typically, a small PowerBuilder application requires approximately 4 MB of memory. More complex applications require 6 or 8 MB or more.

Application versus system memory

Do not set your application's memory requirements too high. There must be enough memory available for the system to load approximately 5 MB of shared libraries.

PowerBuilder uses CFM to manage its code libraries as system resources. These libraries are the equivalent of dynamic linked libraries (DLLs) on Windows. These libraries are deployed with your application and are stored in the System folder. CFM loads the libraries into *system* memory when you run your application, not in your application's memory allocation.

Project build options

Prompt for Overwrite	PowerBuilder overwrites any files it creates when building your application. Select Prompt For Overwrite if you want PowerBuilder to prompt you before overwriting files.
Rebuild	<p>Specify either Full or Incremental from the Rebuild dropdown listbox to indicate whether you want PowerBuilder to regenerate all objects in the application libraries before it creates the executable and dynamic libraries. If you choose incremental, PowerBuilder only regenerates those objects referenced by objects that have changed since the last time you built your application.</p> <p>In most cases, you will want to regenerate your objects before rebuilding your project as a precaution.</p>

Code generation options

Most of these options are disabled if compiled code is not supported on your platform. Compiled code is supported on 32-bit Windows, UNIX, and Power Macintosh.

On UNIX

On UNIX, you must have a supported compiler in your path to generate compiled code. You also need to add the path to the compiler's libraries to your library path environment variable (LIBPATH on AIX, SHLIB_PATH on HP-UX, or LD_LIBRARY_PATH on Solaris).

Use the `pbchkcom -v` command in a shell to find out whether you have a supported compiler. You need one of these compilers: C Set ++ for AIX Version 3.1.4, HP ANSI C++ compiler aCC version A.01.06, or Sun Sparc C++ 4.1 compiler.

Machine Code

Select Machine Code if you want to generate compiled code instead of Pcode. The remaining code generation options are only applicable when Machine Code is selected.

FOR INFO For more information about compiled code and Pcode, see the *Feature Guide* and *Application Techniques*.

Location of temporary files

The machine code generation process will put temporary files in a temporary directory, such as the TEMP directory on Windows and /tmp on UNIX. You can specify a different location in the [PB] section of your PowerBuilder initialization file with the CODEGENTEMP variable. You may want to do this if you have limited space on your local system.

For example:

```
CODEGENTEMP=d:\pbtempdir           // Windows
CODEGENTEMP=/net/big_server/pbtempdir // UNIX
CODEGENTEMP=AltDisk:PB temp folder // Macintosh
```

On UNIX, CODEGENTEMP must be in uppercase.

Trace Information

Select Trace Information if you want to create a trace file when you run your compiled code executable. You can use the trace file to troubleshoot or profile your application.

FOR INFO For more information on obtaining trace information, see "Tracing execution" on page 901.

Error Context Information

Select Error Context Information if you want PowerBuilder to display context information, such as object, event, and script line number, for runtime errors. This option is not available on UNIX.

Executable Format

On 32-bit Windows You can choose to build either a 32-bit (Native) executable or a 16-bit executable.

Managing mixed environments

If you have both 16-bit and 32-bit versions of a project, you should be aware that PowerBuilder doesn't recognize when the executable format has been changed. It will overwrite the previous build. Be sure to specify a full build whenever you switch between 16-bit and 32-bit executable formats.

On the Power Macintosh You can select one of three executable formats:

Select	To create an executable for
PowerPC	Power Macintosh
68K	68K Macintosh
Fat	Both Macintosh platforms

On UNIX The executable format is by default Native (32-bit), and this option is not available.

Optimization

On Windows and Macintosh You can build your application with no optimizations, or you can optimize for speed or space.

On UNIX You cannot change the level of optimization. PowerBuilder builds your application with a default set of optimizations.

Dynamic library options

You can reduce the size of your executable file by distributing some of the objects it requires in a dynamic library.

PBD or DLL

To define a library as a dynamic library to be distributed with your application, select the PBD or DLL checkbox. The label for this checkbox depends on whether you are building Pcode or machine code executable.

If you are generating Pcode, you will create PowerBuilder Dynamic Libraries (PBDs). If you are generating machine code, you will create platform-specific dynamic libraries.

FOR INFO For more about dynamic libraries, see "Using dynamic libraries" on page 883.

Resource File Name

For each of the dynamic libraries you create, you can specify a PowerBuilder resource file. You need to define a resource file for a dynamic library if it uses resources, such as bitmaps and icons, and you want the resources included in the dynamic library instead of having to distribute the resources separately.

FOR INFO For more about PowerBuilder resource files, see "Distributing resources" on page 891.

Using dynamic libraries

You can store the objects used in your distributed PowerBuilder application in more than one library and at execution time dynamically load any objects that are not contained in the application's executable file. This allows you to break the application into smaller units that are easier to manage and makes the executable file smaller. You do this by using dynamic libraries. If you compile using Pcode, PowerBuilder builds PowerBuilder dynamic libraries (PBD files). If you use machine code, PowerBuilder builds Dynamic Link Libraries (DLL files) on Windows and platform-specific shared libraries on Macintosh and UNIX.

Deploying on Windows 3.1

If you want to deploy a PBD with two different applications on Windows 3.1, and the PBD contains a user object that makes a global external function call, you need to compile and deploy the PBD separately with each application to ensure that the reference to the global function is correct.

Dynamic library names

PowerBuilder dynamic libraries are given the name of the PBL with the extension .pbd. For example, the Pcode library built from test.pbl is named test.pbd.

Machine-code dynamic libraries are given the extension .dll on Windows and UNIX. On the Macintosh, lib is appended to the name of the PBL. For example, the machine-code library built from test.pbl is named test.dll on Windows and UNIX, and testlib on the Macintosh.

Reducing the size of dynamic libraries

When PowerBuilder builds a dynamic library, it copies the compiled versions of *all objects* from the source library (PBL file) into the dynamic library. The easiest way to specify source libraries is to simply use your standard PowerBuilder libraries as source libraries.

However, then your dynamic libraries are larger than they need to be because they include all objects from the source library, not just the ones used in your application. You may want to create a PowerBuilder library that contains only the objects that you want to be in a dynamic library.

❖ To create the source library:

- 1 In the Library painter, place in one standard PowerBuilder library (a PBL file) all the objects that you want in the dynamic library.

If you need to create a new library, select Library>Create from the menu bar, then move the objects into the new library by selecting them and then selecting Entry>Move from the menu bar.

- 2 Make sure the application's library search path includes the new library.

Multiple dynamic libraries

You can use as many dynamic libraries as you want in an application. Just create a source library (PBL file) for each of them.

Specifying the dynamic libraries in your project

When you define your project, you tell PowerBuilder which of the libraries in the application's library search path will be dynamic.

❖ **To specify the dynamic libraries:**

- ◆ In the Project painter, select each of the libraries that will be dynamic.

Including additional resources for a dynamic library

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

❖ **To reference additional resources:**

- 1 List the resources in a PBR file as described in "Using PowerBuilder resource files" on page 892.
- 2 Use the Resource File Name box in the Project painter workspace to reference the PBR file in the dynamic library.

Building a project

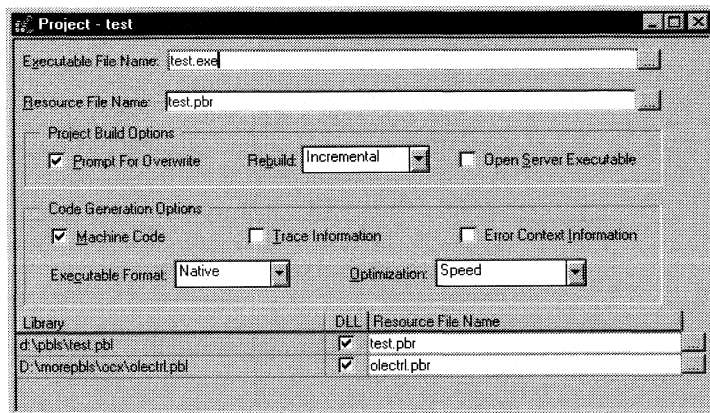
Once you have defined your project, you build it—that is, you tell PowerBuilder to create the executable files and all specified dynamic libraries. You can build your project anytime you have made changes to the objects and want to test or deploy another version of your application.

You can build a project only for your current application.

❖ To build the application:

- 1 Open the project you built in the Project painter.

The Project painter workspace displays.



- 2 Click the Build button in the Project painterBar.

If the application's library list has changed

When you click Build, PowerBuilder checks your application's library list. If it has changed since you defined your project, PowerBuilder updates the Project painter workspace with the new library list. Make whatever changes you need in the workspace, then click Build again.

PowerBuilder builds the executable and all specified dynamic libraries. This process is described in detail in the next section.

How PowerBuilder builds the project

Here is what happens when PowerBuilder builds your project:

- ◆ If you selected Rebuild: Full, PowerBuilder first regenerates all the objects in the libraries.
- ◆ If you selected Prompt for Overwrite, PowerBuilder displays a message box asking for confirmation before overwriting the executable file and each dynamic library.

To create the executable file you specified, PowerBuilder searches through your application and copies the compiled versions of *referenced* objects from the libraries in the application's library search path into the executable file. It does not copy objects that are *not referenced* in the application to the executable file.

Nor does it copy objects to the executable file from libraries you declared to be dynamic libraries. These objects are linked to the application at execution time; they are not stored in the executable file. Instead, PowerBuilder creates a dynamic library for each of the specified libraries and maintains a list of all the dynamic library files that were specified for the application (PowerBuilder maintains the unqualified filenames of the dynamic library files; it does not save the path name).

What happens during execution

When an object such as a window is referenced in the application, PowerBuilder first looks in the executable file for the object. If it doesn't find it there, it looks in the dynamic library files that are defined for the application. For example, if you specified that test.pbl is a dynamic library, PowerBuilder will look for test.pbd or test.dll (on Windows and UNIX) or testlib (on the Macintosh) at execution time. The dynamic library files must be in the search path. If PowerBuilder can't find the object in any of the dynamic library files, it reports an execution-time error.

How PowerBuilder searches for objects

When searching through the application, PowerBuilder doesn't necessarily find all the objects that are used in your application and copy them to the executable file. This section describes which objects it finds and copies and which ones it doesn't.

Which objects are copied to the executable file

PowerBuilder finds and copies the following objects to the executable file.

Objects that are directly referenced in scripts For example:

- ◆ If a window script contains the following statement:

```
Open(w_continue)
```

Then `w_continue` is copied to the executable file.

- ◆ If a menu item script refers to the global function `f_calc`:

```
f_calc(EnteredValue)
```

Then `f_calc` is copied to the executable file.

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

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

Then `m_new` is copied to the executable file.

Objects that are referenced in painters For example:

- ◆ If a menu is associated with a window in the Window painter, the menu is copied to the executable file as a result of the window being copied.
- ◆ If a `DataWindow` object is associated with a `DataWindow` control in the Window painter, the `DataWindow` object is copied to the executable file.
- ◆ If a window contains a custom user object that includes another user object, each user object is copied.
- ◆ Resources assigned in a painter. For example, when you place a `Picture` control in a window in the Window painter, you can associate a bitmap file with it.

Resources assigned this way are copied to the executable file.

On UNIX

On UNIX, when you create a compiled code executable for an application that makes external function calls, you can ignore any error messages like the following:

```
Wind/U Error(187): Function LoadLibrary, No DllMain  
found for DLL libname.dll
```

Which objects are not copied to the executable file

When creating the executable file, PowerBuilder can identify the associations you made in the painter (because those references are saved with the object's definition in the library) and direct references in scripts (the compiler saves this information).

But it does not identify objects that are referenced dynamically through string variables (it would have to read through all the scripts and process all assignment statements to uncover all the referenced objects).

For example:

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

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

Then `d_emp` is not copied to the executable file.

- ◆ If a resource is assigned dynamically in scripts, such as:

```
IF Balance < 0 THEN
    p_logo.PictureName = "frown.bmp"
ELSE
    p_logo.PictureName = "smile.bmp"
END IF
```

Then `FROWN.BMP` and `SMILE.BMP` are not copied to the executable file.

- ◆ If a window script has the following statements:

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

Then the reference to window `w_go` is *not* found by PowerBuilder when building the executable file, so `w_go` is not copied to the executable file.

Which objects are not copied to the dynamic libraries

When building a dynamic library, PowerBuilder does not inspect the objects; it simply removes the source form of the objects. Therefore, the resources (pictures, icons, and pointers) used by any of the objects in the library—*either specified in a painter or assigned dynamically in a script*—are not copied into the dynamic library.

How to include the objects that were not found

If you didn't use any of the types of references described in the preceding sections, you don't need to do anything else to ensure that all objects get distributed: they were all built into the executable file. Otherwise, you have the following choices for how to include the objects that were not found.

Distributing resources

For resources such as icons and bitmaps, you have two choices:

- ◆ Distribute them separately.
- ◆ Include them in a PowerBuilder resource file, then build an executable file or dynamic PowerBuilder library that uses the resource file.

Distributing DataWindow objects

For DataWindow objects, you have two choices:

- ◆ Include them in a PowerBuilder resource file, then build an executable file or dynamic PowerBuilder library using the resource file.
- ◆ Include them directly in a dynamic PowerBuilder library.

Distributing other objects

All other objects (such as windows referenced only in string variables) must be included directly in a dynamic PowerBuilder library.

Summary of distribution possibilities

Object	Separately	Through resource file	Directly in dynamic library
Resource	X	X	
DataWindow object		X	X
Other object			X

FOR INFO For information about distributing resources, see "Distributing resources" on page 891.

FOR INFO For information about dynamic PowerBuilder libraries, see "Project painter options" on page 878.

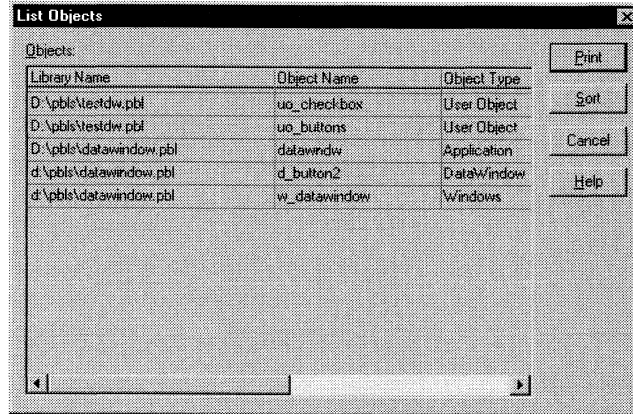
Listing the objects in a project

After you have built your project, you can display a list of objects in the project.

❖ **To list the objects in a project:**

- 1 Build your project.
- 2 Select Design>List Objects from the menu bar.

The List Objects dialog box displays.



The dialog box lists the objects that PowerBuilder placed in the executable file and dynamic libraries it created when it built the project.

What's in the report

The report is a grid DataWindow object with the following columns:

Column	Meaning
Library Name	Source library that contains the object
Object Name	Name of the object
Object Type	Type of object

Using version control

If you are using a version control system to manage your PowerBuilder objects, there are additional columns in the report, which map the objects in the project to your archive files.

FOR INFO For more information, see *Version Control Interfaces*.

What you can do

Because the report is a grid DataWindow object, you can resize and reorder columns just as you can in other grid DataWindow objects.

You can also sort the rows and print the report using the Sort and Print buttons.

Distributing resources

You can choose to distribute your resources (pictures, pointers, and icons) separately or include them in your executable file or dynamic library.

Distributing resources separately

If a resource has not been included in the executable file or in a dynamic library, when a resource is referenced at execution time PowerBuilder looks in the search path for the resource (such as the file FROWN.BMP). So you need to distribute the resources with your application and make sure they get installed in the user's search path.

For example, assume you use two bitmap files as in the following script:

```
IF Balance < 0 THEN
    p_logo.PictureName = "frown.bmp"
ELSE
    p_logo.PictureName = "smile.bmp"
END IF
```

You could distribute the files FROWN.BMP and SMILE.BMP with your application. As long as the files are on the search path at execution time, the application can load them when they are needed.

Windows search path

The Windows search path is as follows:

- 1 The current directory
- 2 The Windows directory
- 3 The Windows System directory
- 4 All directories in the PATH environment variable

Macintosh search path

The Macintosh search path is as follows:

- 1 The folder that contains your application
- 2 System Folder:Extensions:Powersoft 6.0
- 3 Any folder, alias of a folder, or alias of a volume in the folder that contains your application (going down one level only)

- UNIX search path The UNIX search path is as follows:
- 1 The current directory
 - 2 The user's home directory
 - 3 The directories in the user's PATH environment variable

Using PowerBuilder resource files

Instead of distributing resources separately, you can create a PowerBuilder resource file (a PBR file) that lists all dynamically assigned resources.

On Macintosh

On the Macintosh, you can use two types of resource files: PowerBuilder resource files and Macintosh resource files.

FOR INFO For more information about Macintosh resources, see "Macintosh resources" on page 895.

You can also include dynamically assigned DataWindow objects in PBR files. PowerBuilder compiles the listed resources into the executable file or a PBD file, so the resources are available directly at execution time.

Using dynamically assigned DataWindow objects

If you are assigning DataWindow objects to DataWindow controls dynamically in scripts, you *must* create a PowerBuilder resource file and include the objects in the executable or dynamic library file, or include the DataWindow objects themselves in a dynamic library file. You cannot distribute them separately (as you can, for example, bitmaps and cursors).

❖ To create and use a PowerBuilder resource file:

- 1 Using a text editor, create a text file that lists all resource files referenced dynamically in your application (see below for information about creating the file).

When creating a resource file for a dynamic library, list *all* resources used by the dynamic library, not just those assigned dynamically in a script.

- 2 Specify the resource files in the Project painter. The executable file can have a resource file attached to it as can each of the dynamic libraries.

When PowerBuilder builds the project, it includes all resources specified in the PBR file in the executable file or dynamic library. You no longer have to distribute your dynamically assigned resources separately; they are in the application.

Creating the PowerBuilder resource file

A PBR file is an ASCII text file in which you list resource names (such as BMP, CUR, ICO, RLE, and WMF files) and DataWindow objects. To create a PBR file, use a text editor. List the name of each resource, one resource on each line, then save the list as a file with the extension PBR. Here's a sample PBR file:

```
ct_graph.ico
document.ico
codes.ico
button.bmp
next1.bmp
prior1.bmp
```

Naming resources

If the resource file is in the current directory, you can simply list the file, such as:

```
FROWN.BMP
```

If the resource file is in a different directory, include the path to the file, such as:

```
C:\BITMAPS\FROWN.BMP
```

Important information about naming resources

The filename specified in the PBR file must exactly match the way the resource is referenced in scripts. If the reference in the script uses a path, you must specify the same path in the PBR file. If the resource file is not qualified with a path in the script, it must not be qualified in the PBR file.

For example, if the script reads:

```
p_logo.PictureName = "FROWN.BMP"
```

then the PBR file must read:

```
FROWN.BMP
```

If the PBR file says something like:

```
C:\MYAPP\FROWN.BMP
```

and the script doesn't specify the path, PowerBuilder will not find the resource at execution time. That is because PowerBuilder does a simple string comparison at execution time. In the preceding example, when PowerBuilder executes the script it will look for the object identified by the string "FROWN.BMP" in the executable file. It won't find it, because the resource is identified in the executable file as "C:\MYAPP\FROWN.BMP".

In this case, the picture will not display at execution time; the control will be empty in the window.

Including DataWindows objects in a PBR file

To include a DataWindow object in the list, enter the name of the library (with extension PBL) followed by the DataWindow object name enclosed in parentheses. For example:

```
sales.pbl(d_emplist)
```

If the DataWindow library is not in the directory that is current when the executable is built, fully qualify the reference in the PBR file. For example:

```
c:\myapp\sales.pbl(d_emplist)
```

Macintosh resources

On the Macintosh, your application can use two types of resources:

- ◆ PowerBuilder resources, which are the images used for picture buttons, pointers, and other graphic elements
- ◆ Macintosh resources, which hold information for the Finder about your application

PowerBuilder resources

You can use several different graphics formats for the images in your application. They include several Windows formats as well as PICT (the native Macintosh format). PICT files must be named in the format *name*.PCT. Otherwise, PowerBuilder does not recognize the format and does not display the image after you build your executable.

Clipped pointers and icons

On the Macintosh, pointers and icons are 16x16 pixels. In Windows, they are 32x32 pixels. Therefore, when you design pointers to be used on both platforms, use only the middle of the image. The rest won't be visible on the Macintosh.

When you build your application, these graphics elements can be in separate files that you deliver with your application or they can be included in the application's files. To include them in your application, you create a PowerBuilder resource file, which you name in the Application or Project painter. The resource file is a text file that lists the filenames of the resources.

Macintosh resources

Macintosh resources specify information about your application that the Finder needs. Your application's icon and memory are stored in Macintosh resources.

PowerBuilder supplies default versions of the required resources. After PowerBuilder creates your executable using the default versions, you can use ResEdit or another resource editor to make changes, if you want to.

You can also supply your own version of the resources in a resource file. When you build your application, you specify your Macintosh resource file in the Application painter or the Project painter and PowerBuilder uses your versions instead of its defaults.

Other Macintosh resources

PowerBuilder does not have functions for opening resource files and accessing individual resources and has no direct way for you to use Macintosh resources in your application. If you want to use resources in special ways, you will have to create a shared library to access the Macintosh Toolbox functions that manage resources.

If you include other resources in your Macintosh resource file, they will be copied into your application even though PowerBuilder can't access them.

PowerBuilder requires these resources:

Resource Type	Number	What it holds
BNDL	128	The ID of the icon family to be used as the Finder icons. The default is 128. You can change the value to point to another icon family The signature in the BNDL is ignored. The default signature is four question marks (????)
<i>owner</i>	0	The name of the resource is the same as the application's signature, which is stored in the BNDL resource. The contents of this resource are irrelevant
ICN#, icl8, icl4, ics#, ics8, ics4	128	The family of icons that the Finder uses in its various views. There are small and large icons and masks in color and black-and-white. If the resources in your icon family have an ID number other than 128, change the ID value in the BNDL resource
SIZE	-1	Various application behavior settings, as well as the amount of memory required by your application. You specify the Preferred size and Minimum size
DLOG	900	A Debug dialog displayed when you press the d key while starting an application. Uses the DITL 900 resource as its contents
DITL	900	Contents of the Debug dialog. Used by the DLOG 900 resource
cfrg	0	Contents of the Code Fragment names
ALRT	10000	An alert displayed when CFM-68K is not installed. Uses the DITL 10000 and STR# 10000 resources
DITL	10000	A pointer to the STR# 10000 resource that contains CFM messages. Used by the ALRT 10000 resource

Resource Type	Number	What it holds
STR#	10000	The text of the messages displayed when CFM-68K is not installed. Used by the DITL 10000 and ALRT 10000 resources
FREF	128	The file type (always APPL) and local ID of the icon family used for Finder icons (always 0). The filename setting is not used
vers	1, 2	Version information for the Finder, including the version string shown in your application's Get Info box
CODE		The application's code. This resource is created when you build your application and should not be altered

Setting up your Macintosh resource file

If you want to supply your own resource definitions, use the information in this section to specify the required information. The resources you can supply are described in detail here. You don't have to supply your own version of every type—only the ones for which you want to set your own values. PowerBuilder will supply its default version for any resource you leave out.

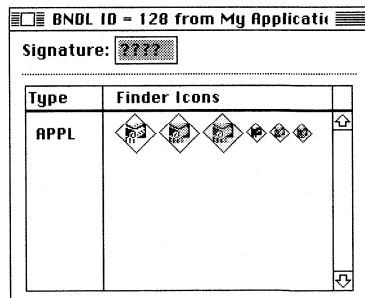
BNDL

All applications require a BNDL resource with a resource ID of 128. It points to your application's Finder icons.

Your application's signature

You specify a signature when you create the executable in the Mac Creator Type edit box. The signature in the BNDL resource is ignored.

To select the icon family for your application, specify its ID number in the BNDL. If you are using ResEdit, open the BNDL resource and choose BNDL>Choose Icon to select another icon family.



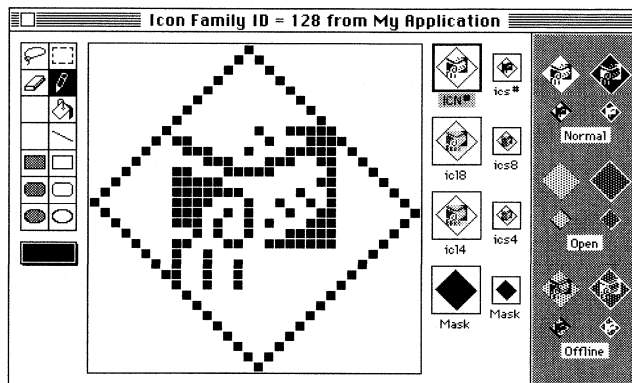
If your application creates its own file types, add those types to this BNDL resource. You can also add icons for the new file types or specify your own icons for standard types, such as TEXT.

Owner

The owner resource is a resource whose name is the same as your application's signature. Only the name matters, not its contents. It simply has to exist.

ICN#, icl8, icl4, ics#, ics8, ics4

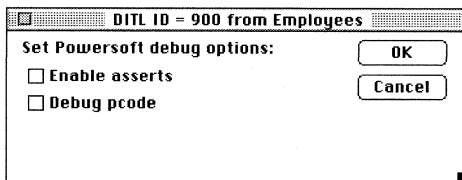
You can create your own set of Finder icons by defining an icon family. An icon family consists of six types of icons. These six cover the various situations for which the Finder needs icons: black-and-white, 8-bit color, 4-bit color, each in a large and small version. When six resources, one of each of the six types, all have the same resource ID, you have an icon family. ResEdit has a template for editing all six types so you can see how they look together.



DITL, DLOG, and ALERT

Only a few of PowerBuilder's messages are stored in resources.

A Debug dialog box is available to help you debug your executables. To use the Debug dialog box, press the D key when you start your application. The DLOG resource numbered 900 contains the dialog and its DITL resource numbered 900 contains the contents.



The resources numbered 10000 are error messages for CFM. The ALERT resource is the window. The DITL resource is a template for the alert's contents. The STR# resource numbered 10000 contains the text of the error messages (see STR# below).

FREF

In a FREF resource numbered 128, specify:

- ◆ Your application's file type, which is always APPL.
- ◆ An ID of 0 for your icon family's local ID. This ID is mapped internally to the icon family ID in the BNDL resource.

You do not need to specify a value for FileName.

The screenshot shows a window titled "FREF ID = 128 from My Application". It contains three input fields: "File Type" with the text "#APPL", "Icon localID" with the text "0", and "FileName" which is currently empty.

STR#

The text of the CFM messages is stored in a STR# resource numbered 10000. If your application is being translated to another language or if you want to change the wording, you can edit the messages in this resource.

vers

Two vers resources, numbered 1 and 2, supply information that the Finder displays in your application's Get Info box. The long version string of vers resource 1 is displayed as the Version label in the Get Info box. The long version string of vers resource 2 is displayed as a subtitle under the application name in the Get Info box.

The screenshot shows a window titled "vers ID = 1 from my application". It contains several fields: "Version number" with the value "2 . 0 . 1", "Release:" with a dropdown menu showing "Beta", "Non-release:" with the value "0", "Country Code:" with a dropdown menu showing "00 - USA", "Short version string:" with the value "2.0.1", and "Long version string (visible in Get Info):" with the value "Release 2.0.1 Build 33".

users 10 = 2 from my application

Version number: . .

Release: Non-release:


Country Code:

Short version string:

Long version string (visible in Get Info):

The long version strings shown above are displayed in this Get Info box.

my application Info

 my application
Build 33

Kind: application program
Size: 221K on disk (175,316 bytes used)

Where: Evita: tempkotwal:

Created: Mon, Sep 23, 1996, 8:10 AM
Modified: Mon, Sep 23, 1996, 8:38 AM
Version: Release 2.0.1 Build 33

Comments:

Memory Requirements

Suggested size:	4096	K
Minimum size:	2048	K
Preferred size:	4096	K

Locked

Note: Memory requirements will increase by 162K if virtual memory is turned off in the Memory control panel.

What happens at execution time

When a resource such as a bitmap is referenced at execution time, PowerBuilder first looks in the executable file for it. Failing that, it looks in the PBDs that are defined for the application. Failing that, it looks in directories in the search path for the file.

Tracing execution

You can trace execution of an executable file built with PowerBuilder. By tracing execution, you can troubleshoot your application if it doesn't behave the same way when run as an executable file as it does when run in the PowerBuilder development environment. You can also use the trace output to profile your application: for example, you can see how many times particular scripts and functions are being executed.

You can generate two kinds of trace files:

- ◆ **With timing information** You collect trace information by adding code to the scripts in the application or adding a window that lets users turn tracing on and off. PowerBuilder generates a binary trace file that you analyze using a comprehensive set of objects and functions or the Application Profiler utility.

FOR INFO For more information about tracing and profiling, see "Tracing and profiling an application" on page 851.

- ◆ **Without timing information** You collect information by running the application with the `/pbdebug` command-line switch. PowerBuilder generates a text file that logs the creation and destruction of objects and the execution of scripts and functions.

FOR INFO For more information about PBDebug tracing, see "Generating a trace file without timing information" on page 870.

Tracing execution
using `/pbdebug`

The way you generate PBDebug trace information for an executable file depends on your platform.

❖ To generate PBDebug trace information on Windows or UNIX:

- ◆ Invoke the executable file using the `/pbdebug` command-line switch:

```
EXEFILE /pbdebug
```

On UNIX, you can also use `-pbdebug` instead of `/pbdebug`.

❖ To generate PBDebug trace information on the Macintosh:

- ◆ Hold down the D key when you invoke the executable and select Debug Pcode from the Set Powersoft debug options dialog box.

As the application executes, PowerBuilder records the trace output in a file called *EXEFILE*.DBG, which is a text file that you can read in any editor.

Enabling tracing

If you are compiling machine code, you must enable tracing at compile time by selecting Trace Information in the Project painter Compile Options group. If you haven't enabled tracing when you compile for machine code, no trace information is generated and the /pbdebug switch has no effect. Tracing machine code execution is not currently supported on UNIX.

If you compile your project in Pcode, the compiler automatically adds the information needed to enable tracing.

FOR INFO For information on creating a trace file when you run your application within PowerBuilder, see Chapter 25, "Debugging and Running Applications".

A P P E N D I X

This appendix describes the Powersoft repository.

The Powersoft Repository

About this appendix

This appendix describes each column in each table in the Powersoft repository.

Contents

Topic	Page
About the Powersoft repository	906
The PBCatTbl table	907
The PBCatCol table	909
The PBCatFmt table	910
The PBCatVld table	911
The PBCatEdt table	912

About the Powersoft repository

PowerBuilder stores application-based information you provide for a database table (such as the text to use for labels and headings for the columns, validation rules, display formats, and edit styles) in system tables in your database. These system tables are called the Powersoft **repository**. The repository tables contain all the information related to the extended attributes for the tables and columns in the database. These extended attributes are used in DataWindow objects.

The system tables

There are five Powersoft system tables:

Table	Contains information about
PBCatTbl	Tables in the database
PBCatCol	Columns in the database
PBCatFmt	Display formats
PBCatVld	Validation rules
PBCatEdt	Edit styles

What to do with the tables

You can open and look at these tables in the Database painter just like other tables. You might want to create a report of the repository information used in your database by building a DataWindow object whose data source is the repository tables.

Caution

You should not change the values in the repository tables. PowerBuilder maintains this information automatically whenever you change information for a table or column in the Database painter.

The PBCatTbl table

Column	Column name	Description
1	pbt_tnam	Table name
2	pbt_tid	SQL Server Object ID of table (used for SQL Server only)
3	pbt_ownr	Table owner
4	pbd_fhgt	Data font height, PowerBuilder units
5	pbd_fwgt	Data font stroke weight (400=Normal, 700=Bold)
6	pbd_fitl	Data font Italic (Y=Yes, N=No)
7	pbd_funl	Data font Underline (Y=Yes, N=No)
8	pbd_fchr	Data font character set (0=ANSI, 2=Symbol, 255=OEM)
9	pbd_fptc	Data font pitch and family (see note)
10	pbd_ffce	Data font typeface
11	pbh_fhgt	Headings font height, PowerBuilder units
12	pbh_fwgt	Headings font stroke weight (400=Normal, 700=Bold)
13	pbh_fitl	Headings font Italic (Y=Yes, N=No)
14	pbh_funl	Headings font Underline (Y=Yes, N=No)
15	pbh_fchr	Headings font character set (0=ANSI, 2=Symbol, 255=OEM)
16	pbh_fptc	Headings font pitch and family (see note)
17	pbh_ffce	Headings font typeface
18	pbl_fhgt	Labels font height, PowerBuilder units
19	pbl_fwgt	Labels font stroke weight (400=Normal, 700=Bold)
20	pbl_fitl	Labels font Italic (Y=Yes, N=No)
21	pbl_funl	Labels font Underline (Y=Yes, N=No)
22	pbl_fchr	Labels font character set (0=ANSI, 2=Symbol, 255=OEM)

Column	Column name	Description
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=Don't Care, 16=Roman, 32=Swiss, 48=Modern, 64=Script, 80=Decorative

The PBCatCol table

Column	Column name	Description
1	pbctnam	Table name
2	pbctid	SQL Server Object ID of table (used for SQL Server only)
3	pbctownr	Table owner
4	pbctcnam	Column name
5	pbctcid	SQL Server Column ID (used for SQL Server only)
6	pbctlabl	Label
7	pbctlpos	Label position (23=Left, 24=Right)
8	pbcthdr	Heading
9	pbcthpos	Heading position (23=Left, 24=Right, 25=Center)
10	pbctjtfy	Justification (23=Left, 24=Right)
11	pbctmask	Display format name
12	pbctcase	Case (26=Actual, 27=UPPER, 28=lower)
13	pbcthght	Column height, PowerBuilder units
14	pbctwdth	Column width, PowerBuilder units
15	pbctptrn	Validation rule name
16	pbctbmap	Bitmap/picture (Y=Yes, N=No)
17	pbctinit	Initial value
18	pbctcmnt	Column comments
19	pbctedit	Edit style name
20	pbcttag	(Reserved)

The PBCatFmt table

Column	Column name	Description
1	pbf_name	Display format name
2	pbf_frmt	Display format
3	pbf_type	Data type to which format applies
4	pbf_cntr	Concurrent-usage flag

The PBCatVld table

Column	Column name	Description
1	pbv_name	Validation rule name
2	pbv_vald	Validation rule
3	pbv_type	Data type to which validation rule applies
4	pbv_cntr	Concurrent-usage flag
5	pbv_msg	Validation error message

The PBCatEdt table

Column	Column name	Description
1	pbe_name	Edit style name
2	pbe_edit	Format string (edit style type dependent; see below)
3	pbe_type	Edit style type (see below)
4	pbe_cntr	Revision counter (increments each time edit style is altered)
5	pbe_seqn	Row sequence number for edit types requiring more than one row in PBCatEdt table
6	pbe_flag	Edit style flag (edit style type dependent; see below)
7	pbe_work	Extra field (edit style type dependent; see below)

Available edit style types

The following edit style types are available:

Edit style type	pbe_type value (column 3)
CheckBox	85
RadioButton	86
DropDownListBox	87
DropDownDataWindow	88
Edit	89
Edit Mask	90

CheckBox edit style (code 85)

Here is a sample row in the PBCatEdt table for a CheckBox edit style:

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>Text</i>	85	1	1	<i>Flag</i>	
MyEdit	<i>OnValue</i>	85	1	2	0	

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>OffValue</i>	85	1	3	0	
MyEdit	<i>ThirdValue</i>	85	1	4	0	

where:

Value	Meaning
<i>Text</i>	CheckBox text
<i>OnValue</i>	Data value for On state
<i>OffValue</i>	Data value for Off state
<i>ThirdValue</i>	Data value for Third state (this row exists only if 3 State is checked for the edit style—bit 30 of <i>Flag</i> is 1)
<i>Flag</i>	<p>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</p> <p>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</p>

RadioButton edit style (code 86)

Here is a sample row in the PBCatEdt table for a RadioButton edit style:

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>Columns</i>	86	1	1	<i>Flag</i>	
MyEdit	<i>Display1</i>	86	1	2	0	
MyEdit	<i>Data1</i>	86	1	3	0	
MyEdit	<i>Display2</i>	86	1	4	0	
MyEdit	<i>Data2</i>	86	1	5	0	

where:

Value	Meaning
<i>Columns</i>	Character representation (in decimal) of number of columns (buttons) across
<i>Display1</i>	Display value for first button
<i>Data1</i>	Data value for first button
<i>Display2</i>	Display value for second button
<i>Data2</i>	Data value for second button
	<hr/> <p>About the display and data values Display and data values are repeated in pairs for each radio button defined in the edit style</p> <hr/>
<i>Flag</i>	32-bit flag. Low-order four hex digits are generic edit type; high-order four are styles within the type. A 1 in any bit indicates the corresponding style is checked. A 0 in any bit indicates the corresponding style is unchecked Bit 31: Left Text Bit 30: 3D Bit 29: Scale Circles Bit 38: Not used (set to 0) Bits 27 – 16 (3 hex digits): Not used (set to 0) Bits 15 – 4 (3 hex digits): Always 0 for RadioButton edit style Bit 3: Always 1 for RadioButton edit style Bit 2: Always 0 for RadioButton edit style Bit 1: Always 0 for RadioButton edit style Bit 0: Always 0 for RadioButton edit style

DropDownListBox edit style (code 87)

Here is a sample row in the PBCatEdt table for a DropDownListBox edit style:

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>Limit</i>	87	1	1	<i>Flag</i>	<i>Key</i>
MyEdit	<i>Display1</i>	87	1	2	0	
MyEdit	<i>Data1</i>	87	1	3	0	
MyEdit	<i>Display2</i>	87	1	4	0	
MyEdit	<i>Data2</i>	87	1	5	0	

where:

Value	Meaning
<i>Limit</i>	Character representation (in decimal) of the <i>Limit</i> value
<i>Key</i>	One-character accelerator key
<i>Display1</i>	Display value for first entry in code table
<i>Data1</i>	Data value for first entry in code table
<i>Display2</i>	Display value for second entry in code table
<i>Data2</i>	Data value for second entry in code table
	<p>About the display and data values</p> <p>Display and data values are repeated in pairs for each entry in the code table</p>
<i>Flag</i>	<p>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</p> <p>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</p>

DropDownDataWindow edit style (code 88)

Here is a sample row in the PBCatEdt table for a DropDownDataWindow edit style:

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>DataWin</i>	88	1	1	<i>Flag</i>	<i>Limit</i>
MyEdit	<i>DataCol</i>	88	1	2	0	<i>Key</i>
MyEdit	<i>DisplayCol</i>	88	1	3	0	<i>Width%</i>

where:

Value	Meaning
<i>DataWin</i>	Name of DataWindow object to use
<i>DataCol</i>	Data column from DataWindow object
<i>DisplayCol</i>	Display column from DataWindow object
<i>Limit</i>	Character representation (in decimal) of <i>Limit</i> value
<i>Key</i>	One-character accelerator key
<i>Width%</i>	Width of the dropdown part of the DropDownDataWindow in %
<i>Flag</i>	<p>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</p> <ul style="list-style-type: none"> Bit 31: Allow editing Bit 30: Auto HScroll Bit 29: VScroll bar Bit 28: Always show list Bit 27: Uppercase Bit 26: Lowercase (if bits 27 and 26 are both 0, then case is Any) Bit 25: HScroll bar Bit 24: Split horizontal scroll bar Bit 23: Empty string is NULL Bit 22: Required field Bit 21: Always show arrow Bit 20: Not used (set to 0) Bits 19 – 16 (1 hex digit): Not used (set to 0) Bits 15 – 8 (2 hex digits): Always 0 for DropDownDataWindow edit style Bit 7: Always 0 for DropDownDataWindow edit style Bit 6: Always 0 for DropDownDataWindow edit style Bit 5: Always 0 for DropDownDataWindow edit style Bit 4: Always 1 for DropDownDataWindow edit style Bit 3 – 0 (1 hex digit): Always 0 for DropDownDataWindow edit style

Edit edit style (code 89)

Here is a sample row in the PBCatEdt table for an Edit edit style:

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>Limit</i>	89	1	1	<i>Flag</i>	<i>Key</i>
MyEdit	<i>Format</i>	89	1	2	0	<i>Focus</i>
MyEdit	<i>Display1</i>	89	1	3	0	
MyEdit	<i>Data1</i>	89	1	4	0	
MyEdit	<i>Display2</i>	89	1	5	0	
MyEdit	<i>Data2</i>	89	1	6	0	

About the example

This example shows an Edit edit style using a code table of display and data values. There is a pair of rows in PBCatEdt for each entry in the code table *only if* bit 23 of *Flag* is 1.

FOR INFO For information about code tables in edit styles, see Chapter 16, "Displaying and Validating Data".

where:

Value	Meaning
<i>Limit</i>	Character representation (in decimal) of <i>Limit</i> value
<i>Key</i>	One-character accelerator key
<i>Format</i>	Display format mask
<i>Focus</i>	Character "1" if Show Focus Rectangle is checked. NULL otherwise

Value	Meaning
<i>Flag</i>	<p>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</p> <p>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</p>

Edit Mask edit style (code 90)

Here is a sample row in the PBCatEdt table for an Edit Mask edit style:

Name	Edit	Type	Cntr	Seqn	Flag	Work
MyEdit	<i>Format</i>	90	1	1	<i>Flag</i>	<i>DtFcKy</i>
MyEdit	<i>Range</i>	90	1	2	0	<i>SpinInc</i>
MyEdit	<i>Display1</i>	90	1	3	0	
MyEdit	<i>Data1</i>	90	1	4	0	
MyEdit	<i>Display2</i>	90	1	5	0	
MyEdit	<i>Data2</i>	90	1	6	0	

About the example

This example shows an Edit Mask edit style using a code table of display and data values as part of a spin control. Rows 2 and beyond exist in PBCatEdt only if the edit mask is defined as a spin control (bit 29 of *Flag* is 1). Rows 3 and beyond exist only if the optional code table is populated.

FOR INFO For information about using an edit mask as a spin control, see Chapter 16, "Displaying and Validating Data".

where:

Value	Meaning
<i>Format</i>	Display format mask
<i>DtFcKy</i>	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 DateTime = "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"
<i>Range</i>	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
<i>SpinInc</i>	Character representation (in decimal) of spin increment
<i>Display1</i>	Display value for first entry in code table
<i>Data1</i>	Data value for first entry in code table
<i>Display2</i>	Display value for second entry in code table

Value	Meaning
<i>Data2</i>	Data value for second entry in code table
	<p>About the display and data values Display and data values are repeated in pairs for each entry in the code table</p>
<i>Flag</i>	<p>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</p> <ul style="list-style-type: none"> Bit 31: Required Bit 30: Autoskip Bit 29: Spin control Bit 28: Read only (code table option) Bit 27: Use code table Bit 26: Not used (set to 0) Bit 25: Not used (set to 0) Bit 24: Not used (set to 0) Bit 23 – 16 (2 hex digits): Not used (set to 0) Bit 15 – 8 (2 hex digits): Always 0 for Edit Mask edit style Bit 7: Always 0 for Edit Mask edit style Bit 6: Always 0 for Edit Mask edit style Bit 5: Always 1 for Edit Mask edit style Bit 4: Always 0 for Edit Mask edit style Bits 3 – 0 (1 hex digit): Always 0 for Edit Mask edit style

Index

Symbols

- + operator 529
- @
 - used in crosstabs 734
 - used in validation rules 586

Numerics

- 24-hour times 564

A

- accelerator keys
 - and CheckBox edit style 572
 - and RadioButton edit style 573
 - assigning to menu items 272
 - defining 214
 - indicating, in StaticText controls 227
- access level
 - changing in function 143
 - of functions 136
 - of object-level structures 154
- access, to database 398
- activity log 348
- AddItem function 231
- aggregate functions, in graphs 697
- Alert box 167
- alignment
 - in DataWindow objects 516
 - of command buttons 222
 - of controls 208
- alignment extended attribute 364
- alignment for paragraphs 759
- Alignment Grid dialog box 207
- ALRT resources 898
- ALT key
 - and menu items 272, 273
 - defining accelerator keys 214
- Alter Table dialog box
 - altering table definition 369
 - applying display formats 557
 - applying edit styles 569
 - Font button 370
 - specifying extended attributes 365
- ALTER TABLE statement 377
- ampersand (&)
 - defining accelerator keys 214
 - in Menu object text 272
- ancestors
 - objects 249, 251
 - scripts 254
 - windows 191
- AND operator, in Quick Select 438
- AppleScript, executing scripts 29
- application icon 178
- application library search path 57
- Application objects
 - creating new 45
 - definition 44
 - displaying structure of 50
 - events, list of 62
 - properties 63
 - storing 45
- Application painter
 - displaying application structure 50
 - opening 45, 48
- Application profiler 852
- application templates, creating 49
- applications
 - building, basic steps 37
 - creating new 45, 49
 - definition 44
 - displaying structure of 50
 - library search path 68
 - MDI design 162
 - running 844
 - signature 879

- specifying properties 54
- windows 162, 165
- area graphs
 - about 684
 - making three-dimensional 687
- arguments
 - adding, deleting, and reordering in functions 144
 - changing for function 143
 - defining for function 136
 - passing by reference 137
 - passing by value 137
 - passing in function 137
 - passing structures as in functions 155
 - referencing retrieval 450
 - using in pipelines 799
 - using retrieval 448
- arrays
 - declaring argument as in function 137
 - in retrieval arguments 449
- ASCII text and rich text 752
- asterisks (*)
 - displaying user input as 570
 - in text boxes 227
 - wildcard character 72
- audience for this book xxi
- Auto Size setting, in graphs 712
- autoincrement columns, in DataWindow objects 545
- autoinstantiating user objects 304, 311
- Autosize Height
 - bands 507
 - with nested reports 678
- average, computing 530
- axes
 - scaling 718
 - specifying line styles 721
 - specifying properties in graphs 716
 - specifying text properties 711
 - using major and minor divisions 719
- B**
- Background Color dropdown toolbar 472
- background colors
 - in three-dimensional controls 220
 - setting 497
- background.color property, on the Expressions property page
 - about 636
 - specifying colors 657
- bands
 - in DataWindow painter workspace 469
 - resizing in DataWindow painter 478
 - setting colors for 498
- bar graphs
 - about 684
 - making three-dimensional 687
 - specifying overlap and spacing 716
- base class objects 249
- base reports 661
- BAT files 27
- bind variables, used with nested reports 675
- bitmaps
 - in rich text 762
 - specifying column as 358
- blob columns
 - creating 786
 - making visible 790
- blob data
 - creating columns for 786
 - saving in databases 791
- blobs, adding to DataWindow objects 536
- BMP files
 - adding to DataWindow objects 524
 - and Picture controls 235
 - and PictureBox controls 223
 - in rich text 762
 - naming in resource files 893
- BNDL resource 879, 897
- boolean expressions
 - in filters 594
 - in validation rules 585, 588
- border property, on the Expressions property page 63
- borders
 - around default command buttons 223
 - for text boxes 227
 - in DataWindow objects 506
 - in graphs 721
 - three-dimensional 220
- Borders dropdown toolbar 472
- boundaries, displaying in DataWindow painter 512
- breaks, in grouped DataWindow objects 600

- Browse Class Hierarchy command 81
 - Browse property sheets 118
 - Browser
 - about 118
 - class hierarchies 81
 - pasting functions 146
 - pasting structures with 156
 - regenerate descendants 93
 - brush.color property, on the Expressions property page
 - about 638
 - specifying colors 657
 - brush.hatch property, on the Expressions property page 638
 - built-in functions
 - for Menu objects 281
 - for windows and controls 186
 - including in user-defined functions 139
 - business cards 423
 - buttons
 - adding to DataWindow objects 532
 - adding to toolbars 20
 - creating for visual user objects 309
 - custom 23
 - deleting from toolbar 21
 - Function painter 132
 - moving on toolbar 21
 - PowerScript painter 108
 - Structure painter 149
 - Window painter 169
- C**
- caching data
 - in DataWindows 481
 - turning off 482
 - calling
 - ancestor scripts 256
 - passing arguments 137
 - user-defined functions 146
 - cancel command button 223
 - cascading menus 265, 268
 - case
 - converting in DataWindow objects 570
 - of text boxes 227
 - sensitivity and code tables 580
 - categories, graph
 - basics 683
 - specifying 696
 - Category axis, graph 683
 - centering controls 208
 - ChangeMenu function 292
 - Check Out Library Entries dialog box 86
 - check out status 73
 - CheckBox controls 201, 226
 - CheckBox edit style, defining 572
 - Checked property 269, 271
 - check-in/check-out 83
 - child windows
 - about 164, 165
 - specifying window type 173
 - CHOOSE CASE statements 119
 - Class browser 250
 - class hierarchies 81
 - class user objects
 - creating instance of 310
 - custom 297
 - overview 296
 - standard 296
 - using 310
 - ClearValues function 571
 - Clicked events, for Menu objects 280
 - clipboard
 - copying data to 495
 - using 28, 113
 - clipping pointers and icons 895
 - Close events, in Application object 44
 - CloseUserObject function 310
 - CloseWithReturn function, passing parameters between windows 186
 - code
 - checking out and in 83
 - inserting comments 28, 113
 - inserting into scripts 122
 - recompiling 93
 - saving in a file 139
 - user-defined functions 138
 - using structures 153
 - code generation options 880
 - code tables
 - about 578
 - defining 578

- in Specify Retrieval Criteria dialog box 520
 - modifying during execution 571
 - processing 580
 - using display values in crosstabs 732
 - using display values in graphs 696
 - using in dropdown listboxes 571
- Color dropdown toolbar
- adding custom colors to 219
 - in Window painter 218
- color property, on the Expressions property page
- about 641
 - specifying colors 657
- colors
- assigning to controls 218
 - background, with 3D controls 220
 - changing in Database painter 348, 375
 - customizing 219
 - default 55
 - for DataWindow objects 497
 - in display formats 559
 - in Select painter 444
 - of inherited script icon 253
 - specifying for windows 175
- Column Display Format dialog box 556
- column graphs
- about 684
 - specifying overlap and spacing 716
- column names 506
- Column Object Validation tab page 588
- Column Validation dialog box 584
- columns
- adding to DataWindow objects 521
 - appending to table 367
 - applying display formats to 557
 - applying edit styles to 569
 - applying validation rules to 584
 - defining display formats 556, 557
 - defining edit styles 568, 569
 - defining validation rules 584, 588
 - displaying as a dropdown DataWindow 575
 - displaying as checkboxes 572
 - displaying as dropdown listboxes 571
 - displaying as radio buttons 573
 - displaying with fixed formats 573
 - foreign key 378
 - formatting in DataWindow objects 554
 - formatting in reports 554
 - graphing data in 689, 694
 - initial values 587
 - named in DataWindow painter workspace 471
 - presenting in DataWindow objects 566
 - presenting in reports 566
 - preventing updates in DataWindow objects 543, 546
 - removing display formats 557
 - removing edit styles 569
 - removing validation rules 585
 - reordering in Grid style 494
 - resizing in Grid style 494
 - restricting input 573
 - selecting in Select painter 444
 - sliding to remove blank space 517
 - specifying extended attributes 364
 - specifying for crosstabs 732
 - updatable, in DataWindow objects 543, 546
 - validating input in DataWindow objects 582
 - validating input in forms 582
 - variable length 507
- command key 273
- CommandButton controls
- changing color of 218
 - defining accelerator keys for 214
 - prefix 201
 - setting a default 223
 - using 222
- comment extended attribute 364
- comments
- displaying in Library painter 73
 - in menu definition 276
 - in scripts 28, 113
 - in window definition 181
 - including in SQL statements 400
 - of saved objects, modifying 73
 - updating 74
- communication
- between user objects and windows 314
 - using user events 315
- Compile menu 254
- compiling
- on import 99
 - regenerating library entries 93
 - scripts 123

- user-defined functions 140
- Composite presentation style
 - about 660
 - limitations 663
 - using 664
- composite reports
 - about 660
 - creating 664
 - limitations 663
 - specifying footer position 679
 - starting on new page 679
- computed columns, including in SQL Select 445
- computed fields
 - adding to DataWindow objects 525
 - creating from toolbar 24
 - defining 527
 - defining custom buttons for 531
 - in crosstabs 741
 - specifying display formats 557
 - summary statistics 530
- conditional modification
 - example, gray bar 624
 - example, highlighting rows 629
 - example, rotating objects 627
 - example, size and location 632
 - modifying objects 622
- Constructor event 302
- context-sensitive help 115
- continuous data, graphing 684
- control-level properties 186
- controls
 - adding to windows 181, 197
 - aligning to grid 207
 - assigning colors to 218
 - calling ancestor scripts for 256
 - changing names 202
 - copying in Window painter 210
 - declaring events 322
 - defining properties 199
 - deriving user objects from 295, 299
 - in descendent objects 251, 308
 - moving and resizing 207
 - naming 201
 - referring to, in scripts 283
 - selecting 198
 - specifying accessibility of 217
 - with events, list of 196
- count
 - computing 530
 - in graphs 697
- Create Database command 351
- Create Index dialog box 381
- Create Library dialog box 75
- Create Local Database dialog box 351
- Create Table dialog box
 - Font button 370
 - in Database painter 361, 362
 - specifying extended attributes 365
- CREATE TABLE statement 377
- CREATE VIEW statement 386
- creator type, changing 879
- Crosstab Definition dialog box 731
- crosstabs
 - associating data 731
 - basic properties 738
 - changing column and row labels 740
 - changing definition of 739
 - creating 730
 - defining summary statistics 741
 - dynamic 728
 - functions 743
 - grid lines in 738
 - modifying data 739
 - overview 726
 - previewing 737
 - property conditional expressions in 748
 - specifying columns 732
 - specifying multiple columns and rows 735
 - static 728, 747
 - using expressions 733
 - using ranges of values 744
- ctrl key 273
- CUR files
 - naming in resource files 893
 - selecting mouse pointers 177, 499
- currency display format 558
- custom class user objects
 - about 297
 - writing scripts for 304
- custom colors 219
- custom events, PowerBuilder 323
- custom MDI frames 168

- custom visual user objects
 - about 295
 - building 300
 - writing scripts for 310
- Customize dialog box 21

D

data

- associating with graphs in DataWindow objects 694
 - caching in DataWindows 481
 - changing 391, 483
 - copying to clipboard 495
 - default properties 55
 - formatting in DataWindow objects 554
 - formatting in reports 554
 - importing 395, 456, 485
 - pipng 794
 - presenting in DataWindow objects 566
 - presenting in Freeform style 421
 - presenting in Grid style 422
 - presenting in Label style 422
 - presenting in N-Up style 424
 - presenting in reports 566
 - presenting in Tabular style 421
 - printing 486
 - retrieving and updating 391, 541
 - retrieving in DataWindows 480, 482
 - retrieving in reports 482
 - saving in external files 396, 488
 - saving in HTML Table format 489
 - storing in DataWindow objects 539
 - updating, controlling 543
 - validating in DataWindow objects 582
 - validating in forms 582
- data entry forms 421
- Data Manipulation painter
 - about 340
 - closing 397
 - opening 390
 - printing 396
 - sorting rows 392
- Data Pipeline painter
 - about 340, 794
 - working in workspace 801

- Data Retained On Save dialog box 539
- data source
 - defining for DataWindow objects and reports 430
 - modifying 509
 - Query 455
 - Quick Select 431
 - SQL Select 441
 - Stored Procedure 457
- data types
 - in display formats 559
 - in graphs 717
 - of arguments 137
 - of blob columns 786
 - of return values 134
 - of structure variables 150
 - pastng into scripts 118
 - when piping data 795
- data validation
 - in code tables 580
 - with validation rules 582
- data values
 - in graphs 697
 - of code tables 578
 - specifying fonts in tables 370
 - using in graphs 696
- database access 398
- database administration
 - database access 398
 - executing SQL 399
 - painting SQL 399
 - security 398
- Database Administration painter
 - about 340, 398
 - button 398
 - using 398
- Database Blob Object dialog box 787
- database errors 125
- Database painter
 - about 339
 - accessing Database Administration painter 398
 - changing colors in 348
 - creating OLE columns 786
 - creating tables 360
 - defining display formats 556
 - defining validation rules 584
 - opening 342

- previewing data 390
- refreshing table list 353
- specifying extended attributes 364
- working with edit styles 568
- workspace 343
- database profiles in pipelines 798
- databases
 - accessing through Quick Select 431
 - accessing through SQL Select 441
 - administering 398
 - changing 342
 - connecting to 417
 - controlling access to 398
 - controlling updates to 543
 - creating and deleting local SQL Anywhere 351
 - creating tables 360
 - destination in pipelines 794
 - ensuring referential integrity 374
 - executing SQL statements 403
 - importing data 395, 485
 - limiting retrieved data 594
 - logging work 348
 - piping data 794
 - retrieving, presenting, and manipulating data 390, 406
 - source in pipelines 794
 - specifying fonts 370
 - stored procedures 457
 - storing blob objects in 785
 - system tables 354
 - updating 391, 483
 - using as data source in DataWindow object or report 430
- DataWindow controls
 - placing in windows 197
 - prefix 201
- DataWindow objects 751
 - adding objects 521
 - aligning objects 516
 - and graphs in 689
 - blobs, adding 536
 - borders in 506
 - buttons, adding 532
 - changing margins 486
 - columns, adding 521
 - Composite presentation style 660
 - computed fields, adding 525
 - computed fields, defining 527
 - controlling updates in 543
 - creating new 417
 - creating OLE columns 786
 - custom buttons that add computed fields 531
 - data source, modifying 509
 - data sources 430
 - data, storing in 539
 - defaults 428, 496
 - display formats 554
 - displaying boundaries 512
 - distributing 889, 892
 - drawing objects, adding 523
 - edit styles 566
 - expressions in computed fields 529
 - generating 460
 - Graph presentation style 707
 - graphs, adding 536
 - grid style 498
 - grid, working in 494
 - Group presentation style 602
 - groupbox, adding 523
 - including in resource files 894
 - initial values for columns 587
 - modifying 417
 - multiple column 424
 - naming 461
 - nesting reports 667
 - newspaper columns in 501
 - OLE 772
 - overview 406
 - pictures, adding 524
 - positioning of objects in 538
 - presentation styles 420
 - previewing 479
 - printing 488
 - prompting for criteria 519
 - repository information used 503
 - resizing objects during execution 506
 - result sets, modifying 511
 - retrieval arguments, modifying 510
 - retrieval criteria 519
 - retrieving as needed 541
 - saving 461
 - setting colors 497

- setting timer 497
- sharing with other developers 540
- sorting rows 597
- suppressing repeating values 598, 600
- tab order 504
- text, adding 522
- units of measure 497
- using 407
- validation rules 582
- years, how interpreted 563
- DataWindow painter
 - copying objects 514
 - defining validation rules 588
 - deleting objects 513
 - equalizing size 517
 - equalizing spacing 516
 - importing data 485
 - keyboard shortcuts 476
 - MicroHelp 476
 - modifying data 483
 - moving objects 514
 - nesting reports 660
 - printing data 488
 - resizing bands 478
 - resizing objects 515
 - retrieving data 480
 - saving data 488
 - selecting objects 474
 - sliding objects 517
 - toolbars in 472
 - undoing changes 478
 - using property sheets 473
 - working in the workspace 469
 - working with display formats 557
 - working with edit styles 569
 - zooming objects 478
- DataWindow, OLE
 - activating object 779
 - OLE object 774, 775
 - presentation style 774, 776
 - previewing 779
 - see also* OLE object
- DataWindows
 - caching data 481
 - previewing without retrieving data 481
 - retrieving data 480, 482
 - turning off caching 482
- dates
 - display formats for 562
 - displaying in Library painter 73
 - including in DataWindow objects 531
- dBASE file, using as data source for DataWindow object 431
- DBMS
 - changing 342
 - controlling database access 398
 - CREATE VIEW statement 386
 - defining primary keys 377
 - executing SQL statements 403
 - exporting table or view syntax 389
 - generating SQL statement 389
 - specifying an outer join 388
 - stored procedures 457
 - supported 339
- DDE application, using as data source for DataWindow object 431
- debugging
 - overview 820
 - trace information 901
- Default 3D variable 220
- default command buttons 223
- Default Global Variable Types dialog box 60
- Default to 3D command 220
- Default3D preference variable 220
- defaults
 - application 55
 - control names 201
 - for DataWindow objects 496
 - for DataWindow objects and reports 428
 - global objects 60
 - menu item names 266
 - sequence of control 211
- Defining retrieval arguments 449
- Delete Database command 352
- Delete Library dialog box 76
- DELETE statements
 - building in Database Administration painter 401
 - specifying WHERE clause 546
- DeleteItem function 231
- DeleteRow function 484
- descendent menus
 - building 285

- inherited characteristics 286
- descendent objects
 - allowed changes 251
 - calling ancestor functions 256
 - displaying in Application painter 50
 - inheritance hierarchy 249
 - regenerating 93
 - resetting properties to ancestor's 252
- descendent scripts
 - calling ancestors 256
 - extending ancestors 255
 - overriding ancestors 254
- descendent user objects
 - building 307
 - writing scripts for 310
- descendent windows
 - calling ancestor functions 256
 - characteristics of 193
 - creating 190
 - unique control names 203
- Describe Rows dialog box
 - displaying in preview 485
 - in Data Manipulation painter 395
- design guidelines 161
- destination databases 794
- Destructor event 302
- detail bands
 - in DataWindow painter workspace 471
 - resizable 507
- directories
 - reporting on 103
 - viewing tree structure 70
- Disabled property 224
- disk space 92
- Display Ancestor Script command 254
- display expressions in graphs 714
- Display Format Definition dialog box 556
- display formats
 - about 554
 - adding buttons in DataWindow painter 558
 - adding buttons in Report painter 558
 - applying to columns 557
 - assigning from toolbar 24
 - colors in 559
 - data types 559
 - defining 559
 - defining in Report painter 557
 - deleting 591
 - for dates 562
 - for numbers 561
 - for strings 562
 - for times 564
 - in databases 364
 - in DataWindow objects 552
 - in reports 552
 - maintaining 591
 - masks 559
 - removing 557
 - sections 559
 - setting during execution 560
 - using in graphs 713
 - working with in Database painter 556
 - working with in DataWindow painter 557
- display values
 - of code tables 578
 - using in crosstabs 732
 - using in graphs 696
- display-only fields in DataWindow objects 570
- DISTINCT keyword 442
- DITL resources 898
- divisions, axis 719
- DLL files 883
- DLL files, and external user objects 295
- DLOG resources 898
- DO...LOOP statements 119
- document window 166, 167
- documents, storing in databases 785
- dot notation
 - referring to Menu objects 283
 - referring to properties of windows and controls 186
 - referring to structures 153
- DoubleClick event in ListBox control 231
- drag and drop events 302
- drawing objects
 - adding to DataWindow objects 523
 - list of 196
 - using 236
- drop lines, graph 720
- DROP VIEW statement 389
- dropdown menus
 - about 260

- adding items to 265
- changing order of objects 268
- deleting objects 269
- inserting object into 268
- triggering clicked events 280
- dropdown toolbars 18
- DropDownDataWindow edit style
 - defining 575
 - defining code tables with 579
- DropDownDataWindow Edit Style dialog box 577
- DropDownListBox controls
 - defining accelerator keys for 216
 - edit property 234
 - prefix 201
 - using 233
- DropDownListBox edit style
 - defining 571
 - defining code tables with 579
- DropDownPictureListBox control prefix 201
- DropDownPictureListBox controls
 - adding images 235
 - using 235
- duplicate values, index 381
- dynamic libraries
 - execution search path 886
 - objects copied to 888
 - overview 883
 - specifying in Project painter 884
- dynamic user objects 310

E

- edges, displaying in DataWindow painter 512
- Edit edit style
 - defining 570
 - defining code tables with 578
- Edit Mask edit style
 - defining 573
 - defining code tables with 579
 - spin controls 575
- edit masks
 - keyboard behavior in 229
 - using 227
- Edit menu, in PowerScript painter 28, 113
- edit style, and selection criteria 437

- Edit Style dialog box 568
- edit styles
 - about 566
 - applying to columns 569
 - deleting 591
 - in databases 364
 - in DataWindow objects 552
 - in reports 552
 - in Specify Retrieval Criteria dialog box 520
 - maintaining 591
 - removing 569
 - working with in Database painter 568
 - working with in DataWindow painter 569
 - working with in Report painter 569
- EditMask controls
 - defining as spin controls 229
 - prefix 201
 - using 227
- EditMask property sheet 228
- elevation, in 3D graphs 722
- ellipses, in command buttons 222
- Enabled property 269, 271
 - and PictureBox controls 224
 - for controls 217
- encapsulated functions 136
- ENTER key 223
- Equality Required property 520
- error messages, customizing in validation rules 587
- Error objects
 - default 60
 - defining descendent user object 303
 - using 845
- error rows, correcting in pipelines 811
- errors
 - compile 124
 - execution-time error numbers 846
 - handling 844
 - saving a function with 142
- ESC key 223
- event IDs, naming conventions 323
- events
 - about 5
 - and drawing objects 236
 - application 62
 - calling ancestor scripts for 256
 - changing, in PowerScript painter 109

- declaring your own 322
 - DoubleClickd, in ListBox controls 231
 - for custom and external visual user objects 302
 - for windows and controls 185
 - of Picture controls 235
 - selected 281
 - SystemError 850
- Events dialog box, in User Object painter 316
- EXE files
- copying objects into 887
 - objects excluded from 888
- executable files
- copying objects into 887
 - objects excluded from 888
- executables
- compiling resources into 892
 - creating in Project painter 875, 885
 - overview of creating 874
 - running from PowerBuilder 24
 - specifying dynamic libraries 884
- execution
- handling errors 844
 - placing user objects 310
 - previewing windows 183
 - running in trace mode 901
- execution plan, SQL 402
- expanding objects in Application painter 51
- Explain SQL command 402
- expressions
- in computed fields 529
 - in crosstabs 733
 - in filters 594
 - in graphs 714
 - in OLE client names 789
 - in validation rules 585, 588
 - specifying graph series with 698
 - specifying graph values with 697
- Expressions property page 620
- Extend Ancestor Script command 255
- extended attributes, piping 796
- extended column attributes
- about 364
 - how stored 906
 - picture columns 358
 - used for text 503
- External data source
- importing data values 456
 - modifying result sets 511
 - updating data 543
- external data, importing 395
- external files
- importing data from 485
 - saving data in 488
 - saving functions in 139
 - saving table data in 396
 - using as data source for DataWindow object 431
- external functions
- including in user-defined functions 139
 - structures as arguments 155
- external visual user objects 295

F

- File Close command 127
- file editor 27
- File Export 122, 139
- File Import 122, 140
- files
- exporting 139
 - exporting from script 122
 - for PictureBox controls 223
 - importing 140
 - importing into script 122
 - rich text 757
 - saving functions in 139
 - saving scripts in 122
- fill state, checkbox 226
- filters
- in Data Manipulation painter 394
 - removing 595
- focus
- for default command button 223
 - moving from column to column 504
 - of controls in windows 211
- Font dialog box
- changing text properties in 504
 - specifying fonts for tables 370
- font.escapement property, on the Expressions property page 641
- font.height property, on the Expressions property page 644

font.italic property, on the Expressions property page 645
font.strikethrough property, on the Expressions property page 646
font.underline property, on the Expressions property page 647
font.weight property, on the Expressions property page 647
fonts
 choosing, for controls 205
 default 55
 in DataWindow object, changing 504
 rich text formatting 759
 specifying for tables 370
footer bands, in DataWindow painter workspace 472
Foreground Color dropdown toolbar 472
Foreign Key Definition dialog box 378
foreign keys
 about 374
 defining 378
 displaying in Database painter 375
 joining tables 387, 447
 opening related tables 376
format property, on the Expressions property page 648
Freeform style
 default wrap height 429
 detail band in 471
 header band in 470
 of DataWindow objects and reports 421
FREF resource 899
FUN file 139
Function Declaration dialog box 143
Function For Toolbar dialog box 531
Function painter
 opening 132
 workspace 138
functions
 about 5
 browsing 118
 built-in 281
 calling 256
 communicating between user objects and windows 315
 crosstab 743
 for drawing objects 236
 for windows and controls 186
 in descendent menus 286

 passing and returning structures 155
 pasting into scripts 118
 pasting names of 121
 user-defined 130

G

General keyword, in number display formats 561
GetFormat function 560
GetText function, using in validation rules 588
GetValue function 571
global functions
 access level 136
 opening Function painter 132
 saving 141
 user-defined 130
global objects
 specifying defaults 60, 312
 specifying user objects for 312
global standard class user objects 312
global structures
 about 148
 opening Structure painter 149
 referring to 153
 saving 150
global variables
 and Menu object scripts 282
 and windows 187
 pasting into scripts 118
graph controls, prefix 201
Graph Object property sheet 695, 708
graphics formats 895
graphics, adding to DataWindow objects 524
graphs
 adding to DataWindow objects 536
 adding to reports 411
 autosizing text 712
 changing position of 692
 data types of axes 717
 default positioning in DataWindow objects 538
 defining properties 708
 examples 699
 expressions in 714
 in DataWindow objects 689
 legends in 710

- major and minor divisions 719
 - multiple series 698
 - names of 709
 - overlays in 705
 - overview 682
 - parts of 682
 - placing in windows 723
 - property sheets 691
 - rotating text 713
 - scaling axes 718
 - selecting data 694
 - single series 697
 - sorting series and categories 710
 - specifying borders 721
 - specifying categories 696
 - specifying overlap and spacing of bars/columns 716
 - specifying pointers 721
 - specifying properties of axes 716
 - specifying rows 695
 - specifying series 697
 - specifying the type 710
 - specifying values 697
 - text properties in 711
 - titles in 709
 - types of 684
 - using display formats 713
 - using Graph presentation style 707
 - using in applications 688
 - using in windows 723
 - GraphType attribute 710
 - grid
 - aligning controls 207
 - aligning DataWindow objects 513
 - grid lines, graph 720
 - Grid style
 - basic properties 498
 - detail band in 471
 - displaying grid lines 498
 - header band in 470
 - of DataWindow objects and reports 422
 - reordering columns 494
 - resizing columns 494
 - using split horizontal scrolling 495
 - working in 494
 - GROUP BY criteria 453
 - Group presentation style
 - properties of 605
 - using 602
 - GroupBox controls
 - and radio buttons 225
 - default tab order 211
 - prefix 201
 - groupbox, adding to DataWindow objects 523
 - grouping
 - in SQL Select 453
 - restricting 454
 - groups in DataWindow objects
 - of rows 600
 - sorting 614
 - groups, graphing 695
- ## H
- HAVING criteria 454
 - header bands, in DataWindow painter workspace 470
 - heading extended attribute 364
 - headings
 - default properties 55
 - in DataWindow objects 470
 - specifying fonts in tables 370
 - height property, on the Expressions property page 649
 - Help
 - context-sensitive 115
 - using 12
 - Hide function 236
 - hierarchies
 - browsing class 81
 - displaying in Application painter 50
 - inheritance 249
 - HScrollBar controls
 - prefix 201
 - using 237
 - HTML Table format, saving data in 489
 - hyphens (-) 274
- ## I
- ICO files, naming in resource files 893

- icons
 - application 59
 - Application object 47
 - for windows, choosing 178
 - in Library painter 87
 - Macintosh resources 898
 - identity columns, in DataWindow objects 545
 - Idle event 62
 - IF...THEN statements 119
 - image files 891, 895
 - importing data 395, 485
 - IN operator, in Quick Select 438
 - Include dialog box 72
 - indexes
 - creating 381
 - dropping from tables 379, 380, 382, 383
 - Inherit From User Object dialog box 307
 - inheritance
 - browsing class hierarchies 81
 - building menus with 285
 - building user objects with 307
 - building windows with 190
 - hierarchy 193, 249
 - using unique names 203
 - inherited controls
 - deleting 193
 - syntax of 193
 - inherited objects, displaying 50
 - inherited properties 251
 - inherited scripts 253
 - initial values, for columns 587
 - initialization files
 - about 33
 - changing path 35
 - editing 27
 - how PowerBuilder finds them 33
 - saving custom colors 219
 - setting Default 3D variable 220
 - .WindU 36
 - input fields
 - about 759
 - columns 759
 - computed fields 760
 - data 753, 760
 - data format 760
 - editing data 760
 - flashing 760
 - selected 760
 - validation rules 760
 - Input Validation dialog box 584
 - INSERT statements, building in Database Administration painter 401
 - InsertItem function 231
 - InsertRow function 484
 - instance variables
 - accessing structures with 154
 - and Menu object scripts 282
 - in ancestor objects 251
 - in window scripts 187
 - pasting into scripts 118
 - instances, menu 292
 - invisible property 193
 - items
 - adding to menus 264
 - in dropdown listboxes 234
 - in listboxes 231
- J**
- Join dialog box 388
 - joins, in Select painter 447
- K**
- key and modified columns, updating rows 546
 - key and updatable columns, updating rows 546
 - key columns for OLE columns 786
 - key columns, updating rows 546
 - key modification, updating rows 549
 - keyboard
 - for moving and resizing controls 207
 - shortcuts in Database painter 347
 - shortcuts in DataWindow painter 476
 - using paste listboxes with 117
 - using with menus 272
 - keys, database
 - arrows specifying key relationship 433
 - displaying in Database painter 375
 - dropping from tables 379
 - specifying in DataWindow objects 545

updating values in DataWindow objects 549
using primary and foreign 374
keywords, display format 559

L

label extended attribute 364

Label style

detail band in 471
of DataWindow objects and reports 422
removing blank lines 517

labels

default properties 55
indicating accelerator keys 216
mailing 422, 517
specifying fonts in tables 370

LargeIcon view 246

layer attribute of graphs 692

Layout dropdown toolbar 472

left alignment of controls 208

legends

in graphs 684
specifying text properties 711
using 710

levels, menu 265

libraries

about 6
creating 75
deleting 76
deleting from search path 58
dynamic 883
maintaining 75
migrating 95
optimal size of 66
optimizing 92
organization of 67
rebuilding 94
regenerating 93
reporting on 103
specifying search path 57
storage of objects in 66
storing Application objects 45
updating comments 74
viewing tree structure 70

Libraries property tab 58

library entries

checking in 88
checking out 85
check-out status 83, 89
copying, moving, and deleting 77
exporting to text files 96
regenerating 93
reporting on 102
searching 78
selecting 73
updating comments 74
viewing checked out 87

library list 57, 68

Library painter

changing print settings 184
Class browser 250
displaying window comments 181
expanding and collapsing trees 70
finding called functions 144
icons in 87
jumping to painters 80
modifying object comments 73
opening 70
popup menu 71
restricting displayed objects 71

LIKE operator, in Quick Select 437

Line controls 201

line drawing objects 523

line graphs

about 684
making three-dimensional 687

line styles, graph 721

lines, menu 274

List Objects dialog box 890

List view 246

ListBox controls

indicating accelerator keys 227
prefix 201
setting tab stops 231
using 230

ListView control prefix 201

ListView controls

LargeIcon view 246
list view 246
properties 246
report view 246

- SmallIcon view 246
 - using 244
 - view style 246
 - local SQL Anywhere databases 351
 - local variables, and Menu object scripts 282
 - locked menu names 267
 - log files
 - about 348
 - clearing 350
 - saving 349
 - viewing 349
 - logging
 - exporting table syntax 389
 - starting 349
 - stopping 350
 - logical operators 437
 - LookupDisplay function 696
- ## M
- Macintosh
 - accelerator keys 272
 - desktop 167
 - executing AppleScript 29
 - popup menus 20
 - resources 878, 895
 - shortcut keys 273
 - Toolbox functions 896
 - mailing labels 422
 - mailing reports (PSR files) 493
 - main window 166
 - main windows
 - about 162
 - specifying window type 173
 - major divisions, in graphs 719
 - masks
 - for display formats 559
 - using 227
 - Match button with validation rules 586
 - match patterns, validation rules 586
 - Matching Library Entries dialog box 79
 - MDI
 - custom frames 168
 - frames 162
 - menus 167
 - MicroHelp 168
 - MDI applications
 - about 166
 - creating shell 49
 - menu bars
 - about 260
 - adding items to 264
 - adding to windows 291
 - changing order of objects 268
 - deleting objects 269
 - in Window painter 170
 - inserting objects into 268
 - Menu objects
 - about 260
 - adding to cascading menus 265
 - adding to menu bars 264
 - changing order of 268
 - Click event 280
 - deleting 269
 - duplicate names 267
 - events 280
 - inserting in descendent menus 287
 - properties 269, 271
 - referring to, in scripts 283
 - renaming 267
 - Selected event 281
 - Shift Over 287
 - using variables 282
 - writing scripts for 280
 - Menu painter
 - opening 262
 - saving menus 276
 - workspace 263
 - menu scripts, calling ancestor scripts 256
 - menus
 - associating with windows 174
 - building 260
 - calling ancestor functions 256
 - cascading 265
 - changing during execution 292
 - creating separation lines 274
 - deleting objects 269
 - dropdown 265
 - in descendent objects 251
 - inserting objects into 268

- MDI 167
- menu bars 264
- MicroHelp 168
- overview 260
- previewing 278
- printing definitions 279
- saving 276
- shortcut keys 273
- using inheritance with 285
- window 260
- message boxes 166
- Message objects
 - default 60
 - defining descendent user object 303
- MessageBox function 167
- messages, error 846
- metafiles, specifying columns as 358
- MicroHelp
 - about 168
 - displaying with Selected event 280
 - in DataWindow painter 476
- MicroHelp text 270
- military time 564
- minor divisions, in graphs 719
- modal dialog 166
- modal windows 165
- modeless dialog 167
- Modify Result Set Description dialog box 511
- mouse pointers, in DataWindow objects 499
- Move function 236
- movable modal dialog 166
- MultiLineEdit controls
 - defining accelerator keys for 216
 - prefix 201
 - setting tab stops 231
 - using 227
- multiple columns in DataWindow objects 501
- multiple columns, in DataWindow objects and reports 424
- multiple-document interface 162
- multiple-series graphs 698
- N**
- name tags 423
- names
 - of Application objects 45
 - of columns in DataWindow painter workspace 471
 - of controls 201
 - of DataWindow controls and objects 506
 - of DataWindow objects and reports 461
 - of graphs 709
 - of inherited controls 193
 - of Menu objects 266, 291
 - of Menu objects in inherited menus 286
 - of menus 277
 - of queries 463
 - of structures 150
 - of user objects 306, 310
 - of user-defined functions 131, 134
 - of windows 182
 - pasting function 121
 - pasting into scripts 115
- naming conventions
 - for controls 202
 - for DataWindow objects and reports 461
 - for event IDs 323
 - for queries 463
 - for user objects 306
 - for user-defined functions 134
 - for windows 182
- negative numbers, in TextSize property 205
- nested reports
 - about 660
 - adding another report 674
 - adding to report (DataWindow) 667
 - adding to reports 414
 - autosize height 678
 - changing 673
 - changing content of 673
 - displayed in workspace 670
 - how retrieval works 663
 - limitations 663
 - slide options 678
 - specifying criteria 676
 - using retrieval arguments 669, 675
 - width, adjusting 673
- New Function dialog box
 - for global functions 133
 - Returns listbox 135

- New Name dialog box 740
 - New Structure dialog box 149
 - New User Object dialog box 298
 - newline characters in text 504
 - newspaper columns 501
 - Next Level command 265
 - NULL values
 - allowing in code tables 578
 - allowing in tables 362
 - altering table definition 368
 - specifying display formats for 561
 - NULL values and blob columns 786
 - numbers, display formats for 561
 - N-Up style
 - computed fields in 529
 - detail band in 471
 - header band in 470
 - of DataWindow objects and reports 424
- O**
- object boundaries 512
 - object-level functions
 - access levels 136
 - calling 146
 - opening Function painter 133
 - saving 141
 - user-defined 130
 - object-level structures
 - definition 148
 - opening Structure painter 149
 - referring to 154
 - saving 151
 - objects
 - adding to DataWindow objects 521
 - aligning in DataWindow painter 516
 - application 44
 - compiled form 66
 - copying in DataWindow painter 514
 - copying into EXE files 887
 - copying into executable files 887
 - deleting from DataWindow painter 513
 - distributing 889
 - equalizing size 517
 - equalizing spacing 516
 - exporting syntax 96
 - importing syntax 96
 - inheritance hierarchy 249
 - modifying comments 73
 - moving in DataWindow painter 514
 - pasting into scripts 115
 - pasting with Browser 118
 - referring to, in Menu object scripts 282
 - regenerating 93
 - resizing in DataWindow painter 515
 - selecting, in DataWindow painter 474
 - Objects dropdown toolbar 472
 - off state, checkbox 226
 - OLE
 - columns in DataWindows 786
 - data transfer from DataWindow 774
 - DataWindow objects 772
 - DataWindow presentation style 774
 - previewing columns 790
 - report objects 772
 - OLE 2.0 controls
 - placing in windows 197
 - prefix 201
 - OLE Database Blob command 787
 - OLE object
 - activating object 785
 - display of 785
 - embedding 784
 - icon for 785
 - linking 784
 - updating link 785
 - on state, checkbox 226
 - Open event
 - and Application object 44
 - and popup windows 163
 - opening
 - Application painter 45, 48
 - Browser 118
 - Data Manipulation painter 390
 - Database Administration painter 398
 - Database painter 342
 - Library painter 70
 - Menu painter 262
 - PowerScript painter 108
 - Query painter 462
 - Select painter 441

- Structure painter 149
 - View painter 384
 - Window painter 169
 - OpenUserObject function 310
 - OpenWithParm function 186
 - operators, in Quick Select criteria 437
 - optimizing libraries 92
 - option key 273
 - options
 - choosing for DataWindow objects and reports 428
 - mutually exclusive 224
 - OR operator, in Quick Select 438
 - order
 - of arguments in functions 137
 - of menu objects, changing 268
 - tab, in windows 211
 - ORDER BY clause
 - in SELECT statements 452
 - specifying in Quick Select 436
 - Other event 302
 - outer join, specifying 388
 - Oval controls 201
 - oval drawing objects 523
 - overlap, of columns in graphs 716
 - overlays, in graphs 705
 - Override Ancestor Script command 254
 - owner resource 898
- P**
- page numbers, including in DataWindow objects 531
 - page, graphing data on 695
 - PainterBar
 - about 18
 - adding custom buttons to 23
 - controlling display of 19
 - in PowerScript painter 28, 113
 - in Window painter 170
 - painters
 - about 4
 - displaying objects referenced in application 51
 - jumping to 80
 - opening 11
 - painting SQL statements 399
 - palettes 219
 - paragraph alignment 759
 - Parent reserved word 187
 - parents
 - in Class browser 250
 - of windows 163
 - ParentWindow reserved word 283
 - passing
 - arguments in functions 137
 - parameters between windows 186
 - return values between windows 186
 - structures as arguments in functions 155
 - passwords
 - defining text boxes for 227
 - displaying as asterisks 570
 - fields 570
 - Paste Argument listbox 138
 - Paste Function dialog box 121
 - Paste Global listbox
 - in Function painter 138
 - in PowerScript painter 117
 - Paste Instance listbox
 - in Function painter 138
 - in PowerScript painter 117
 - Paste Object listbox
 - in Function painter 138
 - in PowerScript painter 117
 - Paste SQL button 120
 - Paste Statement button 119
 - Paste Statement dialog box 119
 - pasting
 - from inherited scripts 254
 - into scripts, table 115
 - SQL statements, in Database Administration painter 400
 - statements 119
 - structures 156
 - text, in scripts 28, 113
 - user-defined functions 146
 - with paste listboxes 116
 - paths, library 57
 - PB.INI files
 - how PowerBuilder finds them 33
 - saving custom colors 219
 - setting Default 3D variable 220
 - PBCatCol table 354, 909

- PBCatEdt system table 354
- PBCatEdt table 912
- PBCatFmt system table 354
- PBCatFmt table 910
- PBCatTbl system table 354
- PBCatTbl table 907
- PBCatVld system table 354
- PBCatVld table 911
- PBD files 883
- pbdebug flag 901
- PBLAB60.INI 424
- pen.color property, on the Expressions property page
 - about 649
 - specifying colors 657
- pen.style property, on the Expressions property page 650
- pen.width property, on the Expressions property page 651
- percent display format 558
- performance
 - and fragmented libraries 92
 - and library size 66
- periodic data, in DataWindow objects and reports 424
- perspective, in 3D graphs 722
- phone lists, creating 501
- PICT image format 895
- Picture controls
 - placing in windows 197
 - prefix 201
 - using 235
- picture height 233
- picture mask 233
- picture width 233
- PictureButton controls
 - placing in windows 197
 - prefix 201
 - using 223
- PictureListBox control prefix 201
- PictureListBox controls
 - adding images 232
 - using 232
- pictures
 - adding to DataWindow objects 524
 - specifying column as 358
- pie graphs
 - about 685
 - making three-dimensional 687
- pipeline objects, defining descendent user object 303
- pipelines
 - about 794
 - creating 798
 - data types, supported 795
 - destination database 794
 - destination, changing 809
 - editing source data 800
 - error messages 812
 - errors, correcting 812
 - examples 794, 815
 - executing 800
 - execution, stopping 805
 - extended attributes 796
 - modifying 801
 - opening 814
 - pipeline operations 803
 - retrieval arguments 799
 - reusing 814
 - rows, committing 806
 - saving 813
 - source database 794
- pixels
 - as DataWindow object unit of measure 497
 - saving text size in 205
- placeholders, in validation rules 586
- point of view, in 3D graphs 722
- pointer property, on the Expressions property page 652
- pointers
 - in DataWindow objects 499
 - in graphs 721
 - window, choosing 177
- points
 - saving text size in 205
 - specifying size for tables 370
- polymorphism 131
- PopupMenu function 292
- popup menus
 - Application painter 50
 - assigning colors to controls 218
 - controlling toolbars with 19
 - creating an instance of the menu 292
 - displaying 292
 - in Library painter 71
 - on the Macintosh 20
 - opening PowerScript painter with 108

- use of in applications 260
- using 14
- popup windows
 - about 163
 - modal 165
 - naming parents of 163
 - on the Macintosh 167
 - specifying window type 173
- position
 - changing control's 207
 - changing graph's 692
 - equalizing 209
 - of windows 175
- PowerBar
 - about 18
 - adding custom buttons to 23
 - controlling display of 19
 - displaying available buttons 21
 - using 7
- PowerBuilder initialization file, format 33
- PowerBuilder units 176
- PowerScript
 - about 5
 - expressions in computed fields 529
 - statements 139
- PowerScript painter
 - context-sensitive help 115
 - elements of 108
 - opening 108
 - quitting 127
- PowerTips
 - assigning text in custom buttons 24
 - using 8
- predefined objects in applications 60
- preference variables
 - control name prefixes 201
 - Default3D 220
 - for colors 219
- preferences
 - changing print settings 184
 - setting default grid size 208
- prefixes
 - in window names 182
 - of controls, default 201
 - of user object names 306, 310
- presentation styles
 - of DataWindow objects and reports 420
 - of reports 420
 - using Crosstab 726
 - using Graph 707
 - using Group 602
- preview
 - for crosstabs 737
 - for menus 278
 - for windows 183
 - importing data 485
 - in DataWindow painter 479
 - modifying data 483
 - retrieving rows 480
 - sorting and filtering data 484
- previewing
 - displaying rulers 486
 - OLE DataWindow objects 779
- Primary Key Definition dialog box 376
- primary keys
 - about 374
 - defining 376
 - displaying in Database painter 375
 - identifying updatable rows 545
 - joining tables 387, 447
 - opening related tables 376
- Print Options dialog box 102
- Print Preview command 486
- print specifications, reports 499
- printing
 - data, in preview 486
 - DataWindow objects 488
 - in Data Manipulation painter 396
 - menu definitions 279
 - scripts 115
 - window definitions 184
- Prior Level command 265
- private access level 136
- private libraries
 - checking out working copies 83
 - organizing 67
- procedures, defining 135
- profiles
 - creating from trace file 851
 - in pipelines 798
- profiling an application 851
- Project painter

- building a project 885
- defining a project 876
- overview 875
- specifying dynamic libraries 884
- projects
 - building 885
 - defining 876
 - objects in 889
 - reports of objects 889
- Prompt For Criteria dialog box 519
- properties
 - about 618
 - application 55
 - browsing 118
 - control-level 199
 - defining for graphs 708
 - example, gray bar 624
 - example, highlighting rows 629
 - example, rotating objects 627
 - example, size and location 632
 - in descendent objects 251, 308
 - in scripts 118, 282
 - modifying objects 622
 - of dropdown listboxes 234
 - of listboxes 232
 - of Menu objects 269, 271, 283
 - of StaticText controls 226
 - of windows and controls 186
 - searching for 78
 - specifying colors 657
 - text, of controls 205
 - using expressions 620
 - window-level 171
- property conditional expressions 618, 748
- property sheets
 - about 16
 - arrangement of tabs 16
 - buttons 16
 - defining control properties 199
 - displaying 17
 - for graphs 691
 - for graphs in windows 724
 - in DataWindow painter 473
 - RichText object 767
 - tab pages 199
- property values on the Expressions property page

- about 635
- background.color 636
- border 637
- brush.color 638
- brush.hatch 638
- color 641
- font.escapement 641
- font.height 644, 649
- font.italic 645
- font.strikethrough 646
- font.underline 647
- font.weight 647
- format 648
- pen.color 649
- pen.style 650
- pen.width 651
- pointer 652
- specifying colors 657
- supplying 635
- visible 653
- width 654
- x 654
- x1, x2 654
- y 655
- y1, y2 655
- protected access level 136
- PSR files
 - creating 489
 - mailing 493
 - opening 492
 - overview 491
- public access level 136
- public libraries
 - checking out objects 83
 - organizing 67

Q

- queries
 - defining 462
 - modifying 464
 - naming 463
 - previewing 462
 - running from toolbar 24
 - saving 463

- Query data source 455
 - Query painter, opening 462
 - question marks (?) 72
 - quick application feature 49
 - Quick Select
 - defining data source as 431
 - up and down arrows 433
- R**
- RadioButton controls
 - default tab order 211
 - defining accelerator keys for 214
 - prefix 201
 - using 224
 - using group boxes 225
 - RadioButton edit style, defining 573
 - ranges, cross-tabulating 744
 - ranges, spin value 230
 - RButtonDown event 273
 - RbuttonDown event 302
 - Rebuild Columns At Runtime checkbox 747
 - rebuilding libraries
 - full 94
 - partial 94
 - Rectangle controls 201
 - rectangle drawing objects 523
 - referential integrity, in databases 374
 - RegEdit utility, adding OLE server applications 789
 - regenerating objects 93
 - Report painter
 - defining display formats 557
 - defining validation rules 588
 - nesting reports 660
 - working with edit styles 569
 - Report view 246
 - report, OLE
 - OLE object 774, 775
 - presentation style 774
 - see also* OLE object
 - reports
 - about the repository 460
 - business cards 423
 - Composite presentation style 660
 - Composite style 414
 - creating new 417
 - Crosstab style 413
 - data sources 430
 - defaults 428
 - display formats 554
 - edit styles 566
 - Freeform style 412
 - generating 460
 - graphs, adding 411
 - Grid style 411
 - Group presentation style 602
 - Group style 410
 - initial values for columns 587
 - Label style 412
 - mailing 493
 - modifying 417
 - multiple column 424
 - name tags 423
 - naming 461
 - nested reports 414
 - OLE 772
 - on library contents 102
 - overview 406
 - presentation styles 420
 - print specifications 499
 - PSR files 489
 - retrieving data 482
 - running from toolbar 24
 - saving 461
 - tab order 504
 - Tabular style 409, 410
 - using 407
 - years, how interpreted 563
 - repository
 - about 354, 460, 905
 - deleting orphan table information 359
 - information used in DataWindow objects 503
 - information used in DataWindow objects and reports 460
 - pipng 797
 - storing display formats 556
 - storing edit styles 568
 - storing extended attributes 365
 - storing validation rules 584
 - Resize function 236
 - resizing objects during execution 506

- resource files
 - creating 893
 - Macintosh 878
 - overview 892
- resources
 - distributing 889, 891
 - execution search path 900
 - finding 891
 - Macintosh 895
 - naming in resource files 893
 - PowerBuilder 895
 - specifying for dynamic libraries 884
- response windows 166
 - about 165
 - specifying window type 173
- result sets, modifying 511
- Retain Data to Design option 482
- retrieval arguments
 - defining 448
 - in nested reports 669
 - modifying in DataWindow objects 510
 - referencing 450
 - specifying in pipelines 799
 - specifying in WHERE clause 451
- retrieval criteria
 - in nested reports 676
 - in Quick Select grid 437
 - prompting for in DataWindow objects 519
- Retrieve button 391
- Retrieve command 480
- Retrieve on Preview option 481
- Retrieve Only As Needed 541
- Retrieve Rows to Disk command 542
- retrieving data as needed 483
- Return button 127
- RETURN statements 139
- return type
 - changing for function 143
 - defining 134
 - none 135
 - structure 155
- return values, passing between windows 186
- Returns listbox 135
- rich text 751
 - about 752
 - file for DataWindow 757
 - header and footer 757
 - pictures 762
 - tables 752
 - toolbars 754
 - unsupported formatting 752
- RichText object 767
- RichText presentation style
 - about 753
 - columns 759
 - creating 754
 - data for input fields 760
 - DataWindow settings 758
 - editing keys 768
 - header and footer 754, 757, 763
 - objects 758
 - page numbers 757
 - paragraph settings 759
 - picture objects 762
 - preview 756, 763
 - print preview 764
 - protected from changes 754
 - Rich Text Object settings 758
 - selected text settings 759
 - setting up 756
 - settings 754
 - today's date 757
- RichTextEdit controls
 - editing keys 768
 - popup menu 767
 - prefix 201
 - properties 765
 - using 765
- right alignment, of controls 208
- right mouse button 273
- RLE files
 - adding to DataWindow objects 524
 - and Picture controls 235
 - and PictureBox controls 223
 - naming in resource files 893
- rotation
 - in 3D graphs 722
 - of text in graphs 713
- Round Maximum To, in graphs 718
- RoundRectangle controls 201
- RoundRectangle drawing objects 523
- rows

- allowing users to select 519
 - displaying information about 395, 485
 - errors in pipelines 811
 - filtering 394, 594
 - graphing 695
 - grouping 453, 600
 - modifying in Data Manipulation painter 391
 - modifying in preview 483
 - removing filters 595
 - retrieving 391
 - retrieving as needed 541
 - saving in external files 396
 - sorting 392, 452, 597
 - suppressing repeating values 598
 - RTF 751
 - ruler, in DataWindow painter 513
 - rulers, displaying in print preview 486
 - Run Window button 189
 - Run Window dialog box 189
 - runtime libraries, creating 100
- S**
- Save As command
 - changing function name 143
 - changing structure name 152
 - Save As dialog box 489
 - Save Rows As dialog box 396
 - Save Structure In dialog box 151
 - saving
 - blob data in databases 791
 - data in DataWindow objects 539
 - data in external files 488
 - data in HTML Table format 489
 - DataWindow objects and reports 461
 - functions in files 139
 - menus 276
 - pipelines 795, 813
 - queries 463
 - scripts to files 122
 - structures 150
 - user-defined functions 140
 - windows 181
 - scatter graphs 685
 - scope, variable 187
 - Script button, opening painter with 108
 - Script icon
 - in Select Event listbox 109
 - of inherited scripts 253
 - Script painter *see* PowerScript painter
 - scripts
 - about 5
 - accessing object-level structures 154
 - changing labels in 226
 - changing text size 205
 - compiling 123
 - copying files into 122
 - defined 5
 - displaying referenced objects 51
 - extending 255
 - for custom visual user objects 310
 - for descendent user objects 308
 - for Menu objects 280, 282
 - for user events 328
 - for user objects 310
 - in windows 185
 - inherited 253
 - inserting comments 28, 113
 - overriding ancestor 254
 - pasting with Browser 118
 - pasting with listboxes 116
 - printing 115
 - referring to Menu objects 266
 - referring to structures 153
 - searching for strings in 78
 - viewing ancestor 254
 - Scroll Columns Per Page 179
 - Scroll Lines Per Page 180
 - scrollbars
 - for textboxes 227
 - freestanding 237
 - in listboxes 232
 - on windows 179
 - Search Library Entries dialog box
 - in Library painter 144
 - using 78
 - search path
 - for resource files 891
 - of application libraries 68
 - of resources in resource files 894
 - specifying libraries 57

- search strings, library entry 78
- searching for text 28, 113
- Select All command 198
- Select Application dialog box
 - New button 45
 - opening 48
- Select Event listbox 109
- Select Executable File dialog box 876
- Select Function dialog box
 - for global functions 133
 - for object-level functions 133
- Select Menu dialog box 262
- Select painter
 - adding tables 444
 - colors in 444
 - defining retrieval arguments 448
 - joining tables 447
 - opening 441
 - saving work as query 442
 - selecting tables 443
 - specifying selection, sorting, and grouping criteria 450
 - specifying what is displayed 443
 - SQL toolbox 443
- Select Pointer property tab page 177
- SELECT statements
 - building in Database Administration painter 401
 - displaying 446
 - editing syntactically 446
 - for view, displaying 386
 - limiting data retrieved 594
 - predefined 462
 - saved as queries 462
 - sorting rows 597
- Select Structure dialog box
 - global structures 149
 - object-level structures 149
- Select Tables dialog box 342, 384
- Select Tables dialog box in Database painter 353
- Select User Object dialog box, Inherit button 307
- Select Window dialog box
 - Inherit button 191
 - New button 169
- selected events 281
- selecting
 - application 48
 - code to save in file 139
 - controls in windows 198
 - multiple listbox items 232
 - objects in DataWindow painter 474
- selection criteria
 - allowing users to specify 441, 519
 - specifying in Quick Select 436
 - specifying in SQL Select 451
- Series axis, graph 683
- series, graph
 - as overlays 705
 - basics 683
 - specifying 697
- SetFormat function 560
- SetTabOrder function 505
- SetValue function 571
- shared variables, in window scripts 187
- sheets 167
- shift key 273
- Shift Oversetting for Menu objects 287
- ShiftToRight property 269, 271
- shortcut keys
 - assigning to menu items 272, 273
 - in DataWindow painter 476
 - menus 273
 - triggering clicked events 280
- Show Edges option 512
- Show function 236
- SignalError function 850
- signature for application 879
- SingleLineEdit controls
 - defining accelerator keys for 216
 - prefix 201
 - using 227
 - using edit masks 227
- single-series graphs 697
- size
 - defaults 55
 - displaying in Library painter 73
 - equalizing in DataWindow painter 517
 - of bands in DataWindow painter 478
 - of controls 207, 209
 - of DataWindow objects 515
 - of dropdown listboxes 234
 - of libraries 66
 - of windows 175

- Size Controls command 209
- Slide dropdown toolbar 472
- Slide option 517
- slide options, used in nested reports 678
- SmallIcon view 246
- snaking columns in DataWindow objects 501
- snap to grid 207
- sort criteria, specifying in Quick Select 436
- sort order, listbox 232
- sorting
 - groups 614
 - in graphs 710
 - in SQL Select 452
 - rows 597
- source
 - exporting to text files 96
 - object 66
- source control 83
- source databases 794
- source libraries, creating for dynamic libraries 883
- Space Controls command 209
- space, library 92
- spacing
 - equalizing in DataWindow painter 516
 - of columns in graphs 716
 - of controls 209
- Specify Retrieval Criteria dialog box, displaying to users 519
- Specify Sort Columns dialog box 597
- Specify Update Properties dialog box 544
- spin controls
 - defining edit masks as 575
 - using 229
- spreadsheets, storing in databases 785
- SQL Anywhere databases, creating and deleting 351
- SQL Select
 - adding tables 444
 - defining retrieval arguments 448
 - joining tables with 447
 - selecting columns 444
 - selecting tables 443
 - specifying selection, sorting, and grouping criteria 450
 - specifying what is displayed 443
 - using as data source 441
- SQL Select data source
 - colors in 444
 - SQL toolbox 443
- SQL Statement Type dialog box 120, 400
- SQL statements
 - and user-defined functions 139
 - building and executing 399
 - displaying 446
 - executing 399, 403
 - execution plan 402
 - explaining 402
 - exporting to another DBMS 389
 - for views, displaying 386
 - generating through Quick Select 431
 - generating through SQL Select 441
 - importing from text files 402
 - logging 348
 - painting 399
 - pasting 120
 - typing 402
- SQL toolbox 443
- SQLCache variable 675
- stacked graphs 687
- standard class user objects
 - about 296
 - building 303
 - writing scripts for 303
- standard visual user objects
 - about 295
 - building 299
- Start on New Page command 679
- statements 119
- states
 - of checkboxes 226
 - of radio buttons 224
- StaticText controls
 - defining accelerator keys 216
 - prefix 201
 - using 226
- status
 - checked out 87, 89
 - of library entries 83
- stock pointers, choosing for windows 177
- Stored Procedure data source 457
- stored procedures
 - modifying result sets in DataWindow objects 511
 - updating data in DataWindow objects 543

- updating data in forms 543
- using 457
- STR# resource 899
- strings
 - concatenating 529
 - display formats for 562
- Structure dialog box 152
- Structure painter
 - button 149
 - opening 149
- structures
 - copying 152, 155
 - defining 149
 - embedding 150
 - in descendent menus 286
 - modifying 152
 - passing arguments as in functions 155
 - pasting into scripts 118
 - types of 148
 - using 153
- style
 - default text 55
 - of DataWindow objects 496
 - of windows 171
- Style dialog box, DataWindow painter 496
- StyleBar
 - about 18
 - controlling display of 19
 - in Report painter 473
 - in Window painter 170
 - positioning, in Window painter 170
- styles, presentation 420
- suffix, control name 202
- sum
 - computing 530
 - in graphs 697
- summary bands, in DataWindow painter workspace 472
- summary statistics
 - computing 530
 - in crosstabs 741
- Super reserved word 256
- syntax
 - exporting to another DBMS 389
 - for calling ancestor scripts 256
 - of view SELECT statement 386
- system memory requirements 879

- system options tab 35
- system tables
 - DBMS 354
 - PowerBuilder 354, 365, 905
- SystemError event 44, 62, 850
- SystemError scripts 844

T

- Tab control prefix 201
- Tab controls
 - adding pages 237
 - properties 239
 - user objects in 237
 - using 237
- tab order
 - in DataWindow objects 504
 - in reports 504
 - in windows 211
 - setting 212
- tab stops, setting 231
- tab values 212
- Table painter
 - about 340
 - applying validation rules 585
- tables
 - altering definition of 367
 - applying display formats to columns 557
 - applying edit styles to columns 569
 - applying validation rules to columns 584
 - controlling updates to 543
 - creating 360
 - creating indexes 381
 - dropping 358
 - dropping indexes 379, 380, 382, 383
 - exporting syntax to another DBMS 389
 - extended attributes, specifying 364
 - joining in Select painter 447
 - opening in Database painter 353
 - opening, related to foreign keys 376
 - opening, related to primary keys 376
 - presenting in Freeform style 421
 - presenting in Grid style 422
 - presenting in Label style 422
 - presenting in N-Up style 424

- presenting in Tabular style 421
 - printing data 396
 - refreshing list 353
 - removing from Database painter workspace 358
 - resizing in Database painter workspace 345
 - rich text 752
 - saving data in external files 396
 - selecting for SQL Select 442
 - selecting, in foreign key definition 378
 - specifying fonts 370
 - specifying updatable 544
 - working with data 390
- tab-separated files, using as data source for
 DataWindow object 431
- Tabular style
- detail band in 471
 - header band in 470
 - of DataWindow objects and reports 421
- target data for OLE 782
- testing
- menus 278
 - windows 189
- testing, windows 183
- text
- changing properties, in controls 205
 - cutting, copying, and pasting 28, 113, 369
 - default properties 55
 - editing 27
 - in DataWindow objects 503, 522
 - inserting newline characters 504
 - of menu items 267
 - on toolbar buttons 19
 - rotating in graphs 713
 - searching and replacing 28, 113
 - size 205
- text files
- exporting objects to 96
 - importing SQL statements from 402
- text patterns, matching in validation rules 586
- text properties
- in DataWindow objects 504
 - in graphs 711
- textboxes 227
- TextSize property 205
- This reserved word 318
- three-dimensional borders 220
- three-dimensional graphs
- about 687
 - point of view 722
- Time keyword 565
- timer, setting in DataWindow objects 497
- times, display formats for 564
- timestamps, used in updating rows 547
- title bars
- in Function painter 138
 - in PowerScript painter 108
 - of descendent windows 192
- titles
- of graphs 683, 709
 - specifying text properties 711
- titles in OLE server application windows 789
- Toolbar Item Command dialog box 23
- toolbars
- about 18
 - controlling display of 19
 - custom buttons 23
 - customizing 20
 - docking 20
 - dropdown 18
 - in DataWindow painter 472
 - in PowerScript painter 28, 113
 - in Window painter 170
 - moving 19
 - moving buttons 21
 - resetting 22
- Toolbars dialog box 19
- Toolbox functions 896
- trace information
- analyzing 860
 - collecting 853
- trace mode, running in 901
- tracing an application 851
- Trail the Footer command 679
- Transaction objects
- default 60
 - defining descendent user object 303
- TreeView control prefix 201
- TreeView controls
- adding items 241
 - adding pictures 241
 - properties 242
 - using 241

TriggerEvent function 316

U

underline (_) character

defining accelerator keys for controls 214

in menu items 272

Undo command 116

in Database Administration painter 369

in DataWindow painter 478

unique indexes

creating 381

defining for primary key 377

unique keys, specifying for DataWindow 545

units of measure, specifying for DataWindow objects 497

Units Per Scroll Column 179

Units Per Scroll Line 180

UNIX, .WindU initialization file 36

up and down arrows, in Quick Select 433

updatable columns in DataWindow object 546

Update function 484

UPDATE statements

building in Database Administration painter 401

specifying WHERE clause 546

updates, in DataWindow objects 543

user events

communicating between user objects and windows 315

defining 325

in ancestor objects 251

in windows 185

overview 322

writing scripts for 328

user ID, check-out 85

user interface, design guidelines 161

user objects

autoinstantiating 304, 311

building custom visual 300

building new 298

building standard class 303

building standard visual 299

calling ancestor functions 256

communicating with windows 314

controls 201

custom class 297

custom visual 295

declaring events 322

external 295

in a Tab control 237

names, in windows 310

naming 306

overview 294

placing during execution 310

referring to, in Menu object scripts 283

saving 305

scripts, calling ancestor scripts 256

selecting from toolbar 24

standard class 296

standard visual 295

tab order within 211

triggering events 316

types of class 296

types of visual 294

using 309

using graphs in 688, 723

using inheritance 307

user-defined functions

access level 136

calling 146

changing name of 143

coding 138

defining 132

defining arguments 136

defining return types 134

finding where used 144

in ancestor objects 251

modifying 143

naming 134

return types 134

saving in files 139

types of 130

using 146

using structures 153

where used 144

with same name 131

V

validation rules

about 364, 582

- applying to columns 584
 - customizing error messages 587
 - defining in Database painter 584
 - defining in DataWindow painter 588
 - defining in Report painter 588
 - deleting 591
 - maintaining 591
 - removing 585
 - Value axis, graph 683
 - values
 - defining return types 134
 - ensuring validity of 374
 - fixed, cycling through 229
 - in graphs 683
 - of listbox items 231
 - of structures, copying 155
 - returning 139
 - setting tab 212
 - specifying for graphs 697
 - suppressing repeating 598
 - Variable Types tab page 312
 - variables
 - and Menu object scripts 282
 - in descendent menus 286
 - in retrieval arguments 451
 - in structures 150, 152
 - in window scripts 187
 - pasting 118
 - searching for 78
 - SQLCache 675
 - vers resources 899
 - version control, connecting 84
 - View Definition dialog box 387
 - View Entries Check Out Status dialog box 87
 - View painter
 - about 340
 - opening 384
 - views
 - about 384
 - creating 384
 - displaying in Select Tables dialog box 384
 - dropping 389
 - extended attributes of 365
 - opening 384
 - pipng 794
 - updating 543
 - Visible property 217, 269, 271
 - visible property, on the Expressions property page 653
 - visual user objects
 - custom 295
 - external 295
 - overview 294
 - placing in window or user object 309
 - standard 295
 - VScrollBar controls
 - prefix 201
 - using 237
- ## W
- warnings, compile 125
 - warnings, compiler 125
 - warnings, obsolete 125
 - WHERE clause
 - specified for update and delete 546
 - specifying in Quick Select 436
 - user modifying during execution 519
 - WHERE criteria 451
 - width property, on the Expressions property page 654
 - wildcards, in Library painter 72
 - window objects 160
 - Window painter
 - components of 170
 - customizing the toolbar 170
 - displaying hidden controls 217
 - opening 169
 - positioning StyleBar 170
 - Window Position dialog box
 - controlling scrolling 179
 - moving and sizing windows 175
 - window scripts
 - calling ancestor scripts 256
 - displaying popup menus 292
 - identifying Menu objects in 291
 - Window Style dialog box
 - in Window painter 172
 - Menu checkbox 291
 - window type, specifying 173
 - window-level properties 186
 - window-level variables 187
 - windows

- about 160
 - aligning controls 208
 - choosing icons for 178
 - communicating with user objects 314
 - creating new 169
 - custom MDI frames 168
 - declaring events 322
 - displaying references to 51
 - guidelines when designing 161
 - MDI frames 162
 - naming 182
 - placing controls in 197
 - placing visual user objects in 309
 - previewing 183
 - printing 184
 - referring to, in scripts 282
 - running 189
 - saving 181
 - selecting controls 198
 - sizing and positioning 175
 - specifying color 175
 - style 172
 - types of 162
 - using graphs in 723
 - using menus 174, 260, 291
 - Windows messages, mapping to PowerBuilder 323
 - .WindU initialization file 36
 - WMF files
 - adding to DataWindow objects 524
 - and Picture controls 235
 - and PictureBox controls 223
 - naming in resource files 893
 - working copy
 - checking in 83, 88
 - specifying library 86
 - workspace
 - grid, in Window painter 207
 - in Application painter 50
 - in Data Pipeline painter 799, 801
 - in Database painter 343
 - in DataWindow painter 469
 - in Function painter 138
 - in Menu painter 263
 - in User Object painter 299
 - in Window painter 170
 - of descendent menu 285
 - wrap height, default in Freeform reports 429
- X**
- X and Y values
 - and window position 175
 - in grid 208
 - x property, on the Expressions property page 654
 - x1, x2 property, on the Expressions property page 654
- Y**
- y property, on the Expressions property page 655
 - y1, y2 property, on the Expressions property page 655
 - years in DataWindow objects, specified with two digits 563
 - years in reports, specified with two digits 563
- Z**
- zero display format 561
 - Zoom command
 - in DataWindow painter 478
 - in print preview 487