

## Contents:

### About This Document

## I. Editor Environment

### 1. Introducing Panther Tools

Panther User Interface Components .....	1-1
---	-----

### 2. Editor Basics

Starting the Editor .....	2-2
Opening and Creating Application Components .....	2-3
Opening and Creating a Library .....	2-7
Opening a Middleware Session .....	2-9
Opening a Database Connection .....	2-12
Creating Widgets .....	2-13
Setting Properties .....	2-14
Saving Your Work .....	2-16
Creating and Opening a Repository .....	2-19
Saving Application Components to a Repository .....	2-23
Controlling Inheritance .....	2-29

### 3. Editor Workspace

Editor Workspace .....	3-2
Types of Widgets .....	3-19
Properties Window .....	3-27

## II. Wizards

### 4. Introducing the Wizards

How the Wizards Work .....	4-2
Wizard Templates .....	4-2

---

Navigating in the Wizards .....	4-6
<b>5. Screen Wizard</b>	
Invoking the Screen Wizard .....	5-1
Screen Wizard Dialogs .....	5-3
Output for Two-Tier Architecture .....	5-23
Output for JetNet and Oracle Tuxedo Applications .....	5-25
<b>6. Report Wizard</b>	
Invoking the Report Wizard .....	6-1
Report Types.....	6-3
Report Wizard Dialogs .....	6-12
How It Looks .....	6-23
Including Graphs .....	6-26

### III. User Interface Components

<b>7. Defining Screen Properties</b>	
Creating, Opening, and Saving Screens .....	7-2
Resizing a Screen.....	7-6
Using the Screen Grid.....	7-10
Controlling Screen Location.....	7-11
Controlling Geometry Changes Across Platforms .....	7-12
Including Screen Wallpaper .....	7-17
Specifying a Mouse Pointer.....	7-19
Including Borders and Decorations .....	7-21
Specifying Styles under Windows.....	7-25
Attaching a Menu Bar.....	7-27
Documenting Screens .....	7-29
Defining XML Tags for Screens .....	7-31
Testing Screens.....	7-32
<b>8. Defining Service Components</b>	
Creating and Saving Service Components .....	8-1
Defining the Component Interface .....	8-4
COM Applications.....	8-9

Enterprise JavaBeans.....	8-11
JetNet and Oracle Tuxedo Applications.....	8-15

## 9. Defining Widget Behavior

Identifying and Naming Widgets .....	9-2
Invoking a Popup Menu .....	9-6
Active Versus Inactive Widgets .....	9-7
Protecting Widgets .....	9-8
Double-Click Events .....	9-11
Setting Tabbing Order .....	9-12
Synchronizing Scrolling Arrays .....	9-20
Performing Calculations and Validating Numbers.....	9-26
Documenting Widgets .....	9-30
Defining XML Tags for Widgets .....	9-31

## 10. Manipulating Widgets

Selecting a Widget.....	10-1
Resizing a Widget.....	10-5
Moving and Copying Widgets.....	10-11
Arranging Widgets .....	10-13
Controlling a Widget's Position at Runtime .....	10-18
Undoing Actions.....	10-20

## 11. Controlling the Way Things Look

Giving Widgets Initial Data.....	11-2
Changing Widget Type.....	11-3
Formatting Text .....	11-3
Creating Shifting/Scrolling Fields.....	11-13
Creating a Date and Time Field.....	11-17
Defining a Numeric Format.....	11-20
Giving Screens a 3D Appearance.....	11-23

## 12. Specifying Colors

Color Property Types .....	12-1
Changing Color Properties .....	12-3
Setting Display Attributes .....	12-6

---

## 13. Providing Help Facilities

Determining the Level of Help Requests .....	13-1
Providing Status Line Text .....	13-2
Using Panther Help Screens .....	13-5
Using Selection Screens .....	13-10
Using Table Lookup Screens.....	13-13
Using External Help Sources.....	13-14
Using Tooltips .....	13-15

## 14. Display Widgets

Labeling Information.....	14-1
Creating Output Labels.....	14-3
Using Graphs to Display Data.....	14-4
Pie Charts.....	14-31
Bar/Line Graphs .....	14-39
XY-Plots .....	14-45
High/Low Charts .....	14-48

## 15. Data Entry Widgets

Defining the Input with Keystroke Filters.....	15-2
Controlling What Occurs on Input .....	15-13
Entering Data on Multiple Lines .....	15-17
Using Input Devices .....	15-19

## 16. Grid Widgets

Creating and Editing a Grid Widget.....	16-1
Manipulating a Grid Widget and Grid Members.....	16-2
Grid Widget Features.....	16-4
Moving and Resizing Grid Columns at Runtime .....	16-13
Grid Focus Properties .....	16-14

## 17. Tab Controls

Creating and Editing a Tab Control.....	17-1
Manipulating a Tab Deck and Tab Cards.....	17-3
Creating a Tab Dialog Screen.....	17-5

---

Tab Control Properties .....	17-6
------------------------------	------

## **18. Framesets and Splitters**

Creating And Editing Framesets .....	18-1
Programming With Framesets .....	18-6
Runtime Properties .....	18-12
Web Deployment of Framesets .....	18-14
The Frameset Sample Application .....	18-16

## **19. ActiveX Controls**

Embedding ActiveX Controls in Application Screens .....	19-2
Interacting with ActiveX Controls .....	19-6
Using AxView, the COM Control Viewer .....	19-13

## **20. Push Button Widgets**

Identifying a Push Button .....	20-2
Using Pictures on Push Buttons .....	20-4
Establishing Push Button Behavior .....	20-5

## **21. Selection Widgets**

Using Check Boxes .....	21-2
Using List Boxes .....	21-3
Using Radio Buttons.....	21-7
Using Toggle Buttons.....	21-9
Grouping Selection Widgets.....	21-10

## **22. Graphics Widgets**

Using Boxes and Lines .....	22-1
Using Boxes as Positioning Regions .....	22-6
Adding Borders to Widgets .....	22-9
Displaying a Picture on Widgets .....	22-10
Using Customer Drawn Widgets.....	22-12

## **23. Table Views and Links**

Using Table Views .....	23-2
Using Links .....	23-12

---

## IV. Editors

### 24. Styles Editor

Starting the Styles Editor .....	24-1
Transaction Modes .....	24-3
Using Styles and Classes .....	24-5
Defining a New Transaction Style.....	24-8
Creating a New Transaction Class.....	24-12
Testing Styles .....	24-13
Saving a Styles File .....	24-13

### 25. JIF Editor

Starting the JIF Editor.....	25-2
The JIF Editor Workspace .....	25-3
Defining and Updating Services .....	25-5
Defining and Updating Service Groups .....	25-12
Defining and Updating Queues .....	25-15
JPL Code Generation.....	25-21

### 26. Menu Bar Editor

Starting the Menu Bar Editor.....	26-1
Creating Menus.....	26-3
Creating Submenus .....	26-4
Menu Properties.....	26-6
Item Properties.....	26-7
Item Types .....	26-11
Displaying Pictures on Toolbar Items .....	26-14
Using the Menu List Window .....	26-16
Testing Menus .....	26-18
Saving Menus .....	26-19

## V. Appendices

### A. Keyboard Interface

Navigating Between Screens in the Editor .....	A-1
--	-----

---

Creating Widgets .....	A-3
Selecting and Manipulating Widgets.....	A-4

## **B. Wizard Output**

Screen Wizard Output Property Specifications .....	B-1
Report Wizard Output Property Specifications .....	B-8

## **Index**







# Panther

## Using the Editors

***Prolifics.***

Release 5.51

Document 0404  
March 2017

## Copyright

This software manual is documentation for Panther® 5.51. It is as accurate as possible at this time; however, both this manual and Panther itself are subject to revision.

Prolifics, Panther and JAM are registered trademarks of Prolifics, Inc.

Adobe, Acrobat, Adobe Reader and PostScript are registered trademarks of Adobe Systems Incorporated.

CORBA is a trademark of the Object Management Group.

FLEX/m is a registered trademark of Flexera Software LLC.

HP and HP-UX are registered trademarks of Hewlett-Packard Company.

IBM, AIX, DB2, VisualAge, Informix and C-ISAM are registered trademarks and WebSphere is a trademark of International Business Machines Corporation.

INGRES is a registered trademark of Actian Corporation.

Java and all Java-based marks are trademarks or registered trademarks of Oracle Corporation.

Linux is a registered trademark of Linus Torvalds.

Microsoft, MS-DOS, ActiveX, Visual C++ and Windows are registered trademarks and Authenticode, Microsoft Transaction Server, Microsoft Internet Explorer, Microsoft Internet Information Server, Microsoft Management Console, and Microsoft Open Database Connectivity are trademarks of Microsoft Corporation in the United States and/or other countries.

Motif, UNIX and X Window System are a registered trademarks of The Open Group in the United States and other countries.

Mozilla and Firefox are registered trademarks of the Mozilla Foundation.

Netscape is a registered trademark of AOL Inc.

Oracle, SQL\*Net, Oracle Tuxedo and Solaris are registered trademarks and PL/SQL and Pro\*C are trademarks of Oracle Corporation.

Red Hat and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc. in the United States and other countries.

Sybase is a registered trademark and Client-Library, DB-Library and SQL Server are trademarks of Sybase, Inc.

VeriSign is a trademark of VeriSign, Inc.

Other product names mentioned in this manual may be trademarks or registered trademarks of their respective owners, and are used for identification purposes only.

Send suggestions and comments regarding this document to:

Technical Publications Manager

Prolifics, Inc.

24025 Park Sorrento, Suite 405

Calabasas, CA 91302

<http://prolifics.com>

[support@prolifics.com](mailto:support@prolifics.com)

(800) 458-3313

© 1996-2017 Prolifics, Inc.

All rights reserved.

# Contents:

## About This Document

What You Need to Know .....	xxx
Documentation Website .....	xxx
How to Print the Document.....	xxx
Documentation Conventions .....	xxx
Contact Us! .....	xxx

## Part I. Editor Environment

### 1. Introducing Panther Tools

Panther User Interface Components.....	1-1
Editors .....	1-1
Wizards.....	1-2
Application Building Blocks.....	1-2
Properties Window .....	1-3
Libraries .....	1-3
Repositories.....	1-4
Importer.....	1-4
Transaction Manager.....	1-5

### 2. Editor Basics

Starting the Editor.....	2-2
How to Start the Editor.....	2-2
Editor Startup options.....	2-3

---

Opening and Creating Application Components.....	2-3
Creating a New Application Component .....	2-4
How to Create a New Application Component, Library or Repository ...	2-4
Opening Application Components .....	2-4
Viewing the Library Table of Contents .....	2-5
Opening and Creating a Library .....	2-7
How to Open a Library .....	2-7
How to Open a Remote Library .....	2-7
Creating a Library .....	2-8
How to Create a Library .....	2-8
How to Create a New Remote Library .....	2-9
Opening a Middleware Session .....	2-9
How to Open a Middleware Session .....	2-9
Opening a Database Connection.....	2-12
How to Connect to the Database within the Editor .....	2-12
How to Close a Database Connection within the Editor .....	2-13
Creating Widgets .....	2-13
How to Create a Widget .....	2-13
Setting Properties.....	2-14
Using the Properties Window.....	2-14
How to Set Properties.....	2-15
Saving Your Work.....	2-16
Saving an Application Component.....	2-16
How to Save an Application Component.....	2-17
How to Check for Overlapping Widgets.....	2-17
Naming Conventions .....	2-18
Creating and Opening a Repository.....	2-19
Creating a Repository .....	2-20
How to Create a Repository .....	2-20
How to Create a New Remote Repository .....	2-20
Opening Repository Entries .....	2-20
How to Open a Repository and Its Repository Entries .....	2-21
Viewing the Repository Table of Contents .....	2-22
Saving Application Components to a Repository.....	2-23

---

How to Save a Repository Entry .....	2-24
Populating a Repository with Database Objects .....	2-25
How to Import Database Tables from Your Database .....	2-25
What You Get When You Import Tables .....	2-27
Updating Database Views .....	2-28
Re-importing Database Tables .....	2-28
How to Re-import Database Tables .....	2-28
Controlling Inheritance.....	2-29
How to Turn On Inheritance for a Property .....	2-29
How to Turn Off Inheritance for a Property .....	2-30
Preventing Propagation of Changes .....	2-30
How to Restore Inheritance of All Property Values .....	2-31
Finding the Source of Inheritance .....	2-31
How to Find Parent Widgets or Screens .....	2-31
How to Find Child Widgets or Screens .....	2-32

### **3. Editor Workspace**

Editor Workspace .....	3-2
Editor Windows.....	3-3
Menu Bar .....	3-4
File Menu .....	3-4
Edit Menu.....	3-6
Create Menu .....	3-7
View Menu.....	3-8
Options Menu.....	3-9
Tools Menu .....	3-12
Windows Menu .....	3-13
Report Menu .....	3-14
Toolbars.....	3-14
Dockable Toolbars .....	3-15
Types of Widgets.....	3-19
Display-Type Widgets .....	3-19
Data Entry Widgets.....	3-20
Input Widgets.....	3-21
Command Widget .....	3-22

---

Selection Widgets.....	3-22
Graphics Widgets .....	3-24
Database Widget .....	3-24
COM Widget .....	3-25
Tab Controls.....	3-25
Hidden Widgets.....	3-25
Report Widgets.....	3-26
Frameset Widgets.....	3-27
Properties Window .....	3-27
Property Categories .....	3-30
Widget Properties.....	3-30
Screen Properties.....	3-32

## Part II. Wizards

### 4. Introducing the Wizards

How the Wizards Work .....	4-2
Wizard Templates.....	4-2
Description of the Templates.....	4-3
Customizing the Templates .....	4-5
Navigating in the Wizards .....	4-6

### 5. Screen Wizard

Invoking the Screen Wizard .....	5-1
How to Run the Screen Wizard.....	5-2
Screen Wizard Dialogs .....	5-3
Format Selection Dialog.....	5-3
How to Define the Screen's Format .....	5-4
First Master Dialog.....	5-5
How to Identify the First Master Table of Your Screen .....	5-6
Primary Key Selection Dialog.....	5-8
How to Define a Primary Key.....	5-9
Additional Master Dialog.....	5-9
How to Include Columns from Other Tables in the Master Section .....	5-10
First Detail Dialog .....	5-11

---

How to Identify the First Detail Table of Your Screen.....	5-12
Additional Detail Dialog .....	5-12
How to Include Columns from Additional Detail Tables .....	5-12
Master-Detail Link Dialog .....	5-13
How to Define a Master-Detail Link .....	5-14
First Subdetail Dialog.....	5-15
How to Identify the First Table of The Subdetail Section .....	5-15
Layout Selection Dialog.....	5-15
How to Define Each Section's Layout .....	5-16
Application Model Selection Dialog.....	5-17
How to Define the Architecture of Your Screens .....	5-17
Service Definition Dialog.....	5-19
How to Edit Service Names .....	5-20
Style and Finish Dialog .....	5-21
How to Add Final Touches to Your Screen .....	5-21
Output for Two-Tier Architecture .....	5-23
Two-Tier Client Screen and How It Looks.....	5-24
Two-Tier Selection Screens .....	5-25
Output for JetNet and Oracle Tuxedo Applications .....	5-25
Three-Tier Output and How It Looks .....	5-26
Three-Tier Client Screen.....	5-26
Service Component.....	5-27
Three-Tier Selection Screens .....	5-28
Selection Service Components.....	5-28

## **6. Report Wizard**

Invoking the Report Wizard .....	6-1
How to Run the Report Wizard.....	6-2
Report Types .....	6-3
Record-by-Record .....	6-3
Column.....	6-5
Row .....	6-6
Graph.....	6-7
Matrix.....	6-9
Address Labels .....	6-11

---

Report Wizard Dialogs .....	6-12
Selecting Report Type .....	6-12
Choosing Data .....	6-13
Grouping Data .....	6-15
Matrix Data Groups.....	6-17
Including Totals and Graphs .....	6-18
Finishing Up .....	6-20
How It Looks .....	6-23
Layout Window .....	6-23
Report Structure Window.....	6-25
Including Graphs .....	6-26

## **Part III. User Interface Components**

### **7. Defining Screen Properties**

Creating, Opening, and Saving Screens .....	7-2
Creating a New Screen .....	7-2
How to Create a New Screen .....	7-3
Identifying Screens.....	7-3
Creating a Screen from a Repository.....	7-4
How to Add Objects to Your Screen from a Repository.....	7-4
Creating a Dialog Box .....	7-5
How to Define a Screen as a Dialog Box.....	7-5
Opening and Saving Screens .....	7-5
Resizing a Screen.....	7-6
Defining the Screen Size .....	7-6
How to Change the Size of the Current Screen.....	7-6
Conventions for Controlling Screen Size at Runtime .....	7-6
Changing the Viewport .....	7-6
Maximize and Minimize .....	7-7
Iconifying a Screen.....	7-9
How to Assign an Icon To Display on a Minimized Screen.....	7-9
Using the Screen Grid.....	7-10
How to Control the Size of the Positioning Grid .....	7-10
Controlling Screen Location.....	7-11



---

Controlling Geometry Changes Across Platforms .....	7-12
Spacing Widgets for Portability .....	7-12
How to Control the Amount of Space Between Widgets .....	7-12
Setting the Screen Margin .....	7-13
How to Control How Much Whitespace Exists Between the Screen's Inner Border and Its Contents .....	7-13
Controlling Widget Positions and Screen Size .....	7-13
How to Control How Widgets Move and the Screen Resizes .....	7-13
Example of Vertical and Horizontal Shrinking Properties .....	7-14
Including Screen Wallpaper .....	7-17
How to Assign a Wallpaper as a Screen Background .....	7-17
How to Specify a Wallpaper under Windows .....	7-19
Specifying a Mouse Pointer.....	7-19
How to Define a Unique Cursor Shape for a Screen.....	7-19
Custom Pointer Shapes.....	7-21
How to Create and Specify a Custom Pointer Shape.....	7-21
Including Borders and Decorations.....	7-21
Designing the Screen Borders .....	7-22
How to Define a Screen's Border Display .....	7-22
How to Define the Border Style for Character Mode Applications..	7-22
Displaying a Screen Title .....	7-23
How to Display a Title in the Screen's Border/Title Bar .....	7-23
Assigning a Mnemonic .....	7-23
Displaying a Title Bar .....	7-24
How to Define the Display of the Screen's Title Bar .....	7-24
Displaying a System Menu .....	7-25
How to Prevent Users from Closing a Screen from the System Menu.....	7-25
How to Suppress the Display of a System Menu .....	7-25
Specifying Styles under Windows.....	7-25
The Keep In Frame Property .....	7-26
The Parent Window Property .....	7-26
The Keep On Top Property .....	7-27
Runtime Access to the Window Options Properties .....	7-27
Attaching a Menu Bar .....	7-27

---

How to Attach a Menu to a Screen.....	7-28
Assigning a PopUp Menu.....	7-28
How to Display a Popup Menu that is Different from the Screen's Menu Bar.....	7-29
Testing Menus .....	7-29
Documenting Screens .....	7-29
Adding Comments.....	7-30
How to Add or Edit Comments on Your Screen.....	7-30
Including Additional Information.....	7-30
How to Include Additional Information on a Screen .....	7-30
Defining XML Tags for Screens .....	7-31
How to Set XML Properties for Screens.....	7-31
Testing Screens.....	7-32

## **8. Defining Service Components**

Creating and Saving Service Components .....	8-1
How to Create a Service Component .....	8-2
Creating with the Screen Wizard .....	8-3
Identifying Service Components.....	8-3
Opening and Saving Service Components.....	8-3
Defining the Component Interface .....	8-4
Defining Methods.....	8-4
How to Add or Change a Method .....	8-4
How to Set the Method's Return Type .....	8-5
Parameters .....	8-5
How to Add a Parameter to the Method.....	8-5
How to Generate a Parameter List .....	8-6
Defining Properties.....	8-6
How to Add or Modify a Property .....	8-7
COM Applications.....	8-9
How to Generate a New CLSID.....	8-10
Saving COM Components.....	8-11
Deploying COM Components.....	8-11
Enterprise JavaBeans .....	8-11
The Panther EJB Tab Card.....	8-12

General .....	8-12
Transaction.....	8-13
Environment.....	8-13
Saving Service Components and Generating EJBs.....	8-14
Deploying EJBs on IBM WebSphere Application Server .....	8-14
JetNet and Oracle Tuxedo Applications.....	8-15
Defining Services in JetNet and Oracle Tuxedo applications .....	8-15
Using Wizard-Generated Service Components.....	8-15
Testing Service Components.....	8-15

## 9. Defining Widget Behavior

Identifying and Naming Widgets .....	9-2
Assigning a Widget Name.....	9-3
How to Assign a Name to a Selected Widget .....	9-3
Assigning a Mnemonic .....	9-4
Using Mnemonics .....	9-5
Specifying the Widget's Data Type.....	9-5
Invoking a Popup Menu .....	9-6
How to Invoke a Popup Menu for a Widget or Specific Widgets.....	9-6
Active Versus Inactive Widgets .....	9-7
How to Initialize a Widget's State .....	9-7
How to Change a Widget's State at Runtime .....	9-7
Protecting Widgets .....	9-8
Focus Protection.....	9-8
How to Control a Widget's Focus Protection.....	9-8
Validation Protection.....	9-9
How to Protect a Field from Being Validated When the User Exits the Field .....	9-9
Input Protection.....	9-10
How to Control Data Entry to a Widget .....	9-10
Clearing Protection.....	9-10
How to Define Whether a Widget's Data Can Be Cleared or Not ....	9-11
Double-Click Events .....	9-11
How to Assign a Double-Click Event .....	9-12
Setting Tabbing Order .....	9-12

---

Forward Tabbing Order.....	9-13
Backward Tabbing Order.....	9-13
Changing the Tabbing Order.....	9-13
How to Define the Tabbing Order for a Widget or Group.....	9-14
Specifying Tab Stops.....	9-14
Using a Field Number.....	9-15
Specifying an Occurrence.....	9-15
Specifying a Group.....	9-16
Changing the Tab Order.....	9-16
In Horizontal Arrays.....	9-16
In Vertical Arrays.....	9-17
Autotabbing.....	9-19
How to Control Autotabbing.....	9-19
Synchronizing Scrolling Arrays.....	9-20
Array Behavior.....	9-20
Logical Key Behavior.....	9-21
Automatic Synchronization.....	9-21
Creating and Modifying Synchronized Arrays.....	9-22
How to Synchronize a Set of Scrolling Arrays.....	9-22
How to Change the Size and/or Behavior of a Synchronized Scrolling Group.....	9-22
Identifying Members of the Synchronized Scrolling Group.....	9-23
How to Identify All Members of a Synchronized Scrolling Group ..	9-23
Changing Members of a Synchronized Scrolling Group.....	9-24
How to Add a Scrolling Array to an Existing Group.....	9-24
How to Isolate an Array That is Synchronized Because It Belongs to a Table View or to Table Views Joined by a Server View.....	9-24
How to Isolate an Array That is Part of a Synchronized Scrolling Group 9-24	9-24
Sorting Data in Synchronized Scrolling Arrays.....	9-25
Using an Alternative Scroll Driver.....	9-25
Performing Calculations and Validating Numbers.....	9-26
Math Expression Syntax.....	9-26
How to Attach a Math Calculation to a Selected Widget.....	9-26
Using Functions.....	9-28

---

Check Digit Calculations .....	9-29
How to Attach a Check Digit Calculation to a Widget.....	9-29
Documenting Widgets .....	9-30
Adding Comments.....	9-30
How to Add or Edit Comments for Your Widget.....	9-30
Including Additional Information .....	9-31
How to Include Additional Information for a Widget .....	9-31
Defining XML Tags for Widgets .....	9-31
How to Set XML Properties for Widgets.....	9-32
Generating the XML for Widgets .....	9-32
Processing XML for Multiple Occurrences.....	9-33
Importing XML to Screens .....	9-33

## **10. Manipulating Widgets**

Selecting a Widget.....	10-1
Using a Mouse.....	10-2
Mouse Click .....	10-2
Rubberbanding .....	10-2
Deselecting Widgets.....	10-3
Using the Widget List .....	10-3
Resizing a Widget.....	10-5
Defining a Widget's Size .....	10-6
Dragging to Size.....	10-6
Specifying a Size.....	10-6
Unifying Widget Size.....	10-8
Controlling a Widget's Size at Runtime .....	10-9
General Rules for Resizing at Runtime.....	10-10
Moving and Copying Widgets.....	10-11
Within a Screen .....	10-11
How to Move a Widget Within a Screen .....	10-11
How to Copy a Widget Within a Screen.....	10-11
From Screen to Screen .....	10-12
How to Move a Widget From One Screen To Another .....	10-12
How to Copy a Widget To Another Screen .....	10-12
To and From a Repository.....	10-12

---

Deleting Widgets .....	10-13
How to Delete or Remove a Widget .....	10-13
Arranging Widgets .....	10-13
Aligning Widgets.....	10-14
How to Align Widgets with Each Other .....	10-14
How to Align Widgets to an Absolute Position or Coordinate .....	10-14
How to Position a Widget on a Grid Coordinate .....	10-15
How to Force All Widget Placement on a Grid Coordinate .....	10-15
Spacing Widgets.....	10-16
How to Create Automatic Spacing.....	10-16
How to Create Custom Spacing .....	10-17
Centering Widgets .....	10-18
How to Center Widgets on a Screen .....	10-18
Controlling a Widget's Position at Runtime .....	10-18
How to Set a Widget's Position in Relation to Other Widgets .....	10-19
Undoing Actions.....	10-20
How to Undo an Action.....	10-20
How to Redo an Action .....	10-20
How to Set the Number of Undo Levels .....	10-20

## **11. Controlling the Way Things Look**

Giving Widgets Initial Data.....	11-2
How to Define Initial Text or a Label for a Widget .....	11-2
Changing Widget Type.....	11-3
How to Change Widget Types .....	11-3
Formatting Text .....	11-3
To Display or Not Display .....	11-4
How to Hide a Widget at Runtime .....	11-4
How to Hide the Contents of a Field.....	11-5
Justification.....	11-5
How to Change Text Justification in a Widget .....	11-5
How to Control the Text Justification on Box Widgets .....	11-6
Aligning Button Labels in Motif .....	11-6
Specifying Fonts.....	11-7
Assigning a Prolifics Font .....	11-9

---

How to Assign a Font to a Screen or to One or More Widgets .....	11-9
Assigning a GUI Font Name .....	11-11
How to Assign a GUI-Specific Font .....	11-11
Displaying Null Values .....	11-12
How to Allow a Widget to Accept and Display a Null Value .....	11-12
Controlling Grid Formats .....	11-13
Creating Shifting/Scrolling Fields .....	11-13
Creating an Array .....	11-14
How to Create an Array with More than One Occurrence .....	11-14
Creating a Scrolling Array .....	11-15
How to Define Offscreen Occurrences for a Selected Widget .....	11-15
Creating a Shifting Field .....	11-16
How to Create a Shifting Field for a Selected Widget.....	11-16
Creating a Date and Time Field.....	11-17
How to Set a Date/Time Format for a Selected Widget.....	11-17
Defining a Custom Date/Time Format.....	11-18
Defining a Numeric Format.....	11-20
How to Set a Numeric Format for a Selected Widget.....	11-20
Giving Screens a 3D Appearance.....	11-23
How to Implement the 3D Feature for a Screen.....	11-23
Three-Dimensional Screen and Widget Appearance .....	11-24
Screens .....	11-24
Widgets .....	11-24

## 12. Specifying Colors

Color Property Types .....	12-1
Using Panther Basic Colors.....	12-2
Scheme Color Specification .....	12-2
Extended Color Specification.....	12-3
Changing Color Properties .....	12-3
Setting Color Properties .....	12-3
How to Change an Object's Color Via the Properties Window .....	12-3
How to Set an Object's Color with the Color Palette.....	12-4
How to Add an Extended Color that Is Not Listed.....	12-6
Setting Display Attributes .....	12-6

## 13. Providing Help Facilities

Determining the Level of Help Requests .....	13-1
Providing Status Line Text .....	13-2
Adding Display Attributes to Status Text .....	13-3
Displaying Key Names on the Status Line .....	13-4
Sounding a Message Bell .....	13-4
Using Panther Help Screens .....	13-5
Creating Internal Help Screens.....	13-5
Display-Only Help Screen .....	13-6
Data Entry Help Screen.....	13-6
Multilevel Help System.....	13-8
Attaching Panther Help Screens.....	13-8
How to Assign a Panther Help Screen to a Widget or a Screen.....	13-8
Positioning Help Screens.....	13-9
Using Selection Screens .....	13-10
Creating a Selection Screen.....	13-10
Attaching a Selection Screen.....	13-12
How to Assign a Selection Screen .....	13-12
Using Table Lookup Screens.....	13-13
Creating and Attaching Table Lookup .....	13-13
How to Attach a Lookup Screen to a Data Entry Widget .....	13-14
Using External Help Sources.....	13-14
Attaching Help from Another Source.....	13-15
How to Attach an External Help to an Application Object.....	13-15
Using Tooltips .....	13-15
Adding Tooltips in the Editor.....	13-16
Controlling Tooltip Appearance.....	13-16
Examples .....	13-17

## 14. Display Widgets

Labeling Information .....	14-1
Defining the Look of a Static Label .....	14-2
How to Define the Content of a Static Label .....	14-2



---

How to Resize a Static Label To Fit Its Content.....	14-2
How to Define a Static Label's Color.....	14-3
Creating Output Labels.....	14-3
Defining Output Labels.....	14-3
How to Change the Size Dynamically When Content Changes .....	14-4
Using Graphs to Display Data.....	14-4
How to Create a Graph Widget .....	14-5
Controlling Graph Text.....	14-7
How to Specify a Font for the Graph.....	14-8
Specifying Sizes for Graph Text.....	14-9
Adding a Title to a Graph.....	14-9
How to Place a Title on the Graph.....	14-9
How to Place a Subtitle on the Graph .....	14-10
Adding a Legend to a Graph .....	14-10
How to Add a Legend to the Graph Widget .....	14-10
How to Specify the Legend Placement by Location.....	14-12
How to Specify the Legend Placement by Position.....	14-13
Specifying Text Size for Graph Labels .....	14-13
How to Specify Text Size for Graph Labels .....	14-13
Orienting the Graph Vertically or Horizontally .....	14-14
How to Specify the Graph Orientation .....	14-15
Describing the X and Y Axes of a Graph.....	14-15
How to Specify the Axis Locations .....	14-16
How to Place Labels on the X/Y Axes .....	14-17
Placing Tick Marks on the Axes .....	14-18
How to Specify Minimum and Maximum Values for the X/Y Axes .....	14-19
How to Specify the Type of Scale on the X/Y Axes .....	14-19
How to Specify Tick Mark Increments on the X/Y Axes.....	14-19
How to Specify Tick Mark Style on the X/Y Axes .....	14-20
How to Specify Tick Mark Width on the X/Y Axes.....	14-20
How to Label the Tick Marks on the X/Y Axes .....	14-20
How to Show Tick Marks as Grid Lines.....	14-21
Creating a 3-Dimensional Effect.....	14-22
How to Display a Graph in 3D.....	14-22

---

Defining the Data Series.....	14-23
How to Specify Data Values .....	14-24
How to Specify the Plot Style for a Data Series .....	14-26
How to Specify the Line Style for a Data Series.....	14-28
How to Specify the Line Width for a Data Series .....	14-28
How to Specify the Point Marker Format for a Data Series .....	14-29
How to Specify the Y Axis for a Data Series.....	14-29
How to Specify the Color for a Data Series .....	14-29
How to Enter Legend Text for a Data Series .....	14-30
Converting Data between Chart Types.....	14-30
Pie Charts.....	14-31
How to Create a Pie Chart .....	14-32
How to Display a Pie Chart in 3D .....	14-33
How to Specify the Data Source for a Pie Chart .....	14-33
Describing the Chart Layout .....	14-34
How to Specify the Size of the Pie Chart.....	14-34
How to Specify the Position of the Pie Chart.....	14-34
How to Specify Segment Order and Position.....	14-35
Describing Segment Characteristics.....	14-35
How to Identify the Segments of a Pie Chart .....	14-35
How to Specify the Style for Each Pie Chart Segment .....	14-37
Bar/Line Graphs .....	14-39
How to Create a Bar/Line Graph.....	14-39
How to Format the Bar Type.....	14-41
How to Specify the Bar Type of a Bar/Line Graph.....	14-41
How to Display a Bar/Line Graph in 3D.....	14-42
How to Specify Value Sources for a Bar/Line Graph .....	14-44
How to Display the Values for a Data Series .....	14-45
XY-Plots .....	14-45
How to Create an XY-Plot .....	14-45
How to Display an XY-Plot in 3D .....	14-46
How to Specify the X and Y Value Sources .....	14-47
High/Low Charts .....	14-48
How to Create a High/Low Chart.....	14-49
How to Display a High/Low Chart in 3D.....	14-50

---

How to Specify Value Sources for a High/Low Chart.....	14-52
--	-------

## **15. Data Entry Widgets**

Defining the Input with Keystroke Filters.....	15-2
How to Define an Input Data Filter for a Selected Widget.....	15-2
Digits Only Filter .....	15-2
How to Embed Special Characters in a Digits-Only Field .....	15-3
Yes/No Entries .....	15-3
Character and Numeric Edits .....	15-4
How to Embed Punctuation or Special Characters, Such as Hyphens, Periods, and Pound Signs (#).....	15-4
Alphabetic Edits.....	15-4
Numeric Edits .....	15-4
Alphanumeric Edits.....	15-5
Edit Masks.....	15-5
How to Define an Edit Mask.....	15-5
How to Specify a Range or Character/Numeric Restriction with an Edit Mask.....	15-6
Examples of Edit Masks .....	15-6
Regular Expressions.....	15-7
How to Define a Character-Level Regular Expression.....	15-8
How to Define a Field-Level Regular Expression.....	15-8
Examples of Regular Expressions.....	15-8
Constructing Expressions.....	15-9
Converting Case .....	15-13
How to Ensure that Data is Entered in the Desired Case.....	15-13
Controlling What Occurs on Input .....	15-13
Specifying a Range of Values .....	15-13
How to Set a Range.....	15-13
Defining Prerequisites or Restrictions on Input Data.....	15-14
Required.....	15-14
Must Fill.....	15-15
Select on Entry.....	15-15
Input Protection.....	15-16
Protect from Clearing.....	15-16

---

Entering Data on Multiple Lines .....	15-17
Defining a Word Wrap Field.....	15-18
Word Wrap Function.....	15-18
Using Input Devices .....	15-19
Using Scales .....	15-19
How to Implement a Scale Widget .....	15-19
How to Set the Scale to an Initial Value .....	15-20
How to Allow a Decimal Value Specification on the Scale .....	15-20
Using Option Menus and Combo Boxes .....	15-20
Populating the List with Constant Data.....	15-21
Defining the Number of Occurrences .....	15-22
Using Data from an External Source.....	15-22
Defining Initial Setting.....	15-24

## 16. Grid Widgets

Creating and Editing a Grid Widget .....	16-1
How to Create and Populate a Grid Widget .....	16-1
Manipulating a Grid Widget and Grid Members.....	16-2
How to Select a Grid Widget.....	16-2
How to Resize a Grid Widget.....	16-3
How to Select One or More Grid Members .....	16-3
How to Remove a Grid Member from the Grid .....	16-3
How to Delete the Grid Widget, but not the Widgets within the Grid.....	16-4
Grid Widget Features.....	16-4
Specifying Grid Fonts.....	16-5
Displaying Columns .....	16-5
Defining the Grid's Width.....	16-5
Displaying Column Titles .....	16-6
Defining Stationary Columns.....	16-7
Defining Column Separators .....	16-7
Displaying Rows .....	16-8
Defining the Number of Rows .....	16-8
Displaying Row Titles.....	16-9
Selecting Rows of Data.....	16-10
Defining Row Separators .....	16-10

---

Changing Row Height.....	16-11
Defining Column Click Behavior .....	16-11
Sorting Items .....	16-11
Writing a Custom Sort Function .....	16-12
Specifying a Function .....	16-12
Moving and Resizing Grid Columns at Runtime .....	16-13
How to Allow Users To Move and Resize Grid Members .....	16-13
How to Expand a Single Grid Cell to Its Maximum Length.....	16-14
Grid Focus Properties .....	16-14
Entry and Exit Function Properties .....	16-15
Focus Protection Property .....	16-15
Row Entry Function Property .....	16-15
Row Exit Function Property .....	16-15

## 17. Tab Controls

Creating and Editing a Tab Control.....	17-1
How to Create and Populate a Tab Control.....	17-2
Manipulating a Tab Deck and Tab Cards.....	17-3
How to Select a Tab Deck.....	17-3
How to Make a Tab Card the Topmost Card .....	17-3
How to Select a Tab Card for Editing .....	17-3
How to Change the Tab Card Order.....	17-4
How to Move, Copy, or Delete a Tab Deck.....	17-4
General Notes About Tab Controls.....	17-4
Creating a Tab Dialog Screen.....	17-5
How to Build a Tab Dialog Screen .....	17-5
Tab Control Properties .....	17-6
Tab Deck Properties.....	17-6
Identity .....	17-7
Color.....	17-7
Font .....	17-7
Tab Card Properties.....	17-7
Identity .....	17-7
Geometry.....	17-8
Color.....	17-8

---

Focus .....	17-8
Card Entry and Exit.....	17-8
Expose and Hide Functions.....	17-9
Index Tab Functions.....	17-9
Order of Events .....	17-9
Validation .....	17-10
Format/Display.....	17-10

## 18. Framesets and Splitters

Creating And Editing Framesets.....	18-1
Splitter Widgets.....	18-2
Pane Widgets.....	18-4
Frameset Properties .....	18-5
Programming With Framesets .....	18-6
Opening a Frameset.....	18-6
Cursor Movement and Window Management using Framesets.....	18-7
Closing Framesets .....	18-9
Runtime Properties .....	18-12
Screen and Frameset Properties.....	18-12
Splitter Properties .....	18-12
Pane Properties .....	18-13
Web Deployment of Framesets .....	18-14
Frameset Web Properties.....	18-14
Splitter Web Properties.....	18-14
Pane Web Properties.....	18-16
The Frameset Sample Application .....	18-16

## 19. ActiveX Controls

Embedding ActiveX Controls in Application Screens .....	19-2
How to Embed an ActiveX Control .....	19-2
Setting ActiveX Properties .....	19-3
How to Select an ActiveX Control Registered on your Workstation	19-3
How to Use an ActiveX Control Unavailable on your Workstation.	19-4
How to Set Properties of the ActiveX Control.....	19-4
How to Set Unlisted ActiveX Control Properties .....	19-4

---

Setting Runtime Licensing .....	19-4
How to Enter the Runtime License .....	19-5
Setting Color Properties .....	19-5
Interacting with ActiveX Controls .....	19-6
Setting Properties at Runtime.....	19-6
Transferring Data to the ActiveX Control.....	19-8
Data Transfer Using JavaScript .....	19-8
Data Transfer Using VBScript.....	19-9
Submitting Data to the Web Application Server.....	19-9
Calling ActiveX Methods.....	19-9
Calling Methods Using VBScript .....	19-10
Specifying an ActiveX Event Handler.....	19-11
Using VBScript for Event Handling .....	19-12
Using AxView, the COM Control Viewer .....	19-13

## **20. Push Button Widgets**

Identifying a Push Button.....	20-2
How to Define the Push Button's Text .....	20-2
How to Adjust the Size to Fit its Textual Content .....	20-2
How to Identify the Keyboard Mnemonic .....	20-3
Using Pictures on Push Buttons .....	20-4
How to Assign a Bitmap or Pixmap to a Push Button .....	20-4
How to Create Images in your Application.....	20-5
Establishing Push Button Behavior .....	20-5
Assigning Default and Cancel.....	20-6
How to Specify a Default and/or Cancel Push Button.....	20-6
Setting the Initial State .....	20-6
How to Initialize the Push Button's Status .....	20-7
Attaching an Action to the Button.....	20-7
Displaying a Screen .....	20-8
Invoking a Function .....	20-9
Invoking a Program.....	20-9

## **21. Selection Widgets**

Using Check Boxes .....	21-2
-------------------------	------

---

Using List Boxes .....	21-3
How to Define a List Box's Behavior.....	21-3
Scrolling the Data.....	21-5
How to Make a Scrolling List Box.....	21-5
Populating List Boxes.....	21-6
How to Populate a List Box with Data.....	21-6
Performing Actions .....	21-7
How to Make a List Box Function Like a Menu.....	21-7
Using Radio Buttons.....	21-7
How to Define the Number of Selections.....	21-8
How to Establish an Initial Selection .....	21-8
Using Toggle Buttons.....	21-9
Grouping Selection Widgets.....	21-10
Creating Selection Groups.....	21-11
How to Create a Selection Group.....	21-11
Identifying the Members of a Group.....	21-11
How to Identify the Members of the Group.....	21-11
Specifying Group Properties .....	21-12
How to Select a Group from the Widget List.....	21-12
How to Select a Group via a Group Member.....	21-13
Naming a Group .....	21-14
How to Assign a Name to the Group .....	21-14
Setting the Number of Selections Allowed in a Group.....	21-14
How to Define the Number of Selections that a User Can Make in a Group .....	21-14
Specifying an Initial Selection.....	21-15
How to Identify an Initial Selection for a Group .....	21-15
Adding and Removing Group Members .....	21-16
How to Add a Widget or Widgets to an Existing Group .....	21-16
How to Remove a Widget from a Group .....	21-16

## **22. Graphics Widgets**

Using Boxes and Lines .....	22-1
How to Create a Line or Box.....	22-1
How to Control the Line or Box Placement.....	22-2



---

Specifying Styles for Boxes and Lines .....	22-2
How to Define a Line Style for a Box or Line Widget .....	22-2
Character Mode Lines and Boxes .....	22-3
GUI Lines and Boxes .....	22-3
Creating Frames .....	22-3
How to Include a Title on a Box .....	22-4
Adding Lines .....	22-6
Using Boxes as Positioning Regions .....	22-6
Spacing Widgets within a Box .....	22-7
Minimum Horizontal Space .....	22-7
Minimum Vertical Space .....	22-7
How to Define a Margin Within the Box Frame .....	22-7
How to Control the Box Dimensions and Widget Start Positions .....	22-8
Adding Borders to Widgets .....	22-9
How to Define a Widget Border and Style .....	22-9
How to Include a Title on a List Box or Multiline Text Widget .....	22-9
Displaying a Picture on Widgets .....	22-10
Specifying the Image .....	22-10
How to Display a Picture on a Widget .....	22-10
Portability of Images Files .....	22-11
Storing the Image Files .....	22-12
Sizing the Image .....	22-12
Using Customer Drawn Widgets .....	22-12

## **23. Table Views and Links**

Using Table Views .....	23-2
Accessing Table View Properties .....	23-2
Identity Properties .....	23-3
Transaction Properties .....	23-3
Database Properties .....	23-4
Server View Properties .....	23-6
Service Properties .....	23-7
Setting Table View Properties to Generate SQL .....	23-8
Creating and Linking a Table View .....	23-9
How to Create a New Table View .....	23-9

---

Manipulating Table View Members.....	23-10
How to Identify the Members of a Specific Table View .....	23-10
How to Identify the Table View Members Associated with a Specific Widget.....	23-10
Adding Table View Members .....	23-10
How to Add Widgets to a Table View .....	23-10
Identifying the Root Table View .....	23-11
How to Identify A Table View As The Root Table For The Screen .....	23-11
Using Links.....	23-12
Creating Links .....	23-12
How to Create a Link on Your Screen .....	23-13
Editing Link Properties.....	23-13
Identity Properties .....	23-13
Transaction Properties.....	23-14
Service Properties.....	23-16
Joining Database Tables .....	23-17
How to Join Two Tables with a One-to-Many Relationship .....	23-17
How to Join Two Tables Using a One-to-One Relationship.....	23-18
How to Define a Lookup Specification for Either Link Type.....	23-20
Setting Validation Links.....	23-21
How to Specify a Validation Link.....	23-21

## **Part IV. Editors**

### **24. Styles Editor**

Starting the Styles Editor.....	24-1
Transaction Modes .....	24-3
How Transaction Manager Commands Affect Modes.....	24-4
Using Styles and Classes .....	24-5
Using the Default Styles Settings.....	24-6
Using the Default Transaction Classes.....	24-6
Setting Classes for Menu Items and Push Buttons.....	24-7
Defining a New Transaction Style.....	24-8
Style Widget Properties.....	24-9

---

Identity Properties .....	24-9
Color Properties .....	24-10
Focus Properties .....	24-10
Input Properties .....	24-10
Validation Property .....	24-12
Creating a New Transaction Class .....	24-12
Testing Styles .....	24-13
Saving a Styles File .....	24-13

## 25. JIF Editor

Starting the JIF Editor .....	25-2
How to Start the JIF Editor .....	25-2
The JIF Editor Workspace .....	25-3
View Services Screen .....	25-3
Menu Bar .....	25-4
Defining and Updating Services .....	25-5
How to Define or Update a Service .....	25-6
Controlling Service Behavior .....	25-8
Transaction Type .....	25-9
Cache Service Component .....	25-10
Reply Options .....	25-10
Call Options .....	25-11
How to Delete a Service .....	25-12
Defining and Updating Service Groups .....	25-12
How to Define or Update a Service Group with Services .....	25-13
How to Delete a Service Group .....	25-14
Defining and Updating Queues .....	25-15
How to Define or Update an Independent Queue .....	25-15
How to Define or Update a Queue Associated with a Service .....	25-17
How to Transfer a Queue to Another Queuespace .....	25-19
How to Delete a Queue .....	25-20
JPL Code Generation .....	25-21
Service Code Generation .....	25-21
Output For Service Definition Code .....	25-22
Output For Service Request Code .....	25-22

---

Generating Service Code.....	25-22
Format of Template .....	25-23

## **26. Menu Bar Editor**

Starting the Menu Bar Editor.....	26-1
How to Invoke the Menu Bar Editor .....	26-1
Creating Menus.....	26-3
How to Create A New Menu .....	26-4
How to Insert New Menu Items .....	26-4
Creating Submenus .....	26-4
How to Attach An Existing Menu .....	26-5
How to Edit The New Menu .....	26-5
How to Specify An External Menu .....	26-6
Menu Properties.....	26-6
Item Properties.....	26-7
Item Types .....	26-11
Edit Types.....	26-12
Separator Styles.....	26-13
Displaying Pictures on Toolbar Items .....	26-14
Image Size .....	26-15
Image File Types .....	26-15
Filename Extensions.....	26-16
Using the Menu List Window .....	26-16
Testing Menus .....	26-18
Saving Menus .....	26-19

## **Part V. Appendices**

### **A. Keyboard Interface**

Navigating Between Screens in the Editor .....	A-1
Manipulating Screens .....	A-2
Creating Widgets .....	A-3
Selecting and Manipulating Widgets.....	A-4

---

## **B. Wizard Output**

Screen Wizard Output Property Specifications .....	B-1
Two-Tier Property Specifications .....	B-1
Screen Property Specifications .....	B-1
Widget Property Specifications .....	B-2
Selection Screen Property Specifications .....	B-3
Three-Tier Property Specifications .....	B-4
Screen Property Specifications .....	B-4
Widget Property Specifications .....	B-5
Selection Screen Property Specifications .....	B-7
Report Wizard Output Property Specifications .....	B-8

## **Index**



# About This Document

*Using the Editors* contains detailed information and instructions about using Panther's editors and wizards to build applications, including the screens, their components, and the control flow. The editors are:

- Screen editor for building screens, service components, and reports
- Menu bar editor
- Styles editor
- JIF editor

*Using the Editors* is organized into the following sections:

## Section One: Overview

Introduces you to the basic principles of using the authoring environment to design and build a presentation interface.

## Section Two: Wizards

Gives information about the wizard applications which help you quickly and easily build screens and reports.

## Section Three: User Interface Components

Provides the instructions you need to enhance the look of your application.

## Section Four: The Editors

Explains the [Styles editor](#), the [JIF editor](#), and the [Menu Bar editor](#).

---

## What You Need to Know

---

The guide takes the users of your application into account by presenting information from the end-users perspective. Consider what you want the user to see. Use the guide as a reference throughout the development process.

For the most part, the guide assumes that you are new to Panther. However, as an application development tool, it also assumes that you have familiarity with your development environment and with database terminology.

If you are new to Panther, you should first go through the *Getting Started* book. It guides you through the process of building a Panther application.

The Panther documentation set assumes you are using a mouse in your development environment. Refer to Appendix A, “Keyboard Interface,” for a listing of keyboard alternatives and accelerator keys for using the editors in character mode environments that don't support a mouse.

---

## Documentation Website

---

The Panther documentation website includes manuals in HTML and PDF formats and the Java API documentation in Javadoc format. The website enables you to search the HTML files for both the manuals and the Java API.

Panther product documentation is available on the Prolifics corporate website at <http://docs.prolifics.com/panther/>.



---

# How to Print the Document

---

You can print a copy of this document from a web browser, one file at a time, by using the File→Print option on your web browser.

A PDF version of this document is available from the Panther library page of the documentation website. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe website at <https://get.adobe.com/reader/otherversions/>.

---

# Documentation Conventions

---

The following documentation conventions are used throughout this document.

<b>Convention</b>	<b>Item</b>
Ctrl+Tab	Indicates that you must press two or more keys simultaneously. Initial capitalization indicates a physical key.
<i>italics</i>	Indicates emphasis or book titles.
UPPERCASE TEXT	Indicates Panther logical keys. <i>Example:</i> XMIT
<b>boldface text</b>	Indicates terms defined in the glossary.

Convention	Item
<code>monospace text</code>	<p>Indicates code samples, commands and their options, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.</p> <p><i>Examples:</i></p> <pre>#include &lt;smdefs.h&gt; chmod u+w * /usr/prolifics prolifics.ini</pre>
<i>monospace italic text</i>	<p>Identifies variables in code representing the information you supply.</p> <p><i>Example:</i></p> <pre>String expr</pre>
<code>MONOSPACE UPPERCASE TEXT</code>	<p>Indicates environment variables, logical operators, SQL keywords, mnemonics, or Panther constants.</p> <p><i>Examples:</i></p> <pre>CLASSPATH OR</pre>
<code>{ }</code>	<p>Indicates a set of choices in a syntax line. One of the items should be selected. The braces themselves should never be typed.</p>
<code> </code>	<p>Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.</p>
<code>[ ]</code>	<p>Indicates optional items in a syntax line. The brackets themselves should never be typed.</p> <p><i>Example:</i></p> <pre>formlib [-v] library-name [file-list]...</pre>
<code>...</code>	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none"><li>■ That an argument can be repeated several times in a command line</li><li>■ That the statement omits additional optional arguments</li><li>■ That you can enter additional parameters, values, or other information</li></ul> <p>The ellipsis itself should never be typed.</p> <p><i>Example:</i></p> <pre>formlib [-v] library-name [file-list]...</pre>

Convention	Item
.	Indicates the omission of items from a code example or from a syntax line.
.	The vertical ellipsis itself should never be typed.
.	

---

---

## Contact Us!

---

Your feedback on the Panther documentation is important to us. Send us e-mail at [support@prolifics.com](mailto:support@prolifics.com) if you have questions or comments. In your e-mail message, please indicate that you are using the documentation for Panther 5.50.

If you have any questions about this version of Panther, or if you have problems installing and running Panther, contact Customer Support via:

- Email at [support@prolifics.com](mailto:support@prolifics.com)
- Prolifics website at <http://profapps.prolifics.com>

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address and phone number
- Your company name and company address
- Your machine type
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

*Contact Us!*

---



# Part I Editor Environment

[Introducing Panther Tools](#)

[Editor Basics](#)

[Editor Workspace](#)



# 1 Introducing Panther Tools

*Using the Editors* introduces you to the user interface development component of the Panther framework. If you are new to Panther software, you should first go through *Getting Started*; it guides you through the process of building a Panther application. Refer to *Using the Editors* in the process of application development for more details about an aspect of the interface.

---

## Panther User Interface Components

---

User interface development components are the tools used to build applications. The following Panther user interface components are introduced and described in detail in this book: editors and wizards, screens and widget types, reports, service components, Properties window, libraries, and repositories.

### Editors

The Panther editors are some of the tools used to build your applications. The editors and their functions are as follows:

- **Editor**—Create and edit screens, reports, and service components from scratch, with the help of the wizards, or from visual objects that are automatically created as the result of importing database tables to a repository. The editor provides access to a test mode environment. Test mode simulates how your application behaves and appears to users. You can easily switch between edit and test modes.
- **Menu bar editor**—Create and edit menus that you can attach to individual screens via the editor or define for your application as a whole. In test mode, you can test screens that use the menu bar/toolbar, and quickly switch between edit and test modes.
- **JIF editor** (JetNet/Oracle Tuxedo only)—Create and edit the JIF with the help of the graphical JIF editor. The JIF contains information about services and service groups. For Oracle Tuxedo applications, the JIF also contains information about reliable queues in an application. The JIF is a central repository of all such information and is accessed at runtime to determine the requirements and specifications of services and reliable queues.  
**Note:** The JIF editor does not have a test mode environment.
- **Styles editor**—Enhance the transaction manager's control of the appearance and behavior of widgets on your database application screens.

## Wizards

The wizards are easy to use and guide you through the design process to create screens and reports that work with the transaction manager. The editor provides access to the Panther wizards:

- **Screen Wizard**—Design screens for two-tier, three-tier, and Web architectures.
- **Report Wizard**—Design a Panther report that you can use unchanged, or easily edit for added functionality and refinements.

## Application Building Blocks

The main building blocks of a Panther application are its screens, reports, and service components. The editor provides a graphical environment for building your application components and populating them with widgets.



Screens, being the part of the application that is presented to the end-user, has the greatest number of widget types. For example, a screen can have text widgets for displaying and entering data, push buttons for user actions, check boxes and radio buttons for user selections, and dynamic labels for graphics. These and other widget types are created in the editor.

Service components, which are used in three-tier applications, are created in the editor and then deployed on the application server.

Reports have two views in the editor: report layout, where the different areas of the report are defined, and report structure, which determines the order of the report areas and the processing to occur for each area.

## Properties Window

You can set properties for screens, reports, service components and widgets via the Properties window. The Properties window consists of a variety of features and methods for displaying and setting property values for the selected object in one convenient location. Setting properties allows you to control the look and behavior of your application components.

You can reference Panther objects and their properties through JPL, C or Java. If a property is accessible through the screen editor, its JPL mnemonic is usually a variant of the name used in the Properties window.

## Libraries

In Panther, all screens, reports, menus, service components, JPL modules, and JIFs are stored in libraries. The editor allows you to open and create libraries or use the following distributed libraries:

- Client library for screens, reports, menus, and JPL modules that comprise the client side, or user interface, of the application.
- Server library for service components and JPL modules that define the server side of the application (three-tier only).
- Common library for application-wide objects such as the JIF and configuration files (JetNet/Oracle Tuxedo only).

## Repositories

Panther's visual object repository provides a data dictionary-like mechanism for storing application components and importing the objects' definition to your application screens. Repositories can have multiple entries; each repository entry is composed of widgets. Your repository entries, both the entries and the widgets on them, have properties that define their visual attributes and behavior as well as database definitions if the objects were derived from a database. You can use and reuse these objects wherever they are needed in your application.

The visual object repository provides:

- A mechanism to assure consistency and control among all components of your application.
- A single reference for data elements and templates used in an application.
- A facility for propagating database table and column information to your application.
- A single access method for propagating changes to widget and screen properties without having to individually edit each application screen.

You will learn to create and open repositories, open repository entries, create and save repository entries and control the properties that will continue to maintain an inheritance relationship.

**Note:** You can create and open remote repositories only during JetNet/Oracle Tuxedo development.

## Importer

You can import your database tables and views to a repository. The importing process creates a repository entry for each database table or view and a widget for each database column. These repository entries provide the basis for your screen and report development.

## **Transaction Manager**

When you copy imported database objects from the repository to your application screens with the Panther editor, you automatically provide the transaction manager with all the information it needs to build transactions that access and update the database. Within the authoring environment, you can execute transaction manager commands that access databases without needing to write any SQL (Structured Query Language) commands.



# E Editor Basics

The chapter describes basic editor tasks needed to build your application components:

- Starting the Editor ([page E-2](#))
- Opening and Creating Application Components ([page E-3](#))
- Opening and Creating a Library ([page E-7](#))
- Opening a Middleware Session—JetNet and Oracle Tuxedo applications ([page E-9](#))
- Opening a Database Connection ([page E-12](#))
- Saving an Application Component ([page E-16](#))
- Creating Widgets ([page E-13](#))
- Setting Properties ([page E-14](#))
- Creating and Opening a Repository ([page E-19](#))
- Saving Application Components to a Repository ([page E-23](#))
- Controlling Inheritance ([page E-29](#))

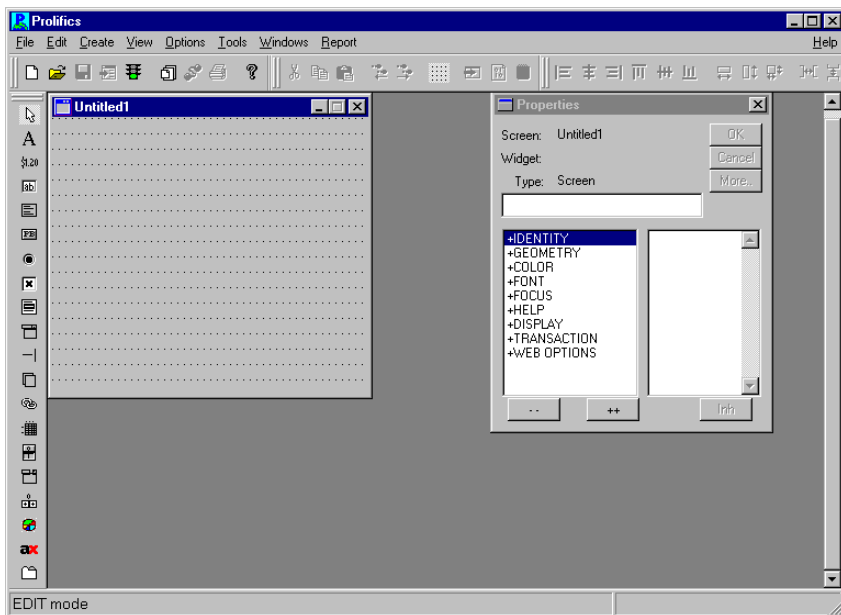
# Starting the Editor

## How to Start the Editor

Do either of the following:

- Click on the Panther icon.
- Type `prodev` (or the full pathname of the Panther executable).

The editor workspace is displayed. It is in this space that you begin creating the interface to your application.



**Figure 5-1** The editor workspace consists of a menu, toolbars, a new screen and an associated Properties window, and can display optional secondary windows.

For a description of each workspace component, refer to Chapter 2, “Editor Workspace.”

## Editor Startup options

`prodev` supports the following startup options:

- `-a`—start in application mode.
- `-debug`—start in debug mode. See Chapter 39, “Using the Debugger,” in *Application Development Guide* for details on using the Panther debugger.
- `-e`—start in Editor mode. This is the default when the `-a` and `-debug` options are not specified.
- `-ini`—in Windows, specifies the name of the Windows initialization file. See Chapter 3, “Windows Initialization File,” in *Configuration Guide* for more information.
- `-ro`—open the screens initially specified in read-only mode (new in Panther 5.50).

The name of the screens, reports and/or components to be opened initially can also be specified.

---

# Opening and Creating Application Components

---

Your Panther application components are created in the editor and then stored in a library or a repository.

## Creating a New Application Component

A new, untitled screen is open by default when you start up the editor. Now you can begin populating the screen with widgets, or create other types of application components.

### How to Create a New Application Component, Library or Repository

1. From File→New, choose
  - Screen—Defines the user interface.
  - Report—Stores the report definition for report output.
  - Service Component—Defines services for three-tier applications.
  - Repository—Stores imported database objects and application-wide templates.
  - Repository Entry—Allows for an inheritance link when you use the repository entry and its objects to create your application components.
  - Library—Stores application objects, files, and graphics.
2. For a new screen or report, choose whether you want to use the wizard. If you use the wizard, Panther guides you through the creation process using imported database tables in a repository (a repository must be open).

## Opening Application Components

To open an application component, the library that it lives in must be open. If you want to ensure that a library opens when you start up an editing session, specify the library's path in the `SMFLIBS` variable. However, if the library is not specified in `SMFLIBS`, you can open the library via the File menu.

Once the library is open, there are two ways to access and open application components:

- Via the File menu where you can open them individually.
- Via the library table of contents (available by choosing View→Library TOC).



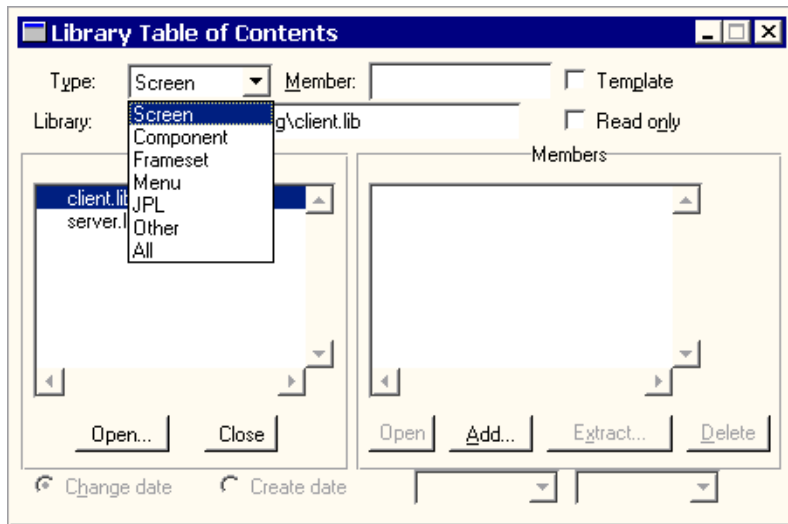
## Viewing the Library Table of Contents

While you are working in the editor, you can:

- Display the contents of a library.
- Open one or more application components in succession.
- Open and edit a JPL module in a library (you can open and simultaneously edit multiple JPL modules).

### How to Open the Library Table of Contents

1. Choose View→Library TOC. A Library Table of Contents window opens.



**Figure 5-2** You can access the library table of contents to open all types of application components.

2. From the Type option menu, select the type of application component:
  - Screen
  - Report
  - Component
  - Menu

- JPL
  - Other (including graphics files)
  - All
3. You can do any of the following from the Library TOC window:
    - Open—Choose to open local libraries.
    - Remote—Choose to open remote libraries (JetNet/Oracle Tuxedo only).
    - Close—Choose to close the selected library (all except JetNet/Oracle Tuxedo).
    - Open—Choose to open the selected screen.
    - Add—Choose to add a disk file to a library.
    - Extract—Extracts a screen to a disk file on the system. The source screen remains in the library.
    - Delete—Choose to delete a screen from a library.
    - To only show Members based on their creation or last change date, choose the Change date or Create date radio button and then choose the starting and ending date in the two option menus to the right of the radio buttons.
  4. To open an application component, select the desired name and choose Open or double-click on it.

The object opens in the workspace. The name is displayed in the title bar in the format: `appComponent@libraryName`.
  5. Continue to open application components, as needed, by repeating step 2.
  6. To close the Library TOC window, choose Close from its system menu or choose View→Library TOC to toggle the display off.

## How to Remove an Application Component

From the Library TOC, select the object and choose Delete. The following prompt is displayed:

```
Remove [appComponent] from library?
```

- Yes—Deletes the named object from the library. If the object is currently open in the workspace, it closes when you respond to the prompt. You cannot recover an object that has been deleted from a library.
- No—Cancels the deletion.

---

## Opening and Creating a Library

---

If you are responsible for creating libraries for the team, you must also ensure that the libraries open by default when you start up Panther. To open libraries, specify the library's path in the `SMFLIBS` variable. However, if you are a member of the team and you want to create and open a local library, then you can do so from the File menu option in the menu bar.

### How to Open a Library

You can have more than one library open at a time.

1. Choose File→Open→Library. The Open Library dialog box opens.
2. Select the library (the default filter extension is `.lib`) that you want to open and choose OK.

**Note:** You can also open a library from the library table of contents window.

### How to Open a Remote Library

(JetNet/Oracle Tuxedo only) You must have opened a middleware session and have a file access server (`devserv`) running which enables access to libraries located on remote machines. For further information on the server types, refer to “Server Executables” on page 2-2 in *JetNet/Oracle Tuxedo Guide*.

1. Choose File→Open→Remote Library. The Open Remote Library dialog box opens.
2. Select a server from the list of servers. The libraries with their respective paths are displayed for the selected server.
3. Select the library that you want to open and choose OK.

**Note:** You can also open a remote library from the library table of contents window.

## Creating a Library

If you are responsible for creating libraries for the team, then it is recommended that you create:

- Client library for screens, menus, graphics, and JPL modules that comprise the client side, or user interface, of the application (two- and three-tier development).
- Server library for screens and JPL modules that define the server side of the application (three-tier only).
- Common library for application-wide objects such as the JIF and configuration files (JetNet/Oracle Tuxedo only).

These libraries are provided by copying the distributed libraries, namely `client.lib`, `server.lib`, and `common.lib`, from `SMBASE/config` to your application development directory. The distributed libraries have copies of files you can modify which are needed for development.

## How to Create a Library

1. Choose File→New→Library. The New Library dialog box opens.
2. Specify the name of a library (with a `*.lib` extension) and choose OK.

## How to Create a New Remote Library

(JetNet/Oracle Tuxedo only) You must be connected to the middleware and have a file access server (`devserv`) running which provides access to remote machines. For further information on the server types, refer to “Server Executables” on page 2-2 in *JetNet/Oracle Tuxedo Guide*.

1. Choose File→New→Remote Library. The New Remote Library dialog box opens.
2. Specify the name of a server in the Server field.
3. Specify the name of a library in the Library field (with a `*.lib` extension) and choose OK.

---

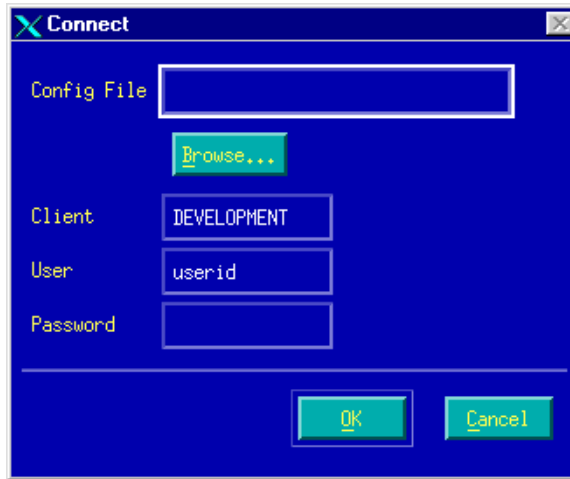
## Opening a Middleware Session

---

In JetNet/Oracle Tuxedo applications, you open a middleware session to connect to a middleware and access the libraries and repositories on the application server. The middleware controls processes and communication between the application's clients and servers. To test an application screen that uses services, you must have a valid middleware connection.

## How to Open a Middleware Session

1. Choose File→Open→Middleware Session. The Connect dialog box appears.

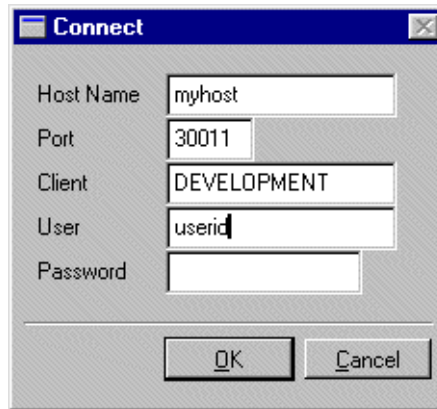


**Figure 5-3** Connect to the middleware for a client on a server machine, also called a local or native client.

For a client on a server machine, the Connect dialog box has the following fields and specifications:

- **Config File**—By default, displays the value of `SMRBCONFIG` variable. Otherwise, use the Browse button to select your configuration file.
- **Client**—Specify client type by name (default is `DEVELOPMENT`). The client name can be no greater than 31 characters.
- **User**—Specify user account (login) name. The user name can be no greater than 31 characters.
- **Password**—Specify application password. The password can be no greater than 8 characters.

For a client not on a server machine, the Connect dialog box has the following fields and specifications:



**Figure 5-4** Connect to the middleware for a client not on a server machine, also called a remote client.

For a client not on a server machine, the Connect dialog box has the following fields and specifications:

- Host Name—By default, displays the value of the `SMRBHOST` variable. This variable provides the network address of the machines to which the client will connect.
  - Port—By default, displays the value of the `SMRBPORT` variable. This variable provides the port numbers associated with the machines to which the client will connect.
  - Client—Specify client type by name (default is `DEVELOPMENT`).
  - User—Specify user account (login) name.
  - Password—Specify application password.
2. Enter the information required in the appropriate fields depending on whether you are a client on a server machine or a PC client.

If the application requires level-two authentication, then you must enter the application password in the Password field.

**Note:** The level of authentication is decided at design time. Thus, a user should be aware of the decided authentication level.

---

# Opening a Database Connection

---

While you are building your application components, you need a direct connection to the database server if you are:

- Developing a two-tier application—to import data definitions into a repository and to test your screens using “real” data.
- Developing a three-tier application—to import from the database to a repository.

In JetNet and Oracle Tuxedo applications, you can also connect to the database to test service components. (Refer to “Service Components” [on page 38-5](#) in *Application Development Guide* for instructions.)

**Note:** During JetNet/Oracle Tuxedo development, making a direct connection to the database server does not affect a database connection made via server initialization.

## How to Connect to the Database within the Editor

1. Choose File→Open→Database or in test mode, choose Database→Connection. The Choose Engine dialog opens.
2. Select the desired engine and enter a connection name, if the default name is not appropriate.
3. For some engines, a Connect to Database dialog opens where you can enter connection options. The options vary according to the selected engine. For JDB, an engine-specific Open File dialog is displayed from which you can select the desired JDB database file.

For information on connecting programmatically, refer to “Programmatically Connecting to the Database” [on page 8-3](#) in *Application Development Guide*.



## How to Close a Database Connection within the Editor

Choose File→Close→Database or in test mode, choose Database→Disconnect. Both options close direct connections on a specified engine. They do not close connections that have been established via the application server.

---

## Creating Widgets

---

When you create widgets on a client screen, users see and use most of the widgets to enter data and display output. You can control how client screens and widgets respond to input and output by setting their properties.

## How to Create a Widget

1. Select the widget type from either of the following:
  - Create toolbar—Select the desired icon. (Refer to Figure 2-3 [on page 2-15](#) for a description of the icons.)
  - Create menu—Select a widget type from the list.
2. Move the mouse pointer to the position on your screen where you want the widget to be placed.
3. Do either of the following:
  - Click once to create a default-size widget, or
  - Drag a box to the desired size for the widget. Release the mouse button.
4. (Recommended) Assign a name to the widget (in the Properties window, under the Identity heading in the Name property).
5. If you make a mistake, you can undo or redo the last several (10 by default) actions by choosing Edit→Undo or Redo (or the Undo/Redo button on the toolbar).

In addition, you can populate application components with objects by simply dragging them from one to another, described in Chapter 9, “Manipulating Widgets.”

For information on the Panther keyboard interface (using Panther without a mouse), refer to Appendix A, “Keyboard Interface.”

---

## Setting Properties

---

The Properties window lists the property settings for selected objects; an object can be a screen, a widget, or several widgets. By default, an application component is selected when no widgets on the screen are selected. Do either of the following to deselect widgets:

- Click on an empty area of the application component to deselect all widgets.
- Deselect widgets in the [Widget List](#) by clicking on one widget to deselect all others, and then press the spacebar to toggle the selected widget off.

You must select a widget or widgets to set properties at the widget-level. A selected widget is displayed with a changed border—on GUI platforms small, black boxes, or grab handles are displayed; in character mode, the widget is surrounded by square brackets ([ ]). A selected grid widget member is displayed in reverse video from unselected members.

Only certain properties can be set for a given widget type. As a result, the information that is displayed in the Properties window is dependent on what object or objects you have selected. If you use the screen wizard to create your client screens, reports, and service components, the wizard automatically sets certain properties for each.

## Using the Properties Window

The Properties window is open by default. To toggle the display of the window, select or deselect Properties Window from the View menu, or, whenever the Properties window is not visible, simply press Enter.

## How to Set Properties

1. Display the Properties window by pressing Enter or by choosing View→Properties Window.
2. Select the application object:
  - To select a widget—Click on the desired widget or widgets (refer to “Selecting a Widget” on page 9-1 for methods of selecting widgets).
  - To select an application component—Click on an empty area, ensuring that no other objects are selected.

The Widget field in the Properties window displays the object's name (or field number). The Type field indicates what type of object is currently selected. If more than one object is selected, three question marks (???) are displayed.

3. Display the properties in the Properties window. Expand a single heading by clicking on it, or expand the entire list by choosing the ++ button.
4. Select the desired property from the Properties listing. Depending on the selected property, you can do one of the following:
  - Some properties automatically invoke a dialog box where you can enter code or text, depending on the property.
    - Enter the information. This might be JPL code, comments about the screen, screen control strings, or text and static data that will display at runtime.
    - Choose OK to save your entry or Cancel to exit without saving changes.
  - In the Setting field, type a value (which might be a character, number, or filename) or select one from a drop-down list of options.
  - If there are predefined values, enter the initial character (for example, type *y* for Yes).
  - Choose More to display one of the following, depending on the property:
    - Zoom dialog box—An expanded Setting field where you can enter multiple lines or longer strings of data.
    - Select Library Member dialog box—Select a file in a specific library. In addition, this dialog box provides the capability to browse libraries with the Open button and to add system files to a library with the Add button.
    - Platform-specific dialog box—Select platform-specific fonts or colors.

- Click on the current value that displays to the right of the property to cycle through each available setting.
5. If you want to restore a setting, choose Cancel.
  6. To commit the property setting, do any of the following:
    - Click on another property or anywhere in the property list.
    - Choose OK.
    - Click anywhere on your application component in the editor workspace.
  7. If you mistakenly changed a property, you can choose Edit→Undo to restore the value. You can reverse the last several actions (up to the number specified).

---

## **Saving Your Work**

---

Screens, reports and service components in a Panther application must be saved to their appropriate libraries. For example, store client screens in the client library and service components in the server library. However, during development you might also store some application components in a repository to provide consistency and control throughout the entire application.

### **Saving an Application Component**

The appropriate libraries or repository must be open in order to save a client screen, service component, report or repository entry. If the development team is sharing libraries, it is the team leader's responsibility to create libraries for the team and to ensure that all the libraries open when you start up an editing session. However, if you are an independent developer working on a screen that you want to save in a local library, you must create the local library and open it via the file menu before saving the screen.

## How to Save an Application Component

1. Choose File→Save. If it is a member of a library or a repository, it is saved to its source. If the application component is untitled, the Save dialog box opens.
2. If more than one library is open, select the one in which the object should be saved.

If you do not have the appropriate library open, choose the Open button or the Remote button. From the Open Library dialog box (or the Open Remote Library dialog box in JetNet and Oracle Tuxedo executables), select the library that you want to open and choose OK.

3. Specify a name in the Member field.

**Note:** In three-tier applications, it is recommended that you adopt a naming convention that identifies client screens with their corresponding service components or vice versa. Table 5-1 on page E-18 lists suggested file extensions.

4. Choose OK.

The name, in the form `appComponent@libraryName`, displays in the title bar.

## How to Check for Overlapping Widgets

HTML does not support overlapping widgets. As a result, Web browsers can render widgets in positions that differ from the appearance in the editor workspace. To ensure that widgets adhere to your original design and do not overlap:

1. Set Check Overlap on Screen Save on the Options menu.

When this option is active, any save operation (Save, Save As, Save All) forces the editor to check for overlapping widgets before actually saving the screen or screens affected by the save request.

2. Screens with overlapping widgets are brought to the top of the stack, the overlapping widgets are selected, and the following message is displayed:

Screen *screenName* contains overlapping widgets. Save anyway?

To check for overlapping widgets *before* saving a screen:

- Choose Edit→Find→Overlapping Widgets. All overlapping widgets are selected; all others are deselected.

## Naming Conventions

Although the editor does not enforce naming conventions, the following standards are recommended for naming screens, libraries, and repositories:

- Names can contain the following characters: alphabetic, numeric, underscore (`_`), dollar sign (`$`), and period.
- Do not use ASCII punctuation characters, spaces, tabs, new lines (`\n`), operation symbols (`+`, `-`, `*`, `|`, etc.), or the `@` sign (reserved for Panther usage).
- Non-ASCII characters (such as accented or Cyrillic letters) and symbols can be used.

In addition, your operating system and those you support may impose their own restrictions for filenames.

```
"hd:Desktop Folder:myfile"
```

Panther does have default extensions for each file type which are used primarily as filter specifiers on Save and Save As dialog boxes as well as Open dialog boxes under some GUI platforms.

Table 5-1 contains general recommendations that you can apply which can help you distinguish one file type from another. You can also adopt your own naming conventions.

**Table 5-1 Recommended filename extensions**

<b>File type</b>	<b>Default filter extension</b>	<b>Recommendation</b>
Client screen	*.scr	Use a convention that identifies client screens with their corresponding service components.
Service component	*.scr	Use a convention that identifies service components with their corresponding client screens.
Library	*.lib	Use a convention that distinguishes libraries from repositories and other file types.

**Table 5-1 Recommended filename extensions**

<b>File type</b>	<b>Default filter extension</b>	<b>Recommendation</b>
JPL	*.jpl	Libraries can contain any type of file, therefore, use a convention that distinguishes one file type from another.
Menu	*.menu	Libraries can contain any type of file, therefore, use a convention that distinguishes one file type from another.
Styles	*.sty	Libraries can contain any type of file, therefore, use a convention that distinguishes one file type from another.
Repository	*.dic	Use a convention that distinguishes repositories from libraries.

## Creating and Opening a Repository

A repository has one or more entries, which originally could have been screens, reports, or service components. You can have multiple repositories; however, since you can have only one repository open at a time, it is recommended that you create one repository per application. If application development is a team effort, then a member of the team is responsible for creating and maintaining a repository to provide consistency across the application. If you are an independent developer working on an application, then you need to create your own repository.

Copying objects from a repository to your application components allows for an inheritance relationship between the copied objects. The wizards also store and use information in the open repository to build new screens and reports.

## Creating a Repository

You must first create a repository in order to import or store screens as repository entries.

### How to Create a Repository

1. Choose File→New→Repository. The New Repository dialog box opens.
2. Specify the name of a repository in the Selection (or File Name) field.  
**Note:** Refer to Table 5-1 on page E-18 for recommendations on file names.
3. Choose OK.

### How to Create a New Remote Repository

(JetNet/Oracle Tuxedo only) You must be connected to the middleware, and you must have a file access server (`devserv`) running which provides access to create repositories on remote machines. For further information on the server types, refer to “Server Executables” on page 2-2 in *JetNet/Oracle Tuxedo Guide*.

1. Choose File→New→Remote Repository. The New Remote Repository dialog box opens.
2. Specify the name of a server in the Server field.
3. Specify the name of a repository in the Repository field (with a `*.dic` extension) and choose OK.

Creating a new repository automatically closes any repository that is currently open. You can now save a screen as a repository entry in this newly created repository by choosing Save As→Repository Entry.

## Opening Repository Entries

While you are working in the editor, you can open a repository to gain access to its contents, so that you can copy repository objects to your application screens. Opening a repository automatically closes any repository that is already open.



**Note:** The repository named `data.dic` (or the repository identified in the setup variable `SMDICNAME`) is automatically opened on starting your editor session.

## How to Open a Repository and Its Repository Entries

1. If the desired repository is already open, skip to step 2. Otherwise do either of the following:
  - Open a repository. Choose File→Open→Repository. The Open Repository dialog box opens.
  - Open a remote repository (JetNet/Oracle Tuxedo only). Choose File→Open→Remote Repository. From the Open Remote Repository dialog box, select a server from the list of servers. The repositories with their respective paths display for the selected server.

2. Select the repository (the default filter extension is `*.dic`) that contains the entry you want. Only one repository can be open at a time.

The default repository name is `data.dic` and it opens by default for an editing session.

3. Choose File→Open→Repository Entry. The Open Repository Entry dialog box opens.
4. Select the desired repository entry and choose OK.

The repository entry opens. The name followed by the `[Repository]` indicator is displayed in the title bar. The format is `repositoryEntry@[Repository]`. Database tables that have been imported to a repository are named without a file extension, by default.

If the entry is being edited by someone else, Panther posts a warning message. If you choose Yes to steal the reservation, the entry is opened in your name and has read and write privileges. Since two people now have the entry open with write privileges, one of you could lose your changes. If you choose No, the entry opens with read-only privileges. If the repository and the entry are under source code management, you are presented with different options (refer to “Maintaining Libraries Under Source Control” on page 10-4 in *Application Development Guide* for more information).

**Note:** If you choose to open a repository entry as a Template, a copy of the entry is created in the repository, but it does not by default inherit from the source repository entry.

To create a repository entry, refer to [page E-24](#), “How to Save a Repository Entry.”

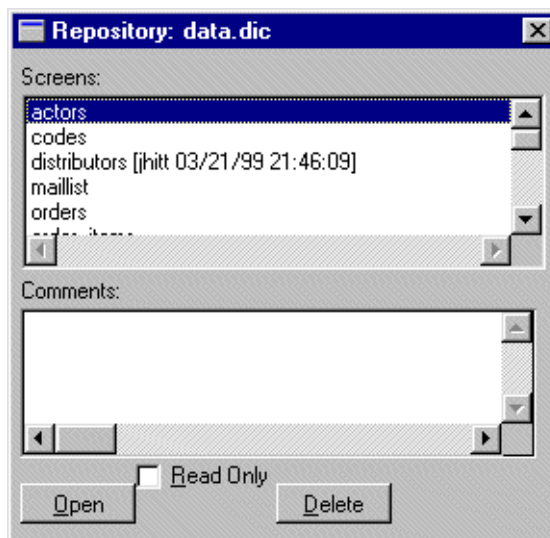
## Viewing the Repository Table of Contents

There are two ways to access and open repository entries:

- Via the File menu where you can open a single screen.
- Via the View menu where you can view the repository's table of contents and open multiple repository screens.

### How to Open the Repository Table of Contents

1. Choose View→Repository TOC. A Repository window opens and lists all the entries that are stored in the open repository.



**Figure 5-5** The Repository TOC lists all the repository entries and indicates whether the entry is currently in use.

2. (Optional) If you want to edit the repository entry, deselect the Read-only check box on the Repository TOC window.
3. Select a repository entry and choose Open.  
The entry opens in the workspace. The name of the repository entry is displayed in the title bar in the format `entry_name@[Repository]`.
4. Continue to open repository entries, as needed, by repeating step 3.
5. To close the Repository TOC window, choose Close from its system menu or choose View→Repository TOC to toggle the display off.

## How to Remove a Repository Entry

From the Repository TOC, select the repository entry and choose Delete. The following prompt is displayed:

Remove [repository\_entry] from repository?

- Yes—Deletes the named object from the repository. If the object is currently open in the workspace, it closes when you respond to the prompt. You cannot recover an object that has been deleted from a repository.
- No—Cancels the deletion.

---

# Saving Application Components to a Repository

---

A repository can store one or more entries. You can:

- Create a repository entry.
- Save a screen, report, or service component as a repository entry.

Regardless of how you create a repository entry, you use the entry and its objects to create your application components—dragging objects from the repository to the application component. This process creates an inheritance link between the parent, or original object, and its child, or copy. So, if you make changes to an object in the repository, those changes are propagated to all your application components derived from it.

**Note:** Widgets stored in repository entries **must** be named to enable inheritance to child widgets. Specify a name in the Name property under Identity in the Properties window.

## How to Save a Repository Entry

1. To open a repository if one is not already open, do one of the following:

**Note:** A repository, `data.dic` or the one named in the `SMDICNAME` setup variable, is opened by default on starting your editor session.

- Create a new repository. Choose File→New→Repository. The New Repository dialog box opens. Enter a name for the repository (refer to Table 5-1 on page E-18 for recommended naming conventions).
  - Create a new remote repository. Choose File→New→Remote Repository. At the New Remote Repository dialog box, specify the name of a repository with a `*.dic` extension) and the name of the appropriate server.
  - Open an existing repository. Choose File→Open→Repository. Select the desired repository from the list (the default filter extension is `*.dic`).
  - Open an existing remote repository. Choose File→Open→Remote Repository. From the Open Remote Repository dialog box, select the desired repository from the appropriate server.
2. Choose File→New→Repository Entry. The New Repository Entry dialog box opens.
  3. Enter a name for the repository entry.

**Note:** Use a naming convention that distinguishes entries you create from those created as a result of importing database tables (which by default have no extension).

A new entry opens with the given name in its title bar, in the format `entry_name@[Repository]`.

4. Add widgets to the entry, assign property settings, etc.
5. Assign a name to each widget in a repository entry—under Identity, in the Name property in the Properties window.
6. Choose File→Save to save the entry in the open repository.

## Populating a Repository with Database Objects

You need to create or open either a repository or a remote repository before importing database objects to it. However, you must have a direct connection to the database to successfully import database objects to a repository. You can browse the contents of your database with Panther's database browser and then import tables and their column definitions to Panther. Through this process, screens are automatically created and stored in the open repository.

By default, when Panther displays the Import Database Objects window, only the database tables are displayed. However, the Options buttons displays a dialog box where you can select to display database views or synonyms.

You can import up to 1000 tables. The maximum number of columns per table is 255.

## How to Import Database Tables from Your Database

1. If the desired repository is not open, create or open it now:
  - To create a new repository, choose File→New→Repository or Remote Repository. At the New Repository dialog box or the New Remote Repository dialog box, enter a name (refer to Table 5-1 on page E-18 for recommended naming conventions). Choose OK.
  - To open a repository, choose File→Open→Repository or Remote Repository. At the Open Repository dialog box or the Open Remote dialog box, selected the desired repository. Choose OK.
2. Choose File→Open→Database. The Choose Engine dialog box opens.
3. Select an engine from the Engine option menu of initialized database engines on your system.

4. Accept or enter a connection name and choose OK. A database Open dialog box appears.
5. Specify the name of the database in the Selection (or File Name) field, enter any other options needed by your database engines, and choose OK.
6. Choose Tools→Import Database Objects (or the Import Objects button on the toolbar). The Import Database Objects dialog box opens.

All tables in the current database are listed, up to a maximum of 1000. (If there are more than 1000 tables in the database, use the Filter field to control the display of table names.)

7. (Optional) If your database supports views or synonyms, the Options button will be active. You can select it to:
  - Choose any combination of tables, views, or synonyms.
  - Choose to name the table views without the user name.
8. Select the desired database table or tables. As you select individual tables, the column names, column data types, and lengths defined in each are displayed under Column Descriptions.

To select a table, click on the item in the Tables list. To select more than one table, do one of the following:

- Shift+click to select contiguous tables; Ctrl+click to select noncontiguous tables.
  - Choose Select All to import all tables.
  - Enter a full or partial string in the Filter field and press Enter to filter the display of tables. Then select the desired tables from the filtered list or choose Select All.
9. Choose Import. The status line informs you of each table being processed.

A repository screen is created for each table and is designated to inherit from the database as do all the widgets that make up the screen (the Inherit From property for the repository screen is set to @DATABASE).

If a repository screen already exists for a database table, the screen is updated (refer to “Re-importing Database Tables” [on page E-28](#) for details on re-importing).
  10. Choose Close to return to the editor.

To display the contents of the repository, choose View→Repository TOC.

## What You Get When You Import Tables

The import process:

- Creates a repository screen (by default has no extension) for each database table that you import. Each screen contains one text widget and one label widget for each column; each widget is appropriately named, as opposed to numbered. Label widget names start with the letter L followed by the name of the associated text widget.
- Sets other data definitions as properties in Panther—such as data length.
- Creates non-displayable table view widgets for each database table. The table view provides Panther's transaction manager with the information it needs to access your data. A table view must be present on your application screens if you want the transaction manager to provide automatic database access. Table view properties include the list of all widgets (columns) in the table view, and identify the primary key associated with the table.
- Creates links for each foreign key that is defined for the database table. The links provide Panther's transaction manager with the defined relationships between two table views: the parent table view and the child table view.
- Sets the Inherit From property to @DATABASE for each repository screen and each text widget. This ensures that future database changes can be passed from database to repository screen, and ultimately to your application, without having to open and edit each repository screen and application screen.
- Automatically checks out existing repository entries under source code management if you are re-importing database tables. Remember to check these screens back into source code management to make them available to other developers. (Refer to “Maintaining Libraries Under Source Control” [on page 10-4](#) in *Application Development Guide* for more information.)

## Updating Database Views

Panther sets the Updatable property to Yes for all database objects—tables, views, and synonyms. However, be aware that a database view or synonym might not contain the primary key columns that are needed for database updates. If the primary key columns are not members of one of Panther's table views, the transaction manager gives an error.

## Re-importing Database Tables

Re-import database tables:

- When your database has changed, or
- To ensure that your application screens reflect the latest database.

## How to Re-import Database Tables

Follow the same directions as described for importing tables the first time. If you have more than one database opened, Panther prompts you to choose which database to import from.

When you re-import, the process proceeds as follows:

1. A repository screen is created if one does not exist for a database table.
2. If a repository screen exists for a database table, but it does not have an Inherit From property setting of @DATABASE, Panther prompts you to choose whether to overwrite the repository entry or leave it unchanged.
3. Existing repository screens are updated if their Inherit From property is set to @DATABASE.
4. Database columns are compared to the widgets in the repository screen by checking the following property settings for each widget:
  - The Inherit From property under Identity is set to @DATABASE.
  - The database column name is identified in the Column property under Database.



- For the table view: The Table property (under Database) identifies the database table name and the Column property identifies the database column.
5. The inherited properties are updated for the existing widgets.
  6. Primary and Foreign keys are verified. Any changes in the primary key columns for the database are reflected in the Primary Keys property for the table view. Any new foreign keys result in a new link widget on the repository screen.
  7. A new widget is added for every table column that has no corresponding widget in the repository screen.

---

## Controlling Inheritance

---

When you copy a repository object to an application screen, all property values are automatically copied and most possess an inheritance relationship between the parent and child objects. You can control which properties will continue to maintain an inheritance relationship. You can:

- Selectively turn inheritance on or off for specific properties.
- Restore the inheritance relationship for all properties associated with a selected screen or widget.
- Prevent propagation of changes.

### How to Turn On Inheritance for a Property

To selectively turn inheritance on or restore it for a specific property:

1. Select the widget or the application screen.
2. Select the property (unhighlighted) in the Properties window.
3. Choose the Inh (Inherit) push button on the Properties window.

The value for the selected property is now highlighted and set—provided that Options→Inherit is on—to match the property for the parent object specified in the Inherit From property for the selected object.

## How to Turn Off Inheritance for a Property

To change a property for a widget that inherits its property value:

1. Select the child widget in your screen.
2. Select the property that requires a custom setting.
3. Enter the new value. The following message is displayed:  
`The value is inherited! Remove inheritance?`  
**Note:** You can turn these inheritance reminders off by choosing OptionsFile→Set Inherit Warnings.
4. Choose Yes to remove inheritance for the selected property. The new value is confirmed. If you choose No to retain inheritance, the original setting is restored.

## Preventing Propagation of Changes

To control the propagation of changes from parent objects to their children, use any of the following methods:

- Turn inheritance off for selected properties by selecting the property and toggling the Inh (Inherit) push button in the Properties window. Highlighted properties will still inherit changes.
- Remove the Inherit From property value for the selected application object. Leaving a blank value removes the inheritance relationship for *all* properties; it does not remove the values. All future changes made to the repository entry are not propagated to the selected widget or screen.
- Choose Options File→Inherit to temporarily turn inheritance off. This option is useful if you are in the process of changing or designing repository entries and don't want changes to propagate at design time. When you toggle the Inherit option back on, inheritance is restored and all changes are propagated

appropriately to open screens and child widgets that inherit from the open repository.

## How to Restore Inheritance of All Property Values

1. Select the widget or your application screen (to inherit screen properties).
2. Under Identity, enter the source of inheritance in the Inherit From property.
  - For widget inheritance: Enter the name of the parent widget and its repository entry in the format: `repository_entry!widget_name`.
  - For screen inheritance: Enter the name of the repository entry.
3. Choose OK. The following message is displayed:  
`Do you want to set all the properties inherited?`
4. Indicate whether all properties should be inherited:
  - Choose Yes to inherit all property values.
  - Choose No to retain current property settings. You can then set inheritance on specific properties. (For details, refer to [page E-29](#), “How to Turn On Inheritance for a Property.”)

## Finding the Source of Inheritance

You can find a parent widget or screen (the one that inherits from) as well as find child widgets or screens (the ones that inherit). To find the parent, or source, of inheritance:

### How to Find Parent Widgets or Screens

1. Select the widget in your application screen or the screen itself.
2. Choose Edit→Find→Parent.

**Note:** If you have selected an object that does not have an Inherit From property value, the Parent menu option is disabled.

The parent repository screen, or the one containing the parent widget, is opened (if it currently is not). The parent widget, the source of inheritance, is selected.

## How to Find Child Widgets or Screens

To find the children of a parent object:

1. Select the widget in a repository screen or the repository screen.
2. Choose Edit→Find→Children. The Find Children dialog box opens.  
**Note:** If you have selected an object that is not in a repository, the Children menu option is disabled.
3. Select one or more of the search criteria:
  - Repository, Open Libraries, and Screens—Searches the currently open repository, open libraries, and all screens that are open in the workspace.
  - Libraries in Current Directory—Searches the libraries in your current directory.
  - Libraries along `SMPATH`—Searches the `PATH` indicated in the `SMPATH` variable (located in `SMVARS` or environment setup).
4. Specify which file extensions should be included in the search filter.
5. Choose OK.  
If matches are found, the Child List window opens.
6. Select the screen you wish to view/edit and choose Open. You can continue to select and open screens from the Child List window.
7. To close the Child List window, choose Close from its system menu.

# 2 Editor Workspace

The editor is a graphical environment for building and manipulating screens, reports, and service components that make up your Panther application. All of the features that make GUIs (Graphical User Interfaces) easy to use are available in the editor—regardless of your environment. With Panther, you can create GUI applications that are virtually platform-independent as well as ensure that your application complies with the GUI standards of your choice.

While using the editor, you can:

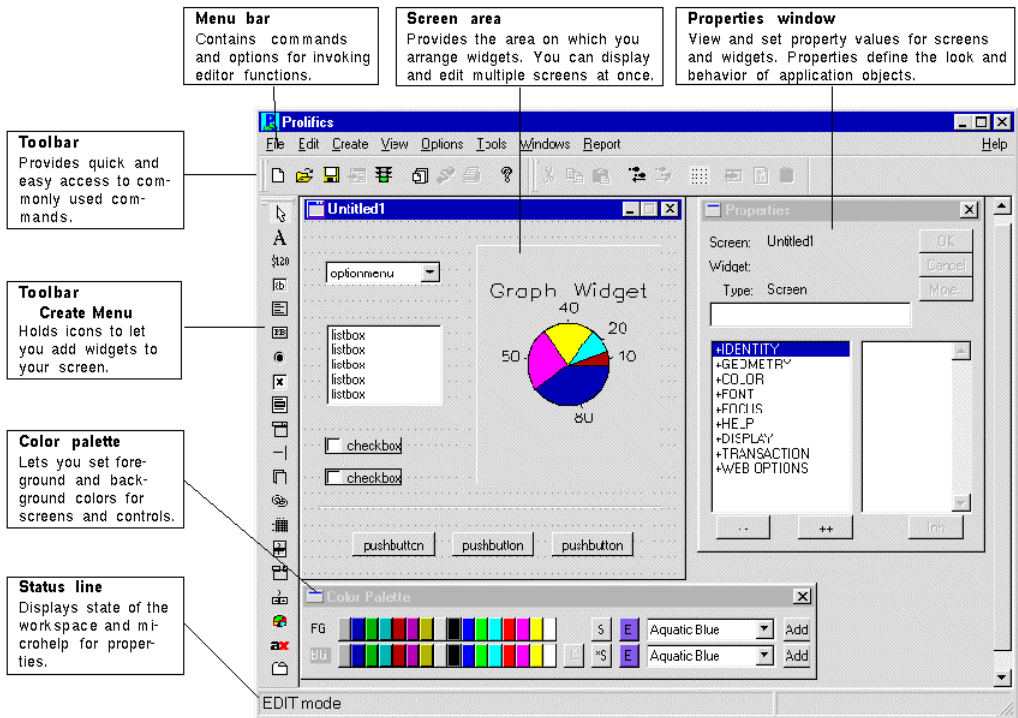
- Control what additional tools and windows are displayed during your editing session.
- Access the standard editing operations in the editor.
- Define how the editor workspace should look.

This chapter addresses the basic aspects of the editor workspace:

- Menu Bar ([page 2-4](#))
- Toolbars ([page 2-14](#))
- Types of Widgets ([page 2-19](#))
- Properties Window ([page 2-27](#))
- Property Categories ([page 2-30](#))

# Editor Workspace

When you start the editor, the editor workspace is displayed consisting of a menu, toolbars, a new screen and an associated Properties window. It can display optional secondary windows for screens, reports or service components. It is in this space that you begin creating the interface to your application.



**Figure 2-1 The Editor Workspace**

The following components are displayed:

**Menu bar**

Contains available editor menu options.

**Toolbars**

With your mouse, you can quickly access commonly used commands. When you point to a button, a tooltip is displayed so that you can become familiar with the associated commands.

In Windows applications, toolbars can be moved and docked to various parts of the MDI frame. (For more information on this feature, refer to [page 2-15](#), “Dockable Toolbars.”)

**Note:** Under Motif, the tooltip does not display when the toolbar button is inactive.

**Edit area**

Displays an untitled application component. Here is where you place widgets and decorations that make up your application component. You can display multiple application components at once.

**Properties window**

Displays the property settings for the selected screen, service component, report or widget. Through the Properties window you define the initial look and feel of the object, such as colors, location, and size.

**Status line**

Displays the current state of the workspace. By default, the editor is in Edit mode. Other possible statuses are Test mode and Application mode. The status line also provides microhelp for Toolbar icons, menu commands, properties, and dialog box options.

**Note:** Under Motif, status line text does not display for inactive menu items.

## **Editor Windows**

While working in the editor you can access other editor tools, such as the Toolbars and Color palette, to facilitate application development. You can ensure that these same windows open automatically every time you open the editor by saving your preferences; choose File→Save Pref.

You can access the other editor windows by selecting the appropriate command from the View menu.

## Menu Bar

The editor menu bar provides you with easy access to the operations and commands that help you develop applications. This section provides a brief overview of each menu option and its contents.

### File Menu

The File menu commands displayed on the toolbar allow you to create new application components, open and save existing components, and access test mode where you can test a screen or report.

The commands associated with the File menu include:

#### New

Creates a screen, service component, report, frameset, JPL file, repository entry, local (or remote) repository, local (or remote) library, or Java file.

#### Open

Opens a screen, report, service component, frameset, JPL module, repository entry, repository, library, database connection, middleware session, or Java file. Starting with Panther 5.30, there is also a Recent Files submenu that allows you to reopen files that were recently opened.

#### Close

Closes the current screen, report, frameset, or service component, an open repository, or a specific library. You can also choose to close a connection to the middleware or to a database.

#### Remove

Removes an entry from the open repository, or remove a specific library member (for example, a screen, a JPL module, a menu script file, a pixmap file).

#### Save

Saves the selected application component with its current name. If the application component is untitled, the Save As dialog box opens and you can save it to a specific library.



**Save As**

Lets you name and save the selected application component to a specific library, to the open repository, or to save as an ASCII text file.

**Save All**

Lets you save all currently open screens with their current names to their respective libraries or repositories. If any of the screens are untitled, the Save As dialog box opens.

**Save Pref**

Saves workspace preferences, such as Property window position and Toolbars display, as well as other settings indicated in the Options menu, such as showing the grid. Preferences saved here are overwritten if the [Save Preferences on Exit](#) option (under the Options menu) is turned on.

**Revert**

Reverts to the last saved version of the current screen. Unsaved changes made to the screen are not saved. An untitled screen reverts to an empty screen.

**Source Mgmt**

Lets you access your installed source code management tool (support for SCCS and PVCS). The option is only available if an open library or repository is under source code management. Choose to check-in or check-out a repository entry, screen or JPL procedure from a repository or library. You can also choose to cancel the check-out request. (For more details, refer to “Maintaining Libraries Under Source Control” [on page 10-4](#) in *Application Development Guide*.)

**Test Mode**

Lets you test the look and feel of the current screen, report, or frameset. You can test application components without saving the changes you made to them. Unsaved changes you have made to other open application components in the workspace are not lost when you go into Test mode. For further information on testing screens, refer to Chapter 38, “Testing Application Components,” in *Application Development Guide*.

**Exit or Quit**

Ends your session in the editor. You are prompted to save any application components that you have created or changed but have not saved.

## Edit Menu

Edit menu commands let you manipulate and design your application screens—providing alignment, centering, spacing, and sizing options as well as standard editing operations, like cut and paste.

The commands associated with the Edit menu include:

### Undo

Undoes last several actions you performed in the workspace. The last action taken is displayed.

### Redo

Re-performs the last action done in the workspace. The last action taken is displayed.

### Cut, Copy, Delete, Paste, and Select All

Allows standard editing commands anywhere within the editor workspace.

### Insert From Library

Allows you to read in existing JPL script, JavaScript, or VBScript from another open library into an open JPL text window.

### Read File

Allows you to read in an existing JPL script, JavaScript, or VBScript from disk into an open JPL text window.

### External Editor

Invokes the editor you specify (via the `SMEDITOR` variable) to edit JPL files, JavaScript, or VBScript in a text window.

### Group

Provides the method for creating and identifying widgets as a selection group, as a table view, and as synchronized scrolling arrays. You can also select group members or entire groups and do group updates.

### Align

Aligns a selected group of widgets (left, center, right, top, middle, or bottom) with respect to each other.

### Size

Resizes a selected group of widgets by equalizing their heights, their widths, or both at once.

**Space**

Equalizes spaces (horizontally or vertically) between widgets within a selected group.

**Center**

Orients a selected group of widgets to the center (vertically and horizontally) on the screen.

**Grid Align**

Orients a group of selected widgets in relation to the editor's grid lines. You can orient vertically, horizontally, or both.

**Find**

Allows you to find a selected widget's source of inheritance or, conversely, find all the widgets that are the children of a repository object (namely widgets that inherit from a parent object). This also allows you the option of finding overlapping widgets on a given screen (for further information, see [page E-17](#), “How to Check for Overlapping Widgets”).

**Arrange**

For reports, you can modify the view of layouts by collapsing some or all of the layout areas, or by changing their relative positions. For more information, refer to “How to Collapse and Expand Layout Areas” [on page 2-3](#) in *Reports*.

## Create Menu

Create menu commands provide options for creating all widgets: labels, text, selection-type widgets, and grid widgets as well as graphical elements, like lines and boxes.

The drop-down list initially lists dual-deployment widgets (such as dynamic label and single line text widget) which are available on all platforms. At the bottom of the list is a submenu of Extended Widgets which cannot be deployed on every platform.

Extended Widgets currently include:

- Graph widget—Available for Motif, Windows and Web.
- ActiveX control containers—Available for Web and Windows.
- Tab decks and tab cards—Available for Windows and Motif.

- Toggle button, combo box and scale widgets—Available for Motif and Windows.

**Note:** Even through toggle buttons, combo boxes and scale widgets do not have HTML equivalents in Web applications, they are converted to appropriate widget types.

## View Menu

View menu commands allow you to specify which of the editor's windows are displayed. The menu settings can be toggled on and off to display or hide specified windows. The View menu contains the following options:

### Properties window

Lists the property settings for a selected screen or widget. You use this window to set properties for all objects on a screen and for the screen as a whole. You can also open the Properties window by double clicking on the workspace screen or on a widget or by pressing Enter. (See [page E-14](#), “Using the Properties Window.”)

### Repository TOC (table of contents)

Lists all screens in the open repository. You can open repository screens successively from the TOC window; you can have multiple repository screens open in your workspace. You can also delete repository entries from the TOC window. (See [page E-22](#), “Viewing the Repository Table of Contents.”)

### Library TOC (table of contents)

Lists all open libraries and their contents. You can open screens successively from the TOC window; and you can have multiple screens open in your workspace. You can open additional libraries (local and remote), add files to a library, delete files from a library, open JPL modules in a library, and extract files from a library to a system file in the TOC window. (See [page E-5](#), “Viewing the Library Table of Contents.”)

### Widget List

Lists all widgets in the active screen and provides an alternative method for selecting widgets, including those that are hidden or placed behind another widget. If you have multiple screens open in the workspace, as you move from screen to screen, the Widget List changes to reflect the active screen selection. (See [page 9-3](#), “Using the Widget List.”)

**Color palette**

Shortcut tool for applying foreground and background colors to screens and widgets. Displays 16 Panther-specific, GUI-specific and GUI-independent colors.

**DB Interactions**

Graphical map, or representation, of the relationships between parent and child table views and their links in the active screen. Provides a method for selecting these database widgets and accessing their properties. (See “Setting Link Properties” [on page 31-8](#) in *Application Development Guide*.)

**Component Interface**

(COM/MTS/EJB only) Provides a way of setting methods (type, name, and parameter) and properties (name, type and read-only options) for service components in three-tier applications. (See [page 7-4](#), “Defining the Component Interface.”)

**Report Structure**

The report structure window displays a diagram of the report structure. Each report element—format, data groups, detail data, and the report itself—has a corresponding node in the structure. (See Chapter 3, “Introducing Report Structure,” in *Reports*.)

## Options Menu

Options menu commands let you to specify how your workspace in the editor looks and acts. You can save these preferences as well as the editor's screen positions by choosing File→Save Pref. Or, you can save your current state and preferences by choosing Save Preferences on Exit from the Options menu. Refer to the following section for details on setting preferences.

**Grid**

Toggles the display of the screen grid (or choose the Grid button on the toolbar). By default, the grid is displayed and provides a guide for placing and aligning widgets. The grid must be displayed for the grid alignment command to function.

**Multiple Select Mode**

(Character mode only) Set by default to allow you to select noncontiguous widgets by simply clicking on each desired widget; then you can carry out commands or set properties on all selected widgets at one time. This option functions like Shift+click or Ctrl+click in windowing environments.

**Inherit**

Toggles inheritance on and off. When Inherit is off, changes made in repository screens are not propagated to open application screens that are set to inherit from a repository. Turning Inherit on immediately restores inheritance and propagates any changes in repository screens to open application screens.

**Set Inherit Warnings**

Allows you to turn inheritance warning messages off or on.

**Multiple Create Mode**

(Character mode only) Causes the Create toolbar and Create menu to stay in create mode rather than automatically returning to the Select icon or Select mode option, respectively.

**Save Preferences on Exit**

This overwrites any preferences you previously saved (including those saved using File→Save Pref).

**Snap to Grid**

(GUI only) Causes any new or repositioned widgets to position to its closest horizontal and vertical anchor points on the grid. If the grid is not displayed, this preference has no effect.

**Auto Release on Screen Close**

When the option is deselected, you are prompted whenever you choose to close a library screen or repository screen that you have reserved from a library or repository. Select this option to turn the prompt off, in which case the reservation is automatically released when you close either type of screen.

**Drag Screen Size**

When this option is enabled, dragging a corner of the report window resizes the report, not just the viewport. When it is disabled, you can resize a report and leave the report's Geometry unchanged.

**Direct to External Editor**

If you select this option, it opens any JPL module directly in your local editor and allows you to edit it; however, to confirm the edits, the local editor reads the JPL from the editor to the JPL text window.

**Configure Toolbars**

This option allows you to choose which toolbar menus are visible in the editor:

- Create—Options to create widgets. (Figure 2-3)
- Main—Options to create and save screens. (Figure 2-5)
- Edit—Options to cut, copy and paste objects and undo edits. (Figure 2-6)
- Positioning—Options to align widgets. (Figure 2-7)
- Report Layout—Options to create, edit, and save reports. (Figure 2-8)
- Report Structure—Options to create and edit the report structure. (Figure 2-9)

#### Editor Tooltips

(GUI only) Determines whether to show tooltips on editor screens.

#### Editor Tabs

This configures the number of tab stops for the whichever external editor you are using.

#### Undo Levels

Allows you to specify a number of undo levels (0-99) that you can perform while working in the editor.

#### Service Alias

(JetNet/TUXEDO only) This configures the user alias to use when testing services. For more information, refer to “Using Service Aliases to Test Services” on page 5-8 in *JetNet/ Oracle Tuxedo Guide*.

#### Check Overlap on Screen Save

Forces the editor to check for overlapping widgets before actually saving the screen or screens affected by the save request. Used for Web screens since HTML does not support overlapping widgets. (See page E-17, “How to Check for Overlapping Widgets.”)

#### Reload Java Classes

Reloads the Java classes when entering test mode or exiting the editor.

#### Enable Debugger

Allows you to debug the events associated from the current screen when you enter test mode. Refer to Chapter 39, “Using the Debugger,” in *Application Development Guide* for details on setting preferences and using the Panther debugger. You can also enable the debugger directly from application or test modes, and set preferences there as well.

## Tools Menu

The Tools menu provides the ability to import database objects, compile Java code, and access the Menubar, Styles, and JIF editors.

### Import Database Objects

Provides access to the Database Browser where you can import database objects to the currently open repository.

### Compile Java

Compiles the specified Java class. You can override the default command string using [SMJAVACOMPILE](#). You can create new Java files from File→New→Java or edit Java files from File→Open→Java. For more information on using Java, refer to Chapter 21, “Java Event Handlers and Objects,” in *Application Development Guide*.

### Generate TM SQL

For the current screen, writes the SQL statements that the transaction manager generates for the screen to a file. One use for these SQL statements is to construct stored procedures in the database.

### Generate Component

For a service component, creates the type library for COM or MTS components or the Java files for Enterprise JavaBeans. You can also prepare EJBs for deployment.

### Menubar Editor

Invokes the Menubar editor which creates and edits menus that you can attach to individual screens via the editor or define for your application as a whole. For more information, refer to Chapter 25, “Menu Bar Editor.”

### Styles Editor

Invokes the Styles editor which enhances the transaction manager's control of the appearance and behavior of widgets on your database application screens. For more information, refer to Chapter 23, “Styles Editor.”

### JIF Editor

(JetNet/Tuxedo only) Invokes the JIF editor where you create and edit JIF files containing information about services, service groups, and reliable queues in a JetNet or Tuxedo application. For more information, refer to Chapter 24, “JIF Editor.”



#### IBM Visual Age for Java

(WebSphere only) Invokes IBM Visual Age for Java if it is available to your computer. You can specify the path for the program by setting [SMIBMVJAVA](#) in your setup file.

#### IBM WebSphere Administrative Console

(WebSphere only) Invokes the IBM WebSphere Administrative Console if it is available to your computer. You can specify the path for the program by setting [SMIBMWSADMIN](#) in your setup file. Windows will check the registry settings for IBM WebSphere if the setup variable is not set.

## Windows Menu

Windows menu commands provide access to all open windows in your editor workspace. At the bottom of the Windows menu is a list of open windows/screens from which you can select and bring focus.

#### Next User Screen

Moves focus from one user-created screen to another.

#### Next System Screen

Moves focus from one Panther system screen (non user-created) to another.

#### Cascade

(Windows environment only) Moves and resizes all open screens and windows into a layered arrangement within the MDI frame.

#### Tile

(Windows environment only) Moves and equally resizes all open screens and windows within the MDI frame.

#### Arrange Icons

(Windows environment only) This neatly arranges the icons for minimized windows.

#### Raise All

(Motif environment only) Raises all Panther screens to the top of the display, and layers them according to the window stack.

## Report Menu

Report menu commands allow the generation of reports from within the editor. In three-tier environments, these reports can be generated only on the client via a local database connection; server-based reports must be processed using the command line or by JPL commands.

### Preview Report

Opens a Run Report dialog box providing options to run the main or a subreport, pass arguments to the report, direct output to the internal viewer, a printer or a file, and access the print and page setup dialog boxes.

### Print Setup

Opens the Print Setup dialog box, allowing the specification of printer options.

### Page Setup

Opens a dialog box allowing basic page setup options including paper size, orientation and margins.

### Output Viewer

Opens the viewer window so that report output files that were saved as metafiles may be viewed.

### Show Property Links

Shows the property links associated with a selected widget in the layout window or node in the report window.

## Toolbars

GUI versions of the editor contain toolbars with the most popular menu items.



**Figure 2-2** The Toolbar in the Motif Version of the Editor

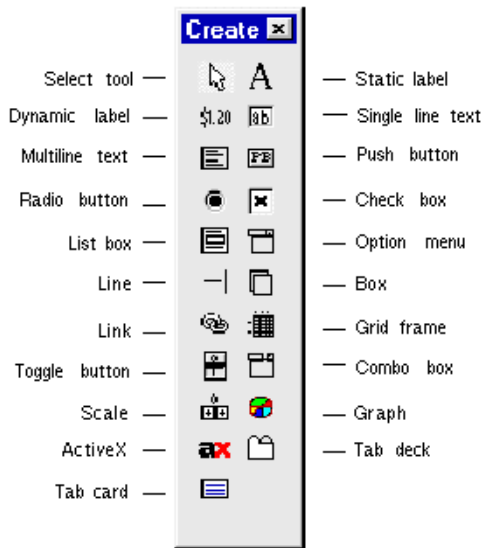
## Dockable Toolbars

In Windows executables, you can expose or hide any of six toolbars. Panther runs within an MDI frame. If a toolbar is visible in the editor, it can be separated, moved and docked anywhere within the frame.

By choosing Options→Configure Toolbars, you can select the check boxes of the toolbars you want to display in the editor.

### Create Toolbar

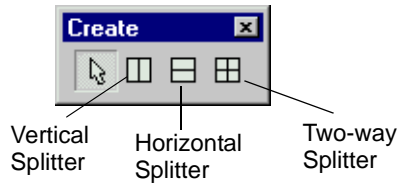
Shortcut tool for creating widgets. Select a widget from the Create toolbar and click on the screen at the desired location to place the widget. To temporarily activate multiple create mode, press Ctrl+click on a widget icon in the Toolbar. Cancel the mode by pressing Enter or by selecting a different widget icon in the toolbar.



**Figure 2-3 Create Toolbar**

### Create Toolbar for Framesets

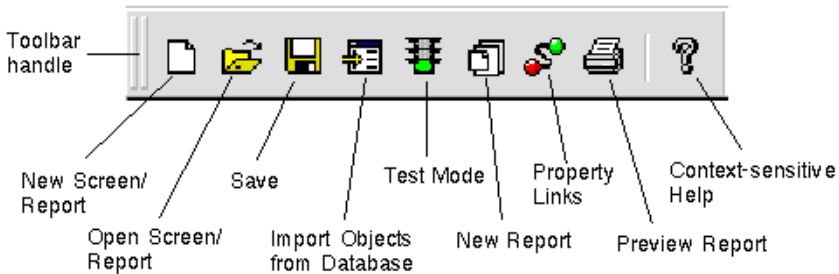
In Windows, shortcut tool for creating splitters in framesets. Select a splitter from the Create toolbar and click on the desired location in the frameset.



**Figure 2-4 Create Toolbar for Framesets**

**Main Toolbar**

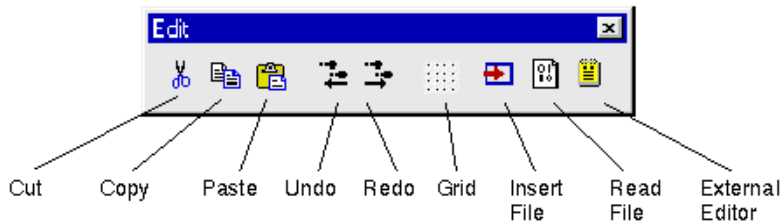
Contains some of the more commonly-used functions from the File and Reports menus.



**Figure 2-5 Main Toolbar**

**Edit Toolbar**

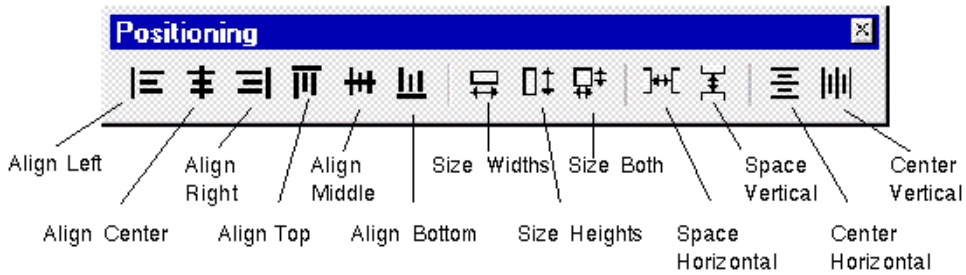
Allows easy access to some of the more commonly-used functions from the Edit menubar.



**Figure 2-6 Edit Toolbar**

**Positioning Toolbar**

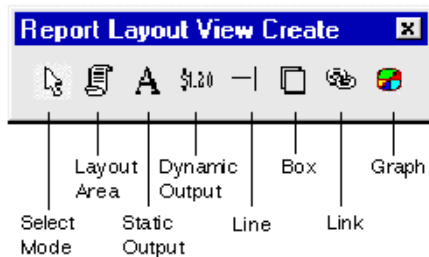
Contains the alignment and positioning functions from the Edit menubar.



**Figure 2-7 Positioning Toolbar**

**Report Layout Toolbar**

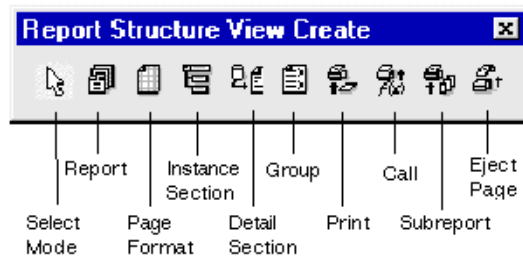
Contains the shortcuts for report widget creation. This toolbar is only active when a report has focus.



**Figure 2-8 Report Layout Toolbar**

**Report Structure Toolbar**

Allows you to create additional nodes in the report structure. This toolbar is only active when the Report Structure window has focus.



**Figure 2-9 Report Structure Toolbar**

## How to Use Dockable Toolbars

- To display a toolbar, choose Options→Configure Toolbars. Select the check boxes of any or all of the toolbars you want to display.
- To hide a toolbar, you can either:
  - Choose Options→Configure Toolbars. Unselect the check boxes for the toolbar(s) you do not want to display.
  - If the toolbar is floating in the MDI frame, you can close the toolbar by using the standard Windows method of clicking on the Close check box (in the upper right-hand corner).
- To automatically dock a toolbar to its former location on the MDI frame, double click on the toolbar's title bar (the top of the toolbar).
- To float a docked toolbar, double click on the toolbar's handle.
- The toolbar can be stretched, shrunk and elongated. When it is docked, it always forms a single, elongated bar.
- Multiple, dockable toolbars are only available in the editor only. At runtime, there is only one toolbar; however, it is dockable.

Chapter 25, "Menu Bar Editor," contains information about making toolbars.

---

# Types of Widgets

---

Widgets are self-contained in that they hold all the information they require. Each widget type has a unique set of properties which, in fact, define the widget—the way it looks and the way it behaves. There are widgets that function as labels and only display text; others are used to accept data, while others only output data. The editor provides, via the Properties window, a way for you to establish the initial behavior and look of your application screens and their content. However, for the most part, you can change these defaults at runtime with Panther-provided library functions or through the property API.

You can create widgets via the Create toolbar or the Create menu. On the Create menu, the widgets that are only available on certain platforms are accessible from Create→Extended Widgets.

Each widget type is briefly described here. References to more detailed information for implementing these widgets are also indicated.

## Display-Type Widgets

Display-type widgets are those which display information, either textually or graphically. For the most part, the user cannot bring focus to these widget types, and therefore, cannot modify the content directly.

Static label **A**

Provides a means for displaying protected, static text to the user. The content of this widget cannot be accessed or modified by the user at runtime. It is for display purposes only. You define the content and the look of the static label at design time. You cannot change its content programmatically at runtime. Refer to Chapter 13, “Display Widgets,” for more information on static labels.

Availability: All platforms

### Dynamic label



Displays output data that is received either from the system, from a database, or programmatically; at runtime, a dynamic label is protected from data entry and, therefore, cannot receive focus. A dynamic label can be textual or pictorial. In addition, you can assign double-click events to dynamic labels. Refer to Chapter 13, “Display Widgets,” for more information on dynamic labels.

Availability: All platforms

### Graph



Presentation graphics include the ability to display data as a pie chart, bar/line chart, or XY-plot, in a two- or three-dimensional perspective. The graph can be populated with data derived from user entry in other fields on the screen, calculated data in other fields, and database-derived widgets. Refer to Chapter 13, “Display Widgets,” for details on creating and using graphs in your application.

Availability: All platforms

## Data Entry Widgets

Text widgets are used for accepting and displaying data at runtime. You can define the initial content of these widgets by setting the appropriate property. Refer to Chapter 14, “Data Entry Widgets,” for information on creating data entry fields.

### Single line text



Provides a single line area to accept or display data. You can display multiple elements by specifying that a single line text widget be an array. Each element in the array appears as a separate occurrence. An array can be arranged either horizontally or vertically.

Availability: All platforms

### Multiline text




Accepts or displays multiple lines of word-wrapped text. The text array is enclosed within a single border. At runtime, this type of array can be scrolled




by dragging the mouse, or by using the keyboard or scroll indicators. The multiline text widget can also be a non-word wrapped array.

Availability: All platforms

Combo box 

Combines a text widget and list box, providing the user with the choice of typing data in the text portion or selecting an item from the drop-down list (nonscrollable in Motif) portion.

Availability: Motif, Windows


Grid widget 

A two-dimensional array that can scroll both vertically and horizontally. The grid widget acts as a container for accepting or displaying tabular data, such as columns and rows from database tables, or spreadsheet-type data. Refer to Chapter 15, “Grid Widgets,” for more information on defining grid properties.

Availability: All platforms


## Input Widgets

These widgets, although very different from one another, in general, provide the user with a means of inputting data without having to type data.

Option menu 

Provides a drop-down (non-scrolling in Motif) list of suggested choices from which the user can choose. Typically, option menus are used when the range of possible values for the field is limited to a relatively small number of valid values and only one selection can be made. The option menu is useful when a user might not know the possible values, and also when it's preferable that the user select a value rather than type an entry character-by-character. An option menu can be populated with data from an external source, like a database, or even from another screen. Refer to Chapter 20, “Selection Widgets,” for more information about option menus and other selection widgets.

Availability: All platforms

Scale 

Provides a way for the user to choose a value within a specified range. This widget is a combination of a scroll bar, or slider, which runs between a minimum and maximum value, and a dynamic label, which displays above a scroll bar or to the left of a horizontal scale. The label changes to reflect the current value. Scale widget values are digits only. Refer to “Using Scales” on [page 14-19](#) for more information.

Availability: All platforms

## Command Widget

A command widget, when activated, executes an action or event. It requires no user input other than choosing the widget. Although other widget types can carry out a command, most GUI applications make frequent use of push buttons to perform an action.

Push button 

Provides a way for users to quickly and easily execute an action or procedure with a single mouse click. You can display a text label or a picture (under the GUI platforms) on push buttons. You can specify the button's initial state—whether it can be pushed (active) or not (inactive)—and identify it as the screen's default or cancel button. When activated, push buttons appear to be pressed. Drop shadows are used in the GUIs, while a color change indicates activation in character mode. Refer to Chapter 19, “Push Button Widgets,” for information.

Availability: All platforms

## Selection Widgets

Selection widgets provide choices to the user. By default, some of these widgets allow for a single selection, while others let the user make several selections. These widgets are protected from data entry and clearing. Selection widgets of like kind can be grouped (for more information on grouping widgets, refer to [page 20-10](#), “Grouping Selection Widgets”). Grouped selection widgets have their own set of properties. Selection widgets can also function alone. Check boxes and toggle buttons are examples. In addition, those widgets that have labels can optionally display pictures (for GUI environments) on them instead of the label text.

For more information on providing user options in your application, refer to Chapter 20, “Selection Widgets.”

Radio button 


Provides the user with a set of choices. Radio buttons work as a group, allowing a user to select one option in the group, which in turn clears the other buttons in the group. By default, only one selection can be made. Refer to “Using Radio Buttons” [on page 20-7](#) for more information.

Availability: All platforms

Check box 


Provides a method for setting or selecting preferences. Check boxes can be grouped so a user can select one or more options, while an ungrouped check box might be used to turn a condition or preference on or off. At runtime, when a check box item is selected, a check mark or `x` appears next to the item (in Windows) and a three-dimensional, shaded box changes color (in Motif). Refer to “Using Check Boxes” [on page 20-2](#) for more information.

Availability: All platforms

Toggle button 

Provides an off/on type of widget. When a toggle button is pushed, the button remains on or in the “in” position. When the button is pushed again, it appears to be off or in the “out” position, so the behavior or action is toggled off. With a group of toggle buttons, the user can, by default, choose any number of toggle buttons in the group. Refer to “Using Toggle Buttons” [on page 20-9](#) for more information.

Availability: Motif, Windows

List box 

Provides a scrollable column of data. List boxes can be one of two types: action or selection. By default, a list box is the latter, and the user can make any number of selections. As an action type, the list box can function like a menu, where each item has an associated event or action. List boxes can be populated with data from an external source, like a database, or by supplying data at design time via the Properties window. Refer to “Using List Boxes” [on page 20-3](#) for more information on list boxes.

Availability: All platforms

## Graphics Widgets

There are two graphic elements that you can draw to enhance the look and organization of your screens. You can specify size and color for these types of widgets. They are not field widgets and therefore cannot receive or display data; they are decorative. Refer to Chapter 21, “Graphics Widgets,” for more information about using lines and boxes on your screens.

Line 

Allows you to draw vertical or horizontal lines by dragging the mouse in the desired direction and length. You can also assign colors and styles.

Availability: All platforms

Box 

Allows you to draw a box, extending it to the desired coordinates on your screen. You can specify the color of the lines (foreground) that make up the box as well as the area within the lines (background color). You can assign a label to the box and use the box as a border around one or more widgets.

Availability: All platforms

## Database Widget

There is a single database widget that is specific to database definitions. It is used to define relationships between parent and child table views on your screen. This widget type is used by the transaction manager in order to carry out database transactions. Refer to Chapter 22, “Table Views and Links,” for more information on linking table views.

Link 

A hidden-at-runtime widget that is automatically created when you import database tables. The link is used by the transaction manager to describe relationships between parent and child table views. In the editor, the link appears like a label widget, with the name of the parent and child table views in the format `parent+child`.

Availability: All platforms

## COM Widget

The ActiveX container allows you to specify which ActiveX control to make a part of your application.

ActiveX 

ActiveX is a widget which utilizes part of Microsoft's COM (Component Object Model) technologies. An ActiveX control is a component program object that can be created and reused by many applications in the same computer or in a distributed network. For more information, refer to Chapter 18, “ActiveX Controls.”

Availability: Windows, Web

## Tab Controls

The tab control allows widgets to be grouped onto individual display cards inside a tab deck. Refer to Chapter 16, “Tab Controls,” for more information.

Tab Deck 

The Tab Deck is a special kind of container which has a group of cards associated with it. Together with the Tab Card (see below), this widget is used to create a tab control.

Availability: Windows, Motif

Tab Card 

The Tab Card consists of a tab and a set of widgets associated with that card.

Availability: Windows, Motif

## Hidden Widgets

There are two additional widget types—both of which are permanently hidden at runtime and design time. However, like other Panther widgets, these too have their own set of properties, which you can access by selecting the hidden widget from the [Widget List](#) or from a variety of access points (discussed later in this guide).

### Table view

A group of related widgets, usually associated with and named for a single database table. Although the table view is not visible, its group members are. The table view must be present on your application screen if you want to use the transaction manager to provide automatic database access.

A table view widget is automatically created on each repository entry when you import database tables into a repository. When you copy a database-derived widget from a repository entry, Panther creates a table view on the destination screen, provided the copied widget belonged to a table view in the repository entry. Refer to Chapter 22, “Table Views and Links,” for more information on table views and their properties.

Availability: All platforms

### Group

A hidden group widget is automatically created when you group selection widgets, when you define widgets to scroll together, and when you include widgets in a grid widget. Through the group widget's properties you can define the behavior of the widgets as a group, or single entity, as opposed to defining the behavior for each individual widget. Refer to “Grouping Selection Widgets” [on page 20-10](#) for more information on creating groups.

Availability: All platforms

## Report Widgets

When a report layout window has focus, the Create menu displays the widget types available for reports. When the report structure window has focus, the Create menu lists the report nodes that can be added to the report structure.

The following widget types are only available for report layouts:

### Layout area

The layout area widget defines a specific part of a layout window. That part, once named, can be accessed by a structure node as part of a report. It can contain, among other things, data, headings, header and footer information, subtotals and totals.

### Static output

Provides a means to display data that is specified in the editor. It is similar to the static label in screens.

### Dynamic output

Receives data at runtime from another widget or from a database. This widget is similar to the dynamic label in screens.

## Frameset Widgets

### Vertical splitter

Divides the frameset into two or columns, and is represented in the frameset screen in the editor by one or more vertical mullions.

Availability: Windows (Panther 4.5+)

### Horizontal splitter

Divides the frameset into two or more rows, and will be represented in the editor by one or more horizontal mullions.

Availability: Windows (Panther 4.5+)

### Two-way splitter

Divides the frameset into four panes, and will be represented in the editor by one vertical and one horizontal mullion.

Availability: Windows (Panther 4.5+)

---

# Properties Window

---

The editor provides, via the Properties window, a way for you to establish the initial behavior and look of your application components and their content.

You must select an object (screen, service component, report, widget) to set its properties. By default, the application component is selected when none of its widgets are selected. Do either of the following to deselect widgets:

- Click on an empty area of the application component to deselect all widgets.

- Deselect widgets in the [Widget List](#) by clicking on one widget to deselect all others, and then press the spacebar to toggle the selected widget off.

The Properties window consists of the following sections for displaying and setting property values for the selected object:

Screen

Displays the filename of the current application object (screen, report, service component). By default, the name is `Untitled` followed by a number.

Widget

Displays the name of the selected widget, as indicated in the Name property under the Identity heading. If the widget has not been named, its field number assignment is used. If more than one widget is selected, three question marks (???) are displayed.

Type

Displays the type of object currently selected. Types include: widget types (like Push Button, Static Label, Link), group types (like Group, Sync Group, Table View), Screen (when a screen is selected), Report File (when a report is selected) or Component (when a service component is selected). If more than one widget type is selected, three question marks (???) are displayed.

Setting field

Use this field to enter, select, or edit a setting for a property. Depending on the selected property, the setting field can function as an option menu, as a combo box where you can type a value or select from the drop-down list of possible values, or as a text field where you can type a value.

If a property has more than one predefined value:

- The setting field functions as an option menu or combo box; click on the indicator to display the list of valid choices. Select the desired setting by clicking on it, or, in the case of a combo box, you can type a value.
- You can enter just the first character in the setting field; for example, to specify a Color Type under the Color heading, you can type `b` for Basic colors, `e` for Extended colors, or `s` for Scheme.
- You can cycle through each value by clicking on the current property value in the Properties list. As you cycle through each value, the setting field is updated.



### Properties window

Contains an expandable list which displays all major headings and the properties for the selected object or objects. When the properties list is collapsed, each heading is preceded by a +, indicating that the heading can be expanded. Click on the heading, or press Enter, to view the properties under the selected heading. To expand the entire listing, choose the ++ push button. To collapse a heading preceded by -, click on the heading or choose the - - push button to collapse the entire list.

You can search for properties (when the list is expanded) by typing an initial character. The cursor moves to the next occurrence that matches.

The left column displays all applicable properties for a selected object; the right column displays the current property value setting. When more than one widget type is selected, only the properties that apply to all are displayed. In addition, three question marks (???) are displayed in the values column if you select multiple widgets that have differing property values.

Property values in the right column that are displayed in reverse video indicate that the setting is inherited from another source. For further information on inheritance, refer to [page E-29](#), “Controlling Inheritance.”

### More... button

Depending on the property you select, the More button displays one of the following:

- An expanded window where you can type a text string or multiple lines of text.
- A Selection window where you can search for and specify platform-specific colors and fonts.
- The Select Library Member dialog box where you can select a file from an open library, browse libraries with the Open button, and add disk files (such as bitmaps and icons) to a library with the Add button.

### Inh (Inherit) button

Toggle inheritance on or off for the selected property. This button is available only if the selected object has a value in the Inherit From property—that is, if the object is inheriting some or all of its properties from a repository. Refer to “Controlling Inheritance” [on page E-29](#) for more information.

### OK and Cancel buttons

Choose OK or click anywhere in the editor workspace to accept a setting. Choose Cancel to leave the value unchanged.

## Property Categories

Each object has default properties. Some of the properties apply to all objects; other properties are, in fact, what define and make objects unique. Setting properties through the editor allows you to initialize the content of your application components. After you complete your application interface, you can also control and alter the behavior and look of widgets and screens at runtime by using Panther's API, which includes C library functions, a property API, and Java interfaces.

## Widget Properties

In general, the property headings listed here for widgets are common to all widget types; the properties within each heading vary depending on the widget type. Some properties appear only if other property specifications have already been indicated. For example, if the Data Formatting property is set to Date/Time, a System Update property is available. These dependent properties are preceded by -> in the Properties window. In addition, a default setting for a property can vary depending on the widget type; however, if the property is listed in the Properties window, you can change it.

**Note:** If you have more than one widget type selected, only those properties that apply to all appear in the Properties window.

### Identity properties

Used to identify or change the identity of a selected widget, for example, name, type, source of inheritance (if applicable), label text, initialization state (for example, active vs. inactive), and memo text (for additional comments and programmatic use).

### Geometry properties

Define the size and location of a selected widget. Array, scroll and shift properties are included under Geometry.

### Color properties

Assign foreground and background colors. These can be Panther colors, platform-specific colors, or color aliases. In addition, for character mode, you can assign other attributes to the widget, like dim and blink.

### Font properties

Assign a font (this overrides a screen-wide or application-wide font specification). This can be a Prolifics font, a GUI-specific font, or an alias font name.

#### Focus properties

Used to define what happens when a widget gets and loses focus. These properties apply to any widget type that can receive focus. For example, entry and exit functions, tabbing order and behavior, and Java tags.

#### Help properties

Define the type of help or guidance a user might seek when the selected widget receives focus at runtime. For example, status line text, selection screen identification, and help screens.

#### Input properties

Used to define the characters, types of data, and restrictions on user input for a selected widget. These include things like defining the minimum and maximum values, initial values, keystroke filters, and whether the field is protected (protected from input or protected from clearing).

#### Validation properties

Used for writing, defining, and identifying the functions and events that occur during the widget's validation, that is, when the field receives or loses focus at runtime.

#### Format/Display properties

Control the way data is formatted and appears to the user. For example, things like date/time and numeric formats, textual justification (positioning), and whether the widget displays a picture instead of text.

#### Graph properties

Provide graph-specific information for designing a pie chart, bar graph, XY plot, or high/low graph.

#### Database properties

Provide database-specific information used by Panther to construct and generate SQL statements. These property values are initially derived from your database definitions: for example, the widget's column name in the database. In addition, you can define how the widget is used in a transaction; for example, if it should be used in the `SELECT` statement or in the `WHERE` clause of the SQL statement. Refer to Chapter 33, "Using Automated SQL Generation," in *Application Development Guide* for more information on Database properties.

#### Transaction properties

Provide database-specific information as required by the transaction manager to control display attributes and protections based on the kind of transaction

being performed. Refer to Chapter 22, “Table Views and Links,” for more information on Transaction properties.

Server View properties

Provide information about how the transaction manager should fetch data for the widgets in a table view.

Service properties

(JetNet/Tuxedo only) Provide the names of services to call for specific transaction manager events. You set service properties on table view and link widgets. Refer to “Service Properties” on page 22-7 for more information on Service properties.

Composition properties

Define properties for the widget when operating in a report.

Web Options properties

Define properties for the widget when operating in a web application.

XML properties

Define XML tags and attributes for importing and exporting widgets in XML.

## Screen Properties

The screen itself has its own set of properties. You gain access to these properties when a screen is selected (no widgets on the screen can be selected).

Identity properties

Identify the screen—like the screen's name, its source of inheritance, and whether the screen should function as a dialog box.

Geometry properties

Define the initial size and state of a screen as well as whether it is can be resized by a user at runtime.

Color properties

Assign a background color.

Font properties

Assign a font (this overrides the application-wide font specification). This can be a Prolifics font, a GUI-specific font, or an alias font name.

Focus properties

Control what happens when an application component has focus and what happens when focus changes: for example, the name of the menu that is associated with the screen, and the names of exit and entry functions.

Procedures properties (Service component)

Control the functions of a service component at runtime. These properties are similar to the those of the Focus properties of a screen, including entry and exit functions, and JPL procedures.

Help properties

Define the type of help or guidance a user might seek when a screen receives focus: for example, status line text and help screens.

Display properties

Control the screen's visual characteristics including things like its border style, color, decorations, and mouse pointer style.

Transaction properties

Define the root table and transaction model used by the transaction manager to carry out its commands at runtime.

Web Options properties

Define properties for using the screen in a web application.

XML properties

Define XML tags and attributes for importing and exporting screens in XML.





# Part II Wizards

Introducing the Wizards

Screen Wizard

Report Wizard





# 3 Introducing the Wizards

The Panther wizards are easy to use and guide you through the design process to create screens and reports that work with your database and the transaction manager. The editor provides access to the Panther wizards:

- **Screen Wizard**—Design screens for two-tier, three-tier, and Web architectures that you can use unchanged, or customize accordingly.

For Web applications, you can select the Web-friendly output option on the Format Selection dialog of the screen wizard.

For JetNet/Oracle Tuxedo applications, you can use the wizard to design service components as well as screens.

For further information on the screen wizard, refer to Chapter 4, “Screen Wizard.”

- **Report Wizard**—Design a Panther report that you can use unchanged, or easily edit for added functionality and refinements. For further information on the report wizard, refer to Chapter 5, “Report Wizard.”

This chapter describes:

- How the wizards interact with repositories and templates to generate screens and reports.
- Templates and how to customize wizard output via the templates.
- How to navigate within the wizards.

---

## How the Wizards Work

---

The screen wizard and report wizard take advantage of database objects in your repository to build screens and reports. Also, the first time you create a new screen or report with the wizards, Panther automatically creates the appropriate template screens in the open repository and related JPL and menus in the distributed libraries. Therefore, as a prerequisite to using the wizards, you must:

- Open or create a repository (refer to “Creating a Repository” [on page E-20](#) for details)—The repository must contain at least one table that inherits from @DATABASE (specified in the object's Inherit From property). This property setting is the result of importing the desired database objects.
- Import the desired database objects (tables, views, and/or synonyms) to the open repository (refer to “Populating a Repository with Database Objects” [on page E-25](#) for details) if the objects are not already imported.

The wizards guide you through a series of dialogs, prompting you for database specifications and design preferences required to create the appropriate presentation for your screens or reports. All the dialogs provide navigation buttons. The wizards set several properties for wizard-generated screens, reports, and the widgets on them. For further information on property specifications set by the wizards, refer to Appendix B, “Wizard Output.”

---

## Wizard Templates

---

In addition to using database objects in your repository, wizard-generated screens and reports use template screens as a source for graphical objects (such as push buttons and link widgets) not available from database entries in your repository. The JPL modules and menu/toolbar templates are passed on from the template screens to wizard-generated screens and reports via property settings.

Table 3-1 lists the templates created for Panther when you use the screen wizard for the first time.

**Table 3-1 Screen wizard-generated templates**

<b>Template</b>	<b>JetNet/TUXEDO</b>	<b>Two-tier/COM/MTS</b>
smwizard	*	*
smwizsrv	*	
smwizis	*	*
smwizweb	*	*
smwzmenu	*	*
smwizard.jpl	*	*
smwizsrv.jpl	*	

Table 3-2 lists the templates created for Panther when you use the report wizard for the first time.

**Table 3-2 Report wizard-generated templates**

<b>Template</b>	<b>JetNet/Oracle Tuxedo</b>	<b>Two-tier/COM/MTS</b>
smwizr	*	*
rwizard.jpl	*	*

## Description of the Templates

The following is a brief description of Panther templates (screens, menus, and JPL modules). After running the wizards, you can:

- Access the template screens in the open repository by choosing File→Open→Repository Entry.

- Access the template JPL modules by choosing File→Open→JPL and selecting the desired file from the appropriate library.

`smwizard`

It provides wizard-generated client screens with graphical objects and properties not available from database-derived entries in your repository. For example, it provides graphical push buttons, some widget types, and some screen property assignments (such as JPL procedures and menu name).

`smwizsrv`

It provides wizard-generated service components with graphical objects and properties not available from database-derived entries in your repository. For example, it provides some widget types and screen property assignments (such as JPL procedures).

`smwizis`

It provides screen wizard-generated selection screens and associated selection service components with graphical objects and properties not available from database-derived entries in your repository. For example, it provides push buttons and some screen property assignments (such as JPL procedures).

`smwizweb`

It provides wizard-generated screens specifically for use as Web applications with graphical objects and object properties not available from database-derived entries in your repository. For example, it provides graphical push buttons, some widget types, and some screen property assignments (such as JPL procedures).

`smwizrw`

It provides wizard-generated reports with graphical objects and object properties not available from database-derived entries in your repository. For example, it provides report areas, page headers and footers, and widgets such as lines, labels, graphs, and some overall report property assignments (such as JPL procedures).

`smwzmenu`

The screen wizard's template menu/toolbar `smwzmenu` is a binary file used by Panther at runtime and resides in the distributed client library. The ASCII script file from which it was made resides in the `config` directory under the name `smwzmenu.mnu`.

The menu is a good example of how to build menus with toolbars for your application. It demonstrates how to:

- Create multiple pull-downs.
- Place dividers exclusively on the menu or on the toolbar.
- Access standard GUI operations such as Edit and Windows.
- Set mnemonics on menu items.
- Use a menu that resides within another menu file (for example, the submenu `sm_dbi_menu`).
- Set status line and tooltip text.
- Attach pixmaps.

The pixmaps used for the screen wizard's template toolbar are delivered in `client.lib`, in the platform-specific format: BMP files (active versions only) for Windows, and XPM files for Motif.

#### JPL modules

The screen and report wizards set properties that make public the wizard's standard JPL module depending on the output generated. The standard JPL modules are as follows:

- `smwizard.jpl`—Resides in the distributed client library. It is used by the wizard-generated client screen.
- `smwizsrv.jpl`—Resides in the distributed server library (JetNet/Tuxedo only). It is used by the wizard-generated service component.
- `rwwizard.jpl`—Resides in the distributed client library or, in JetNet/Oracle Tuxedo only, in the common library. It is used by the wizard-generated reports.

## Customizing the Templates

The templates and the widgets in them have property settings that are inherited by your wizard-generated screen or report. You can alter or customize the properties of a template and any of its widgets to better suit your development requirements (For

example, you might want to define your own pixmaps on push buttons). Almost all properties for wizard-generated screens and reports are propagated from the template screens through inheritance. However, the following properties are not inherited:

- Title property (under Identity)
  - Note:** Do not alter this property on the template screens because the wizards and Panther technical support use this information for version verification.
- Comments property (under Identity)
- Height and Width properties (under Geometry)

When you run the wizards after editing the templates, the wizard output reflects the template modifications.

---

## Navigating in the Wizards

---

There are a variety of buttons available on screen and report wizard dialog boxes. They are:

### Next and Back buttons

Choose Next to indicate to the wizard that you are satisfied with the settings on the dialog box and to proceed to the next one.

Choose Back to traverse the wizard dialog boxes backwards. You can revisit any dialog box on which you have already made entries.

### Preview button

Choose Preview at any time to view the screen or report that is being built. You cannot save the screen or report at this point. Acknowledge the display to return to the wizard.

### Done button

Choose Done on the Style and Finish dialog box to indicate that you are finished. The wizard-built screen or report is displayed and you can choose to keep the screen or report as is or make other choices in the wizard session.

**Help button**

Choose **Help** to access specific information about the wizard dialog box you are currently viewing.





# 4 Screen Wizard

The screen wizard is a tool intended to help you design screens that work with the Panther transaction manager. It does not require that you have expertise with data types, widget types, primary or foreign keys, editor functions, or properties. It collects the basic design information by prompting you with a series of simple dialogs. The screen wizard is easy to use and quickly guides you through the design process for client screens and service components. You can also use the screen wizard to make a “first cut” of a complex screen.

---

## Invoking the Screen Wizard

---

To use the screen wizard, you must:

- Open or create a repository—The repository must contain at least one table that inherits from @DATABASE (specified in the object's Inherit From property). This property setting is the result of importing the desired database objects. (Refer to “Creating and Opening a Repository” [on page E-19](#) for details.)
- Import the desired database objects (tables, views, and/or synonyms) to the open repository if the objects are not already imported. (Refer to “Populating a Repository with Database Objects” [on page E-25](#) for details.)

The screen wizard stores and uses information in the open repository to build new screens.

Now you can use the screen wizard.

## How to Run the Screen Wizard

1. Choose File→New→Screen (or the New Screen button on the toolbar). The New Screen Tool dialog box opens.



**Figure 4-1** You can access the screen wizard whenever you create a new screen.

If the template screens in the repository are from an old version of the screen wizard, you will be asked whether to replace them. If you have made changes or additions to the template screens, you should back them up first, let the screen wizard overwrite them, and then redo your changes.

**Note:** If the screen wizard needs to add or update its template screens, make sure you have write permission on the open repository. If the repository already contains valid template screens, write permission is not necessary to run the screen wizard.

2. Choose Yes. The Format Selection dialog box opens.

If you choose No, a blank, untitled screen opens in the screen editor workspace. If you choose Cancel, the new screen operation is canceled.

From this point, you are guided through several dialog boxes where you:

- Specify the screen's overall appearance.
- Choose the contents of the master section of the screen.
- Choose the contents of the detail and/or subdetail sections (if any) of the screen.
- Specify the layout: single row or grid display of data.

- Choose the architecture and types of screens that you want the screen wizard to generate.
- Specify the services that you want the screen to use (for JetNet/Oracle Tuxedo applications).
- Choose the types of controls (push buttons or menu bar/toolbar) for the client screens.

---

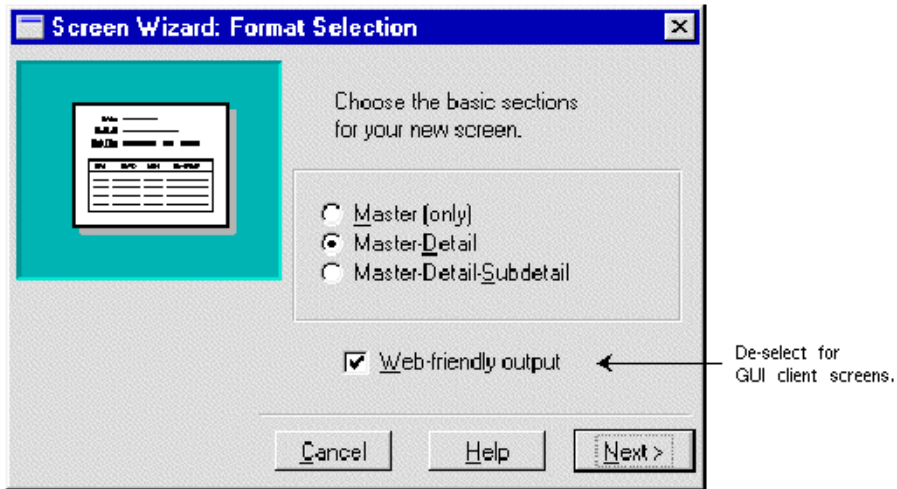
## Screen Wizard Dialogs

---

The screen wizard guides you through a variety of dialog boxes, prompting you for information it needs to create the appropriate presentation for your database application screen. This section describes each of the dialog boxes.

### Format Selection Dialog

The Format Selection dialog requests that you choose the basic structure of the new screen. Each format option determines the number of sections that will appear on a screen.



**Figure 4-2** The Format Selection dialog lets you choose the basic structure of a screen.

## How to Define the Screen's Format

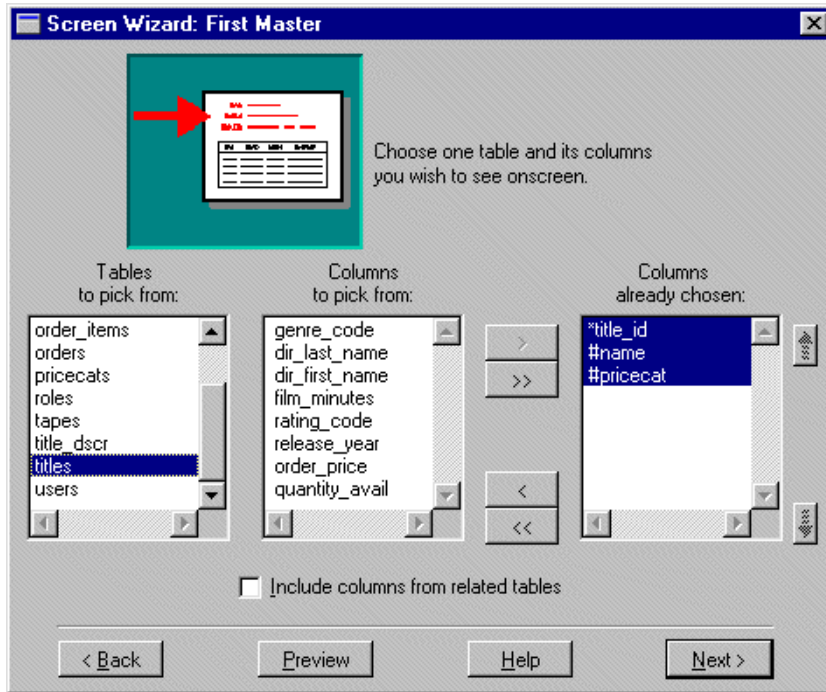
1. Select the radio button for the desired format:
  - Master (only)—Creates a simple screen comprised of a single section, for example, a table maintenance screen. It is intended to manipulate only one table.
  - Master-Detail—(default) Creates a screen with two sections. For instance, a master section might display a recipe, and the detail section list its ingredients.
  - Master-Detail-Subdetail—Creates a screen having three sections. For instance, if a master section displays a list of stores, the detail section lists the sales for the selected store, and the Subdetail section lists the contents of each sale.
2. If the screen is intended for use in a Web application, set the Web-Friendly Output option.

If this option is checked, the wizard creates screens from the `smwizweb` template screen and enforces these Web-specific restrictions:

- Menus and toolbars are disallowed; only push button widgets are allowed as controls for database transactions. These push buttons are positioned at the top of the screen. Because Web database transactions exclude new row operations, the New and Continue push buttons are not created.
  - If you specify a master-detail or master-detail-subdetail format, the Layout Selection dialog disallows grid display for all parent sections. For example, given a master-detail-subdetail format, only the subdetail section can use a grid display; the parent sections (master and detail) must use single-row format.
  - Selection screens are disallowed.
3. Choose Next to proceed to the First Master dialog.

## **First Master Dialog**

The First Master dialog box is where you identify the first table for the Master section and choose which of its columns you want included on your screen.



**Figure 4-3** The First Master dialog displays a list of tables in the open repository. These were created as a result of importing database tables.

## How to Identify the First Master Table of Your Screen

1. Select one table from the Tables to Pick From list.

The list displays the names of all imported database tables in the open repository.

Once you select a table, the other two lists are populated:

- Columns to Pick From—Lists the names of all non-key columns associated with the selected table.
- Columns Already Chosen—Lists the names of all primary key columns and columns requiring data, if any. Primary keys are indicated with a leading asterisk (\*); primary keys defined within the screen wizard are indicated with a plus sign (+). You can reorder primary keys within the list, but you

cannot remove them. Columns which require data entry—specified as not null in the database—are indicated with a number sign (#).

2. To add or remove columns on your screen, do either of the following:
  - Double-click on columns in the Columns to Pick From list or double-click on columns in the Columns Already Chosen list.
  - Select columns in the Columns to Pick From list and choose the > button; use the >> button to add all columns; or select columns in the Columns Already Chosen list and choose the < button; use the << button to remove all columns.

The contents of the Columns Already Chosen list tells the screen wizard which columns to include on the new screen, and the order in which to display them.

**Note:** Click+drag or Shift+click selects contiguous items, and Ctrl+click selects non-contiguous items.

3. (Optional) Reorder columns in the Columns Already Chosen list by selecting the column and using the Up or Down reposition buttons accordingly.

The highlighted items shift up or down within the list, one position at a time.

4. (Optional) To add columns from other tables to the Master section of the screen, choose the Include Columns from Related Tables check box.

The Include Columns from Related Tables check box is active if one or more link widgets are present on the repository screen of the selected table; otherwise, it is inactive.

5. Choose Next.

If the first table you selected has a primary key definition, one of the following occurs:

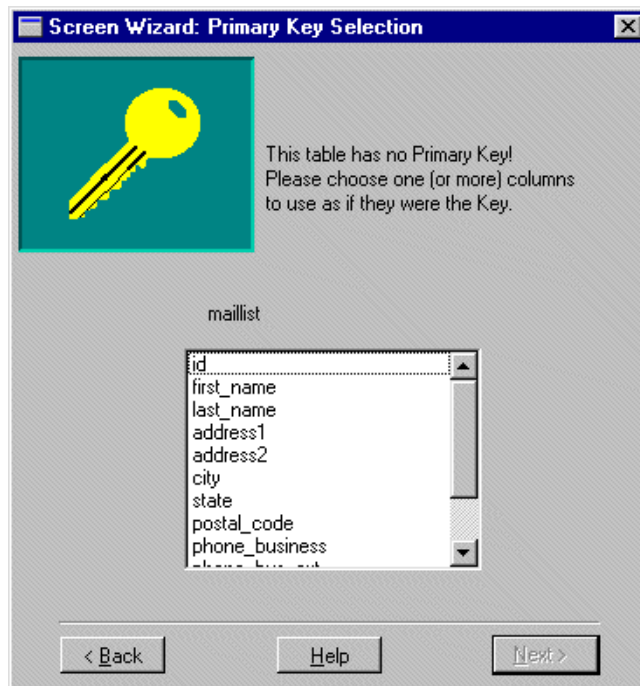
- If you selected the Include Columns from Related Tables check box, the Additional Master dialog opens. (See [page 4-9](#), “Additional Master Dialog.”)
- If you selected a Master-only format, the Layout Selection dialog opens. (See [page 4-15](#), “Layout Selection Dialog.”)
- If you selected either a Master-Detail or Master-Detail-Subdetail format, the First Detail dialog opens. (See [page 4-11](#), “First Detail Dialog.”)

If the selected table does not have a primary key, the Primary Key Selection dialog opens.

## Primary Key Selection Dialog

A primary key is the column, or combination of columns, that uniquely identifies each row in a database table. If the first table you have selected does not have a primary key definition, the Primary Key Selection dialog opens where you can identify a primary key for the table.

**Note:** The Primary Key Selection dialog does not open for any additional tables in the master section because the screen wizard assumes these are non-updatable tables.



**Figure 4-4** The Primary Key Selection dialog is where you define your primary keys.



## How to Define a Primary Key

1. Select one (double-click) or more columns from the list.

Click+drag or Shift+click selects contiguous columns; Ctrl+click selects non-contiguous columns.

2. Choose Next.

Once a primary key is defined, the screen wizard can proceed (refer to the First Master Dialog section, step 6).

Primary Key columns that you define in the wizard are displayed with a leading plus sign (+) to distinguish them from Primary Key columns defined in the repository. Other than this, the screen wizard treats these columns exactly the same.

**Note:** If a user-defined Primary Key is used often, you should set the Primary Keys property (under Database) on the appropriate repository entry.

## Additional Master Dialog

The Additional Master dialog opens if you have selected the Include Columns from Related Tables check box on the First Master dialog. Use this dialog to identify additional tables for the Master section and to specify which of their columns you want included on your screen. The following general rules can assist you in determining when a table is best used as an additional table or as the first table of the next section.

- If the table is used to turn a code number into a text translation, then it is an Additional table. A table that fits this description is commonly known as a look-up table. For example, `PG` is the code for movie ratings and `parental guidance` is the text associated with the code.
- If the table contains at most one row that matches a row in the First table, then it is an Additional table. A table that fits this description is said to have a one-to-one relationship with the First table.
- If the table contains many rows that match a row in the First table, then it is the First table of the next section. A table that fits this description is said to have a one-to-many relationship with the First table.

## How to Include Columns from Other Tables in the Master Section

1. Select a table from the Tables to Pick From list.

This is a hierarchical list of all tables named in links on the repository screen of the First Master table you selected. It also lists tables related to those additional tables. Therefore, you can select columns from the immediate child tables and/or from their descendants.

Once you select a table, the other two lists are populated:

- Columns to Pick From—Lists the names of all columns associated with the selected table.
  - Columns Already Chosen—All items previously selected and associated with the selected table are highlighted.
2. To add or remove columns on your screen, do either of the following:
    - Double-click on columns in the Columns to Pick From list or double-click on columns in the Columns Already Chosen list.
    - Select columns in the Columns to Pick From list and choose the > button; use the >> button to add all columns; or select columns in the Columns Already Chosen list and choose the < button; use the << button to remove all columns.

The contents of the Columns Already Chosen list tells the screen wizard which columns to include on the new screen, and the order in which to display them.

**Note:** Click+drag or Shift+click selects contiguous items, and Ctrl+click selects non-contiguous items. You cannot remove selected columns in the Columns Already Chosen list if they are from the First Master table.

3. (Optional) Reorder columns in the Columns Already Chosen list by selecting the column and using the Up or Down reposition buttons accordingly.

The highlighted items shift up or down within the list, one position at a time.

4. (Optional) To add columns from another table to the Master section of the screen, repeat steps 1 through 3.

The screen wizard collects all of your chosen columns from all of your selected tables.

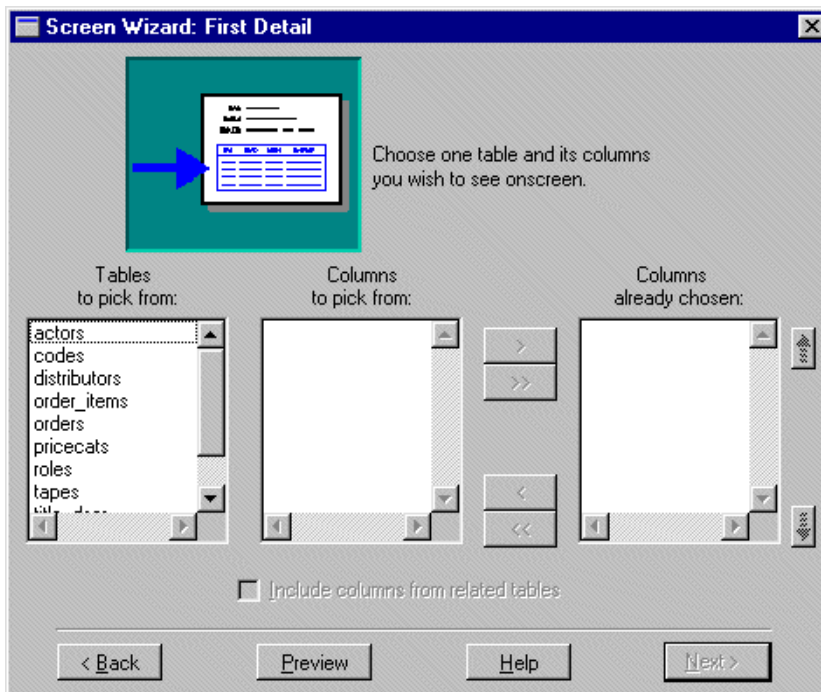
5. Choose Next.

Depending on your specifications, the screen wizard proceeds by doing one of the following:

- If you selected a Master-only format, the Layout Selection dialog opens. (See [page 4-15](#), “Layout Selection Dialog.”)
- If you selected either a Master-Detail or Master-Detail-Subdetail format, the First Detail dialog opens.

## First Detail Dialog

The First Detail dialog is identical to the First Master dialog. Use this dialog to identify the First table for the Detail section and to select which of its columns should be included on your screen.



**Figure 4-5** The First Detail dialog is where you define what fields display in the Detail section of your screen.

## How to Identify the First Detail Table of Your Screen

For identifying the First Master table of your screen, follow the directions as described [on page 4-5](#) under “First Master Dialog.”

The Tables to Pick From list shows all tables in the open repository, but does not include the First Master table you selected.

When you choose Next, the screen wizard does the following if the table you selected has a primary key definition:

- If you selected the Include Columns from Related Tables check box, see [page 4-12](#), “Additional Detail Dialog.”

If the table you selected has a primary key definition but does not have a link, then the Master-Detail Link dialog appears. However, if the table does have a link, then the screen wizard does one of the following:

- If you chose a Master-Detail format, see [page 4-15](#), “Layout Selection Dialog.”
- If you chose a Master-Detail-Subdetail format, see [page 4-15](#), “First Subdetail Dialog.”

If the selected table does not have a primary key definition, see [page 4-8](#), “Primary Key Selection Dialog.”

## Additional Detail Dialog

The Additional Detail dialog opens if you have selected the Include Columns from Related Tables check box on the First Detail screen. Use this screen to identify additional tables for the detail section and which of their columns should be included on your screen.

## How to Include Columns from Additional Detail Tables

Follow the directions as described for identifying Additional Master tables for your screen [on page 4-9](#) under “Additional Master Dialog.”

When you are finished selecting additional tables and their respective columns for the Detail section of your screen, choose Next. The screen wizard checks for a link widget in the open repository that connects the First Master table to the First Detail table. If a usable link is found, the screen wizard proceeds by doing one of the following:

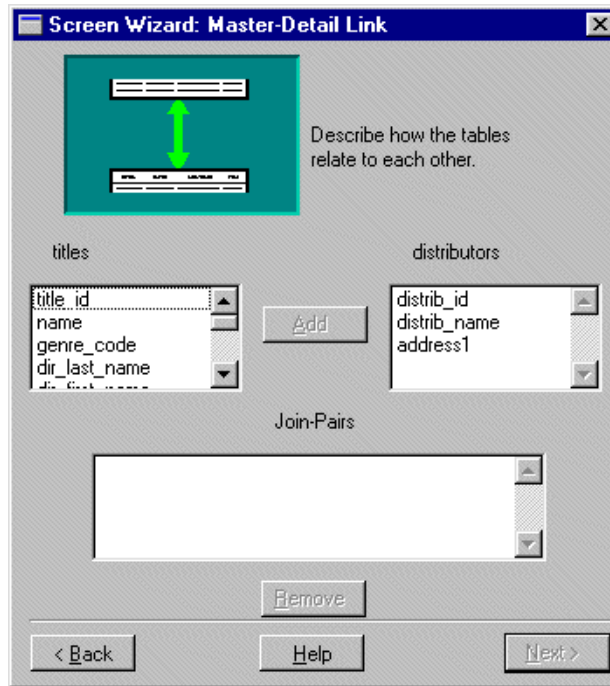
- If you selected a Master-Detail format, see [page 4-15](#), “Layout Selection Dialog.”
- If a Master-Detail-Subdetail format was selected, see [page 4-15](#), “First Subdetail Dialog.”

If there is no link widget to connect the First Master table to the First Detail table, the Master-Detail Link dialog opens.

## Master-Detail Link Dialog

It is required that the First Master and First Detail tables be joined to each other by a link widget. The link defines the relationship between the two tables by keeping a list of which columns in the First Master should be matched to columns in the First Detail. If you have selected a First Detail table that does not have a link to the First Master table on your screen, the Master Detail Link dialog opens.

**Note:** The Master-Detail Link dialog does not appear for additional table selections because the screen wizard only displays additional tables for which links already exist.



**Figure 4-6** The Master-Detail Link dialog is where you define the relationship that connects the First Master and First Detail tables.

## How to Define a Master-Detail Link

1. Select one column from the Master list (left).
2. Double-click on one column from the Detail list (right) to complete the selection (or select the column and choose Add).

The selected columns move to the Join-Pairs list.

3. To add more join conditions, repeat steps 1 and 2 as many times as required.  
The screen wizard collects all of your chosen conditions in the Join-Pairs list.
4. To remove a join condition that you have made, double-click on the entry in the Join-Pairs list (or select the entry and choose Remove).

5. Choose Next when you are done. Where you resume is determined by conditions on the previous First or Additional dialog.

## First Subdetail Dialog

The screen wizard allows you to specify First and Additional tables for the Subdetail section of your screen in exactly the same manner as for Detail tables.

### How to Identify the First Table of The Subdetail Section

The Tables to Pick From list box displays all tables in the open repository, but does not include the First Master or Detail tables you already selected.

For specific information, follow the directions as described for identifying the First Master table.

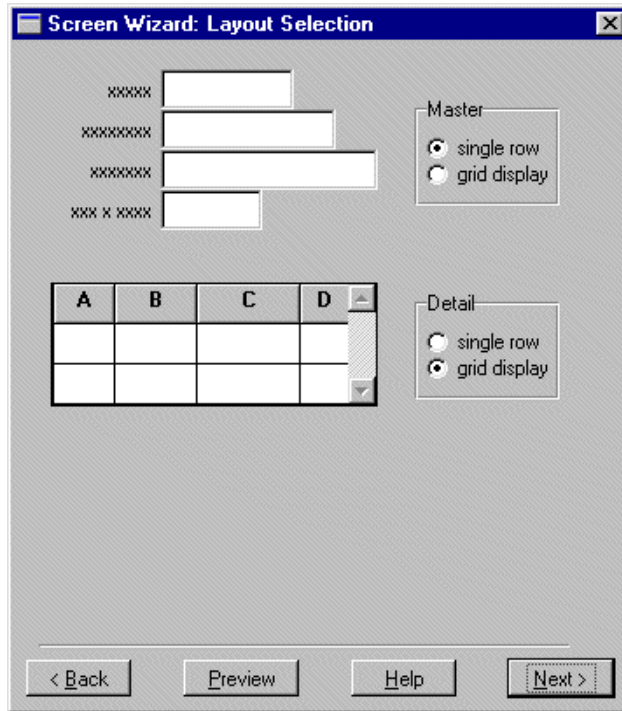
1. Choose exactly one First Subdetail table.
2. Select and reorder the columns as necessary.
3. (Optional) Select the Include Columns from Related Tables check box to select columns from another table. The Additional Subdetail dialog opens and is essentially identical to the Additional Detail dialog.
4. If necessary, the Primary Key Selection dialog opens where you define the table's Primary Key.
5. If necessary, define a Detail-Subdetail Link.
6. Choose Next. The Layout Selection dialog opens.

## Layout Selection Dialog

The Layout Selection dialog opens when you have made all table and column selections you require for your screen. Use this dialog to define the presentation layout for each section. The screen wizard chooses the most reasonable defaults for your screen specification. However, you can easily experiment with all possible variations for each section of your screen. Sample layouts are displayed on the Layout Selection dialog.

## How to Define Each Section's Layout

1. Choose the desired radio button for each section. You will immediately see what that section will look like.



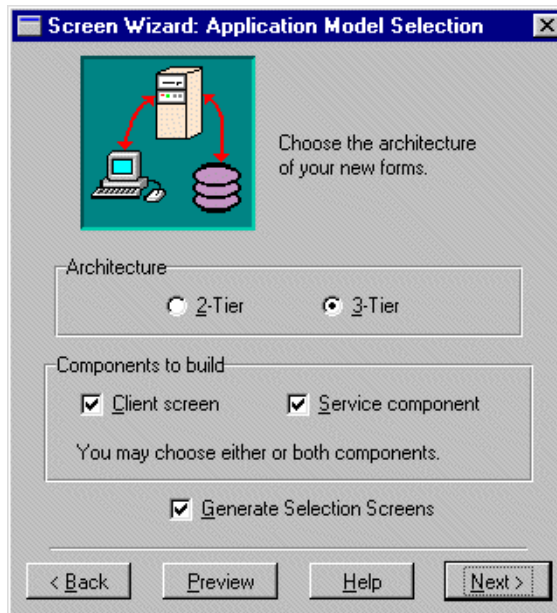
**Figure 4-7 Choose either single row layout or grid display layout to display data.**

- Single row—The names of the columns in the section are shown in label widgets, and the values of the columns are shown in single line text widgets.
  - Grid display—The names of the columns in the section are shown in the grid's column headings, and the values of the columns are shown in the grid's cells.
2. Choose Next. The Application Model Selection dialog opens.



## Application Model Selection Dialog

In JetNet and Oracle Tuxedo executables, the Application Model Selection dialog is used to define the architecture and specify the screens that you want to generate. The screen wizard can generate client screens, service components, selection screens (if any) and their corresponding selection service components (if any) depending on what you specify. For an overview of two-tier and three-tier architectures, refer to Chapter 4, “Defining Application Architecture,” in *Application Development Guide*.



**Figure 4-8** Choose the architecture of your screens and specify the screens that you want to generate.

### How to Define the Architecture of Your Screens

1. Choose the desired Architecture type:
  - 2-Tier—Generates a client screen. The Components to Build section becomes inactive.
  - 3-Tier—(default) Specify whether you want a client screen, a service component, or both.

A client screen provides a user-interface for your application. A two-tier client screen is functionally ready to go; however, for a three-tier client screen to be functionally complete, you must define the services it uses in the JIF.

A service component is identical to a client screen except that it has no push buttons and no menu/toolbar controls, since there is no interaction between a user and a service component. Also, there is no screen title on a service component.

2. To control the creation of selection screens (or pick lists), do one of the following:
  - Choose the Generate Selection Screens check box (default) to automatically create selection screens, selection service components, or both for your Additional table specifications. If you have not specified any Additional tables, the check box is inactive.
  - Deselect the Generate Selection Screens check box to cancel the creation of selection screens.

Selection screens, sometimes called pick lists, are most helpful when entering new records. They display a list of acceptable values for a field (or fields).

The selection service component facilitates the transfer of data from the server to the selection screen on the client side of the application. The server fetches this data from the database and provides the selection screens on the client side of the application with a list of acceptable values for a field (or fields).

3. Choose Next.

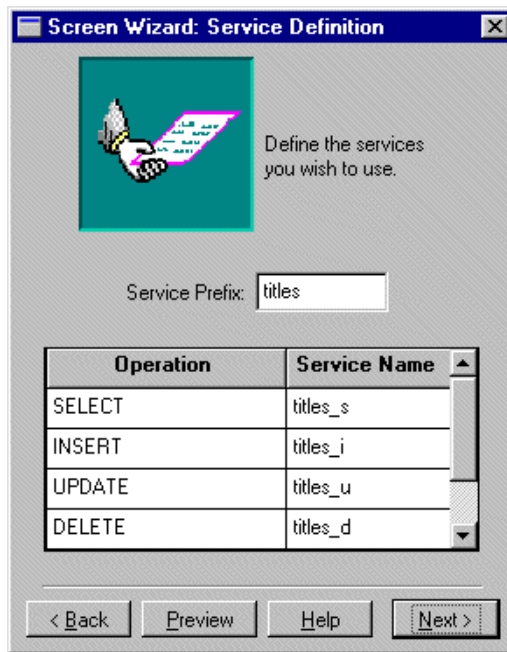
Depending on your specifications, the screen wizard proceeds by doing one of the following:

- If you chose a two-tier architecture or chose to generate only a service component in a three-tier architecture, see [page 4-21](#), “Style and Finish Dialog.”
- If you chose to generate only a three-tier client screen or a client screen and a service component in a three-tier architecture, the Service Definition dialog opens.

## Service Definition Dialog

In JetNet and Oracle Tuxedo executables, the Service Definition dialog appears only if you chose to generate a three-tier client screen. You use the Service Definition dialog to specify the services that are used by the screen. They are set as Service properties for the First Master table view widget on the client screen and on its selection screens (if any); the Service property is also set for link widgets on the client screen.

**Note:** You must also define these services in the JIF using the same service names. Refer to Chapter 24, “JIF Editor,” for information on defining services in the JIF.



**Figure 4-9** The Service Definition dialog is where you can specify the names of services.

The Service Definition dialog consists of the Service Prefix field and a list of screen wizard-named services. The list includes:

- The standard types of transaction manager commands that a service implements: Select, Insert, Update, and Delete operations. In addition, if you specified Additional tables, you will get a Link service per validation link; if you specified Selection Screens on the client side of the application, you will get a Choose service per selection screen.
- Default service names associated with each operation consist of the Service Prefix, followed by the operation type suffix. The operation type suffix consists of one or two characters designating the operation type; for example `_s` for a `SELECT` operation.

**Table 4-1 Default service name conventions.**

Screen type	Set on	Operation type	Service name
Client screen	Master table view	SELECT	<i>servicePrefix_s</i>
Client screen	Master table view	INSERT	<i>servicePrefix_i</i>
Client screen	Master table view	UPDATE	<i>servicePrefix_u</i>
Client screen	Master table view	DELETE	<i>servicePrefix_d</i>
Client screen	Link	LINK_ <i>table</i> *	<i>servicePrefix_l#</i>
Selection screen	Master table view	CHOOSE_ <i>table</i> *	<i>servicePrefix_c#</i>

\* The word *table* stands for *additional Table*.

## How to Edit Service Names

1. Enter a service prefix in the Service Prefix field if you do not like the suggested one (name of the First Master table) and press TAB.
 

**Note:** A service prefix entry is required.

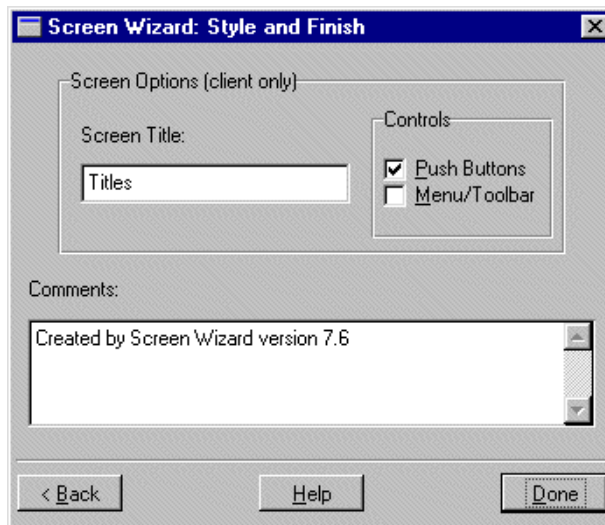
The list of unedited service names is updated with the new prefix specification.
2. (Optional) Edit the individual service names as desired.

**Note:** The Service properties on the client screen are set to the service name values.

3. Choose Next. The Style and Finish dialog opens.

## Style and Finish Dialog

The Style and Finish dialog is where you make the final design decisions.



**Figure 4-10** Add the final touches to your screen on the Style and Finish dialog.

**Note:** The Screen Options are only applicable to client screens.

## How to Add Final Touches to Your Screen

1. (Optional) Enter a new screen title in the Screen Title field if you do not like the suggested title. This specification displays in the screen's title bar and is set as the screen's Title property (under Identity).
2. Check the box for the types of controls that you want on your client screen:
  - Push Buttons—Adds a variety of transaction-specific buttons, for example, View, Select (for update), New, Delete, Save, and Reset as well as

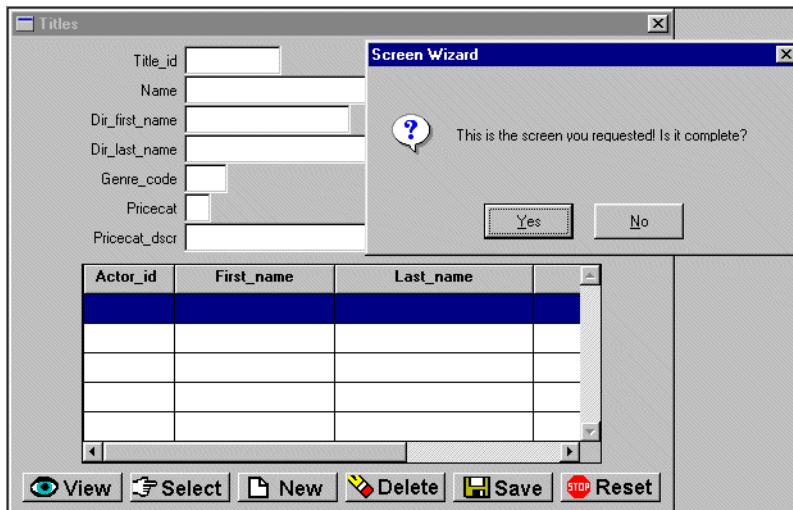
Next/Previous buttons and First/Last record buttons for a two-tier architecture.

- Menu/Toolbar—The menu provides some actions not available with the push buttons. In addition, bitmaps and pixmaps are included for the toolbar and push buttons.
3. (Optional) Enter additional comments in the Comments field. This is a useful place to keep information about this screen during the development process.

**Note:** The comments are written to all of the screens generated.

4. Choose Done.

The completed screen is displayed, along with the following prompt:



**Figure 4-11 Review the completed two-tier screen.**

5. Do either of the following:
- Choose No to return to the Style and Finish dialog.
  - Choose Yes if the screen is acceptable. The screen wizard returns you to the screen editor. Your new screen or screens and any selection screens are open for additional editing or for a trial run in test mode.

Table 4-2 lists the screens generated by the screen wizard and its default names depending on the architecture of your application and your selections in the Application Model Selection dialog.

**Table 4-2 Default names for wizard-generated screens and service components**

Screen type	Default name	Selection screens/service components (if any)
Two-tier client screen	(Untitled #)	<i>childTable.itm</i>
Three-tier client screen	(Client #)	<i>childTable.cit</i>
Three-tier service component	(Server #)	<i>childTable.sit</i>

In a two-tier architecture, you need a direct connection to the database and then you can immediately test your screens.

In JetNet and Oracle Tuxedo applications, you need a remote connection to the database to completely test your client screen and the services it uses. However, before you test your client screen, you must save the corresponding service component and selection service components (if any) to their appropriate libraries, and you must define the services in the JIF.

For further information on testing screens, refer to Chapter 38, “Testing Application Components,” in *Application Development Guide*.

## Output for Two-Tier Architecture

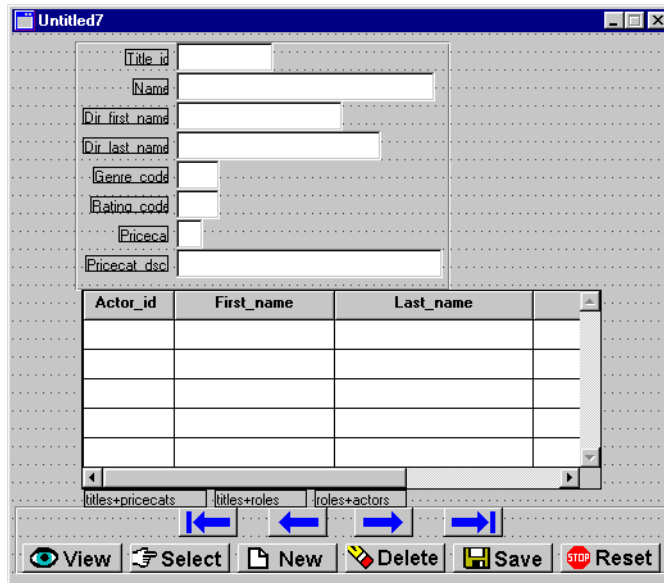
Two-tier output can consist of the following screens:

- Client screens.
- Selection screens.

A number of properties on the screen and on individual widgets are automatically assigned. Some of these settings will work without any modifications. For further information on property settings, refer to [page B-1](#), “Two-Tier Property Specifications.”

## Two-Tier Client Screen and How It Looks

The client screen provides a user interface for your application. You can rearrange widgets and add decorations, but functionally, the screen is complete and ready to go.



**Figure 4-12** A client screen in a two-tier architecture.

Sections with single row layouts are surrounded by a positioning region box to align the widgets inside it. The box does not display at runtime. Below all the sections, the screen wizard places any link widgets needed by the screen. Link widgets are not visible at runtime; therefore, their placement is strictly a matter of convenience.

Push buttons, if specified, appear along the bottom of the screen: four Continue buttons and six command-specific buttons. The buttons are surrounded by a positioning region box to keep the buttons centered horizontally on the screen. The box does not display at runtime.



## Two-Tier Selection Screens

Each selection screen is named for the table it references, followed by the file extension `.itm`. If a screen named `table.itm` already exists in an open library, the screen wizard does not regenerate it. If you need to recreate the selection screen, or want a new one for the same table, rename the old one before invoking the screen wizard.

Selection screens, sometimes called pick lists, are most helpful when entering new records. They display a list of acceptable values for a field (or fields). On the client screen, the selection screen is defined in the appropriate widget's Selection Screen property (under Help). If the property already has a value specification (via the repository), the screen wizard does not overwrite it and does not create a selection screen for the table associated with the widget.

---

# Output for JetNet and Oracle Tuxedo Applications

---

In JetNet and Oracle Tuxedo applications, three-tier output can consist of the following screens:

- Client screen.
- Service component.
- Selection screen.
- Selection service component.

When you generate a client screen and a service component, they are identical because a service component interacts with its client screen counterpart to transfer data to and from the appropriate services. The client screen and service component pair must be kept parallel. Thus, if you modify your client screen, you must also modify its corresponding service component in a similar fashion. This is true of selection screens and their corresponding selection service components.

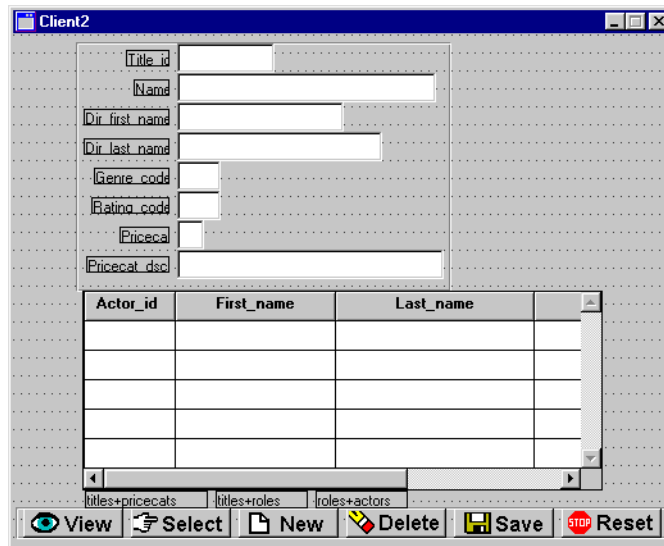
**Note:** Visual modifications (such as color, fonts, and pixmaps) on the client screen need not be duplicated on the service component.

A number of properties on the screen and on individual widgets are automatically assigned. Some of these settings will work without any modifications. For further information on property settings, refer to [page B-4](#), “Three-Tier Property Specifications.”

## Three-Tier Output and How It Looks

### Three-Tier Client Screen

The client screen provides a user interface for your application. It is light gray in color to distinguish it from its associated service component (highlight white background). You can rearrange widgets and add decorations. However, for your screens to be functionally complete and ready to go, you must define the services it uses in the JIF.



**Figure 4-13** A client screen in a three-tier architecture.

Sections with single row layouts are surrounded by a positioning region box to align the widgets inside it. The box does not display at runtime. Link widgets needed by the screen appear below the Master-Detail-Subdetail sections. Link widgets are not visible at runtime; therefore, their placement is strictly a matter of convenience.

The command-specific buttons, if chosen, appear at the bottom of the screen and are surrounded by a positioning region box. The positioning region box keeps the buttons centered horizontally on the screen.

## Service Component

Although the service component is never visible at runtime, it has a highlight white background to distinguish it from its associated client screen (light gray color) in the screen editor. It is used to transfer data from the clients screen to services and vice versa. You can rearrange widgets, but functionally, the screen is complete and ready to go.

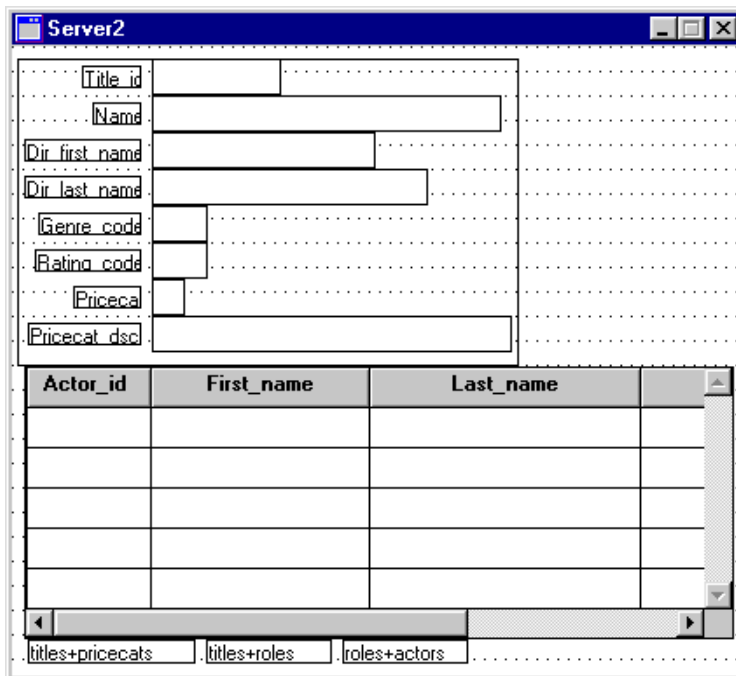


Figure 4-14 A service component.

A service component is identical to a client screen except that it has no push buttons and no menu/toolbar controls, since there is no interaction between a user and a service component. Also, there is no screen title on a service component.

## **Three-Tier Selection Screens**

For every Additional table you have specified via the screen wizard, a selection screen is generated (with a `.sit` extension).

Selection screens, sometimes called pick lists, are most helpful when entering new records. They display a list of acceptable values for a field (or fields). On the client screen, the selection screen is defined in the appropriate widget's Selection Screen property (under Help). If the property already has a value specification (via the repository), the screen wizard does not overwrite it and does not create a selection screen for the table associated with the widget. For further information on using selection screens, refer to [page 12-10](#), "Using Selection Screens."

## **Selection Service Components**

In a three-tier architecture, for every Additional table you have specified via the screen wizard, a selection service component is generated (with an `.sit` extension). It is identical to its corresponding selection screen, except that it does not have push buttons or a title bar.

A selection service component facilitates the transfer of data from the server to the selection screen on the client side of the application. The server fetches this data from the database and provides the selection screens on the client side of the application with a list of acceptable values for a field (or fields).

# 5 Report Wizard

The report wizard helps you design a Panther report that uses Panther's transaction manager to fetch data. It collects design information through a series of dialogs. Though simple to use, the report wizard builds sophisticated reports that you can use unchanged or easily edit for added functionality and refinements.

---

## Invoking the Report Wizard

---

To use the report wizard, you must:

- Open or create a repository—The repository must contain at least one table that inherits from @DATABASE (specified in the object's Inherit From property). This property setting is the result of importing the desired database objects. (Refer to “Creating and Opening a Repository” [on page E-19](#) for details.)
- Import the desired database objects (tables, views, and/or synonyms) to the open repository if the objects are not already imported. (Refer to “Populating a Repository with Database Objects” [on page E-25](#) for details.)

If you wish to add any items, such as `total_price`, to the repository, it is necessary to do so before invoking the report wizard to make the item available to the wizard. For more information on adding an item to the repository, refer to “Adding an Item to the Repository” [on page 4-18](#) in *Reports*.

The report wizard stores and uses information in the open repository to build new reports.

Now you can use the report wizard.

## How to Run the Report Wizard

1. Choose File→New→Report. The New Report Tool dialog box opens:



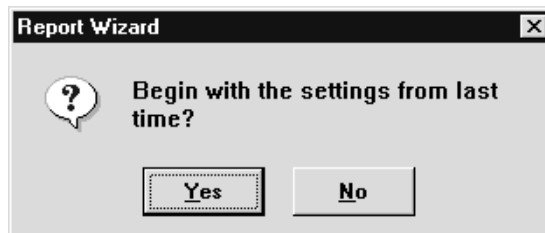
If the template report in the repository is from an old version of the report wizard, you are asked whether to replace it. If you made changes or additions to the template screen, first back it up, then let the report wizard overwrite it. Later, you can restore your changes to the new template.

**Note:** If the report wizard needs to add or update its template screen, make sure you can write to the open repository. If the repository already contains a valid template screen, write permissions are not necessary to run the report wizard.

2. Choose Yes to use the report wizard.

If you choose No, a blank, untitled layout window opens in the screen editor workspace. If you choose Cancel, the new report operation is canceled.

3. If you used the wizard to create a report earlier in this editing session, this dialog asks whether to start with your previous settings:



Choose one of the following:

- Yes to display previous choices in each dialog box. This lets you modify your previous report or quickly build a similar report.
  - No to make new selections for each wizard dialog.
4. The Table Selection dialog box opens.

From this point, you are guided through several dialog boxes where you make these choices:

- Report type: record-by-record, column, row, graph, matrix, or address labels.
- Root database table, which provides the basis of the report information.
- Database columns to appear in the report.
- Sort order of the data.
- Columns that are totaled and how those totals appear.

---

# Report Types

---

The report wizard can create six report types, described in the following sections.

## Record-by-Record

Record-by-record reports present each row in the select set individually. If you select group fields for the report, the data is sorted according to those groups, and a heading containing the group value is printed before the data. If any totals are requested, those totals appear after the data for each group.

In this example, `distrib_name` and `order_num` are group fields, and `total_price` is a total field.

<b>Distributors</b>		3/23/96
<b>Distrib_name Wellesley Hills</b>		
<b>Order_num 1002</b>		
Name	Cinema Paradiso	
Qty	1	
Total_price	25.00	
Name	Pulp Fiction	
Qty	8	
Total_price	192.00	
<hr/>		
<i>Total for</i>	<i>1002</i>	
<hr/>		
<i>Total_price</i>	<i>217.00</i>	
<hr/>		
<b>Order_num 1009</b>		
Name	Au Revoir les Enfants	
Qty	1	
Total_price	28.00	
Name	Beauty and the Beast	
Qty	3	
Total_price	60.00	
Name	Fugitive, The	
Qty	8	
Total_price	199.92	
Name	Siarman	
Qty	1	
Total_price	23.00	
<hr/>		
<i>Total for</i>	<i>1009</i>	
<hr/>		
<i>Total_price</i>	<i>310.92</i>	
<hr/>		
<i>Total for</i>	<i>Wellesley Hills</i>	
<hr/>		
<i>Total_price</i>	<i>527.92</i>	
<hr/>		
<u><i>Grand Total</i></u>		
<i>Total_price</i>	<i>3239.17</i>	

**Figure 5-1 Record-by-record reports list each row of data individually within group headings.**



## Column

Column reports present the data in columns. A single heading contains the column labels; the data is printed below each column label. If totals have been specified, they appear after the data for each group.

<b>Distributors</b>					4/1/96
<u>Distrib_name</u>	<u>Order_num</u>	<u>Name</u>	<u>Price</u>	<u>Qty</u>	<u>Total_price</u>
Somerville	1003	Bull Durham	25.00	1	25.00
		Quiz Show	24.00	9	216.00
		Room With A View, A	25.00	1	25.00
	<i>Totals for</i>	<i>1003</i>		<i>11</i>	<i>266.00</i>
	1010	Client, The	23.50	10	235.00
		Fugitive, The	24.99	10	249.90
		Pulp Fiction	24.00	10	240.00
	<i>Totals for</i>	<i>1010</i>		<i>30</i>	<i>724.90</i>
<i>Totals for</i>	<i>Somerville</i>			<i>41</i>	<i>990.90</i>
<u>Distrib_name</u>	<u>Order_num</u>	<u>Name</u>	<u>Price</u>	<u>Qty</u>	<u>Total_price</u>
Wellesley Hills	1002	Cinema Paradiso	25.00	1	25.00
		Pulp Fiction	24.00	8	192.00
	<i>Totals for</i>	<i>1002</i>		<i>9</i>	<i>217.00</i>
	1009	Au Revoir les Enfants	28.00	1	28.00
		Beauty and the Beast	20.00	3	60.00
		Fugitive, The	24.99	8	199.92
		Starman	23.00	1	23.00
	<i>Totals for</i>	<i>1009</i>		<i>13</i>	<i>310.92</i>
<i>Totals for</i>	<i>Wellesley Hills</i>			<i>22</i>	<i>527.92</i>
<b><u>Grand Totals</u></b>				<b><u>134</u></b>	<b><u>3239.17</u></b>

Figure 5-2 Column reports list the data in columns across the page.

## Row

Row reports present the data in rows across the page. Column labels are displayed on the left side of the page. To the right of each label are the values for that column. If totals are requested, row totals appear at the end of each row, and totals for higher-level groups appear after each set of row data.

<b>Distributors</b>					3/23/96
<b>Distrib_name</b>	Wellesley Hills				
<b>Order_num</b>	1002				
Name	Cinema Paradiso	Pulp Fiction		<i>Totals</i>	
Price	25.00	24.00			
Qty	1	8		<i>9</i>	
Total_price	25.00	192.00		<i>217.00</i>	
<b>Order_num</b>	1009				
Name	Au Revoir les Enfant	Beauty and the Beast	Fugitive, The	Starman	<i>Totals</i>
Price	28.00	20.00	24.99	23.00	
Qty	1	3	8	1	<i>13</i>
Total_price	28.00	60.00	199.92	23.00	<i>310.92</i>
<b>Totals for</b>	<b>Wellesley Hills</b>				
<i>Qty</i>	<i>22</i>				
<i>Total_price</i>	<i>527.92</i>				
<hr/>					
<b><u>Grand Totals</u></b>					
<b><u>Qty</u></b>	<b><u>134</u></b>				
<b><u>Total_price</u></b>	<b><u>3239.17</u></b>				

**Figure 5-3** In row reports, the data for each column appears in a row across the page.

## Graph

Graph reports display numeric data as pie or bar charts. A single report can contain a series of graphs depicting the detail section of the report, the totals section, or a combination of the two.

A graph in the detail section plots a numeric column's values for an entire group on one graph. If your report includes two or more numeric data columns, you can plot them on a single bar graph or plot each separately.

In this example, the detail data is displayed as text and in a graph while the totals are displayed as text only. The two numeric fields are displayed together in a multi-series bar graph.

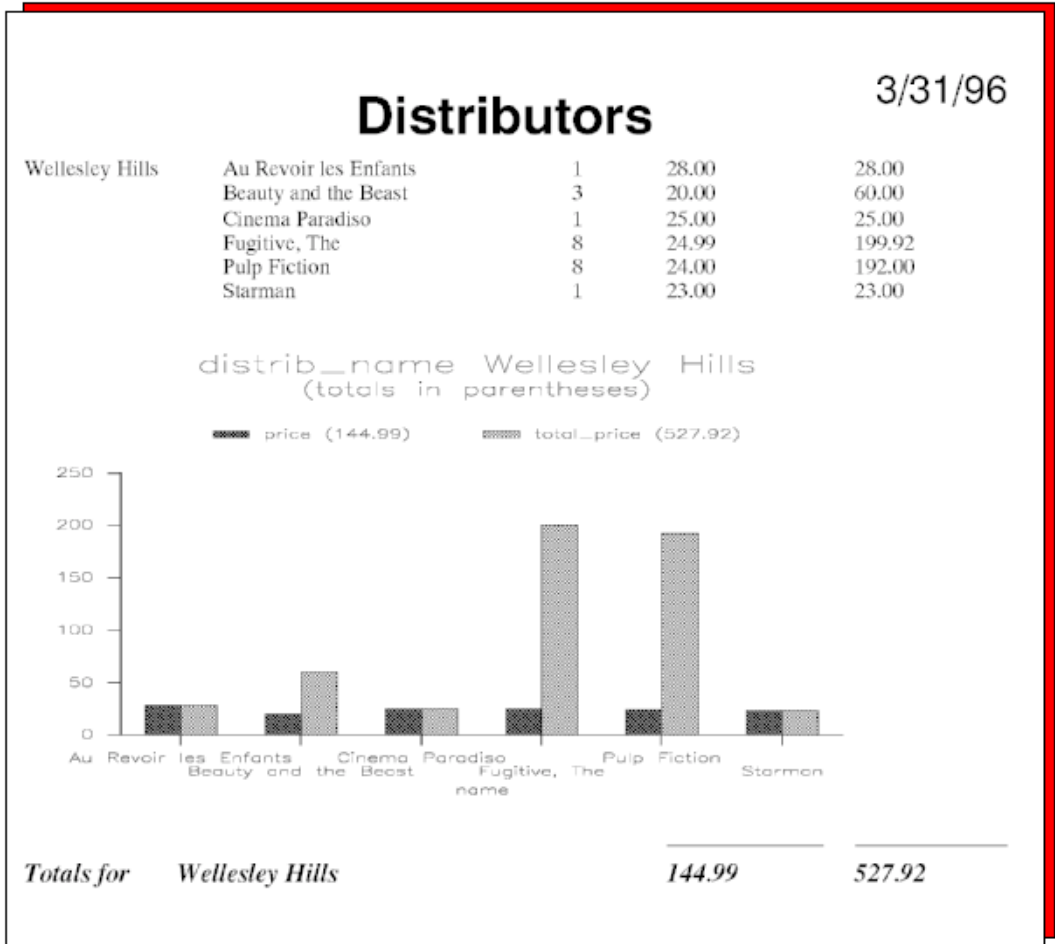


Figure 5-4 Graph reports can present detail data or totals in pie or bar charts.

## Matrix

A matrix report presents data in a cross-tab format with totals reported independently for each row and each column in the matrix. You can also select a data group that sorts the data so that the report contains a matrix for each group. Generally, numeric data is selected to be in the cells of the matrix itself, but the column and row headings can be any column in the database.

The data items are positional in the appropriate cell; the labels for the data items appear before the matrix, following the group name. In the following sample report, the cells report the data for the `qty` and `total_price` columns.

For each fetch, a cell contains the fetch's values for the database columns that label the matrix. If more than one fetch has data for a given cell, the cell shows their total.

<b>Distributors</b>				3/24/96
<hr/>				
<b>Distrib_name</b>				
Wellesley Hills				
<b>Qty</b>				
<b>Total_price</b>				
<hr/>				
Name	Po_num			Totals
	D1456	D1472	D1501	
Au Revoir les Enfants	0.00	0.00	1.00	1
	0.00	0.00	28.00	28.00
Beauty and the Beast	0.00	0.00	3.00	3
	0.00	0.00	60.00	60.00
Cinema Paradiso	1.00	0.00	0.00	1
	25.00	0.00	0.00	25.00
Fugitive, The	0.00	0.00	8.00	8
	0.00	0.00	199.92	199.92
Pulp Fiction	8.00	0.00	0.00	8
	192.00	0.00	0.00	192.00
Starman	0.00	0.00	1.00	1
	0.00	0.00	23.00	23.00
<hr/>				
<i>Totals</i>				
<i>Qty</i>	9	0	13	22
<i>Total_price</i>	217.00	0.00	310.92	527.92

**Figure 5-5 Matrix reports total the values both horizontally and vertically for each cell column.**

## Address Labels

Address label reports produce formatted output for label sheets. The wizard aligns the widgets vertically, one widget per line. After the wizard is finished creating this report, edit the layout window and move widgets to new positions if you want more than one per line.



**Figure 5-6** Address label reports lets you choose the number of columns per page and the number of rows per page.

# Report Wizard Dialogs

The report wizard guides you through a variety of dialog boxes, prompting you for information it needs to create the appropriate presentation for your report.

## Selecting Report Type

The first dialog displayed by the report wizard, Table Selection, asks you to select the report type and root table view—the database table that forms the basis of the report:



Figure 5-7 On the Table Selection dialog, choose the basic structure of the report.

1. Select the type of report:

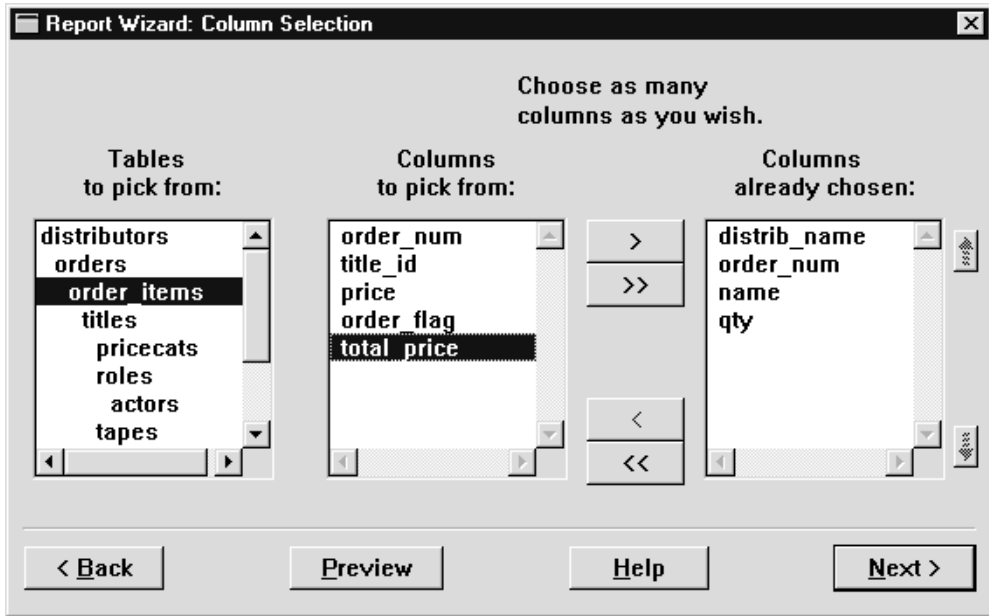


- Record-by-record—Creates a simple report comprised of a row-by-row representation of the data. See Figure 5-1 [on page 5-4](#) for a sample report.
  - Column—(default) Creates a report where the data appears in columns down the page with the column totals following each group. See Figure 5-2 [on page 5-5](#) for a sample report.
  - Row—Creates a report where the data appears in rows across the page. The totals for the primary group follow the group information. The totals for the lower-level data groups appear at the end of each row. See Figure 5-3 [on page 5-6](#) for a sample report.
  - Graph—Creates a report that displays numeric data in a pie or bar chart. (Graph options are also available with other report types.) See Figure 5-4 [on page 5-8](#) for a sample report.
  - Matrix—Creates a multi-column report with both horizontal and vertical totals. See Figure 5-5 [on page 5-10](#) for a sample report.
  - Address Labels—Creates a report with formatted output for label sheets. See Figure 5-6 [on page 5-11](#) for a sample report.
2. Select one of the database tables as the root table view.

The root table view is the database table that forms the basis of the report. For example, a report that shows orders for each distributor can use a table view that contains distributor names for its root table view. Your report can contain data from the root table view and any table view related to it.
  3. When you are done, choose Next.

## Choosing Data

In the Column Selection dialog, you specify the columns to appear in the report. When the dialog opens, it lists all table views that are related to the root table view. The list is organized according to the link relationships in the repository. The root table view you selected earlier is at the top of the list. Levels of indentation indicate how all other tables are linked to the root table view and to each other:



**Figure 5-8** The Column Selection dialog displays a list of table views in the open repository.

## How to Specify Columns for the Report

1. From Tables to Pick From, select a table view.  
When you select a table view, the adjoining list is populated with the names of columns in the selection.
2. To include columns in the report:
  - From Columns to Pick From, select one or more columns. Click+drag or Shift+click to select contiguous items, and Ctrl+click to select non-contiguous items.
  - Add the selections to the rightmost list, Columns Already Chosen, by choosing the > button; add all columns by choosing the >> button.

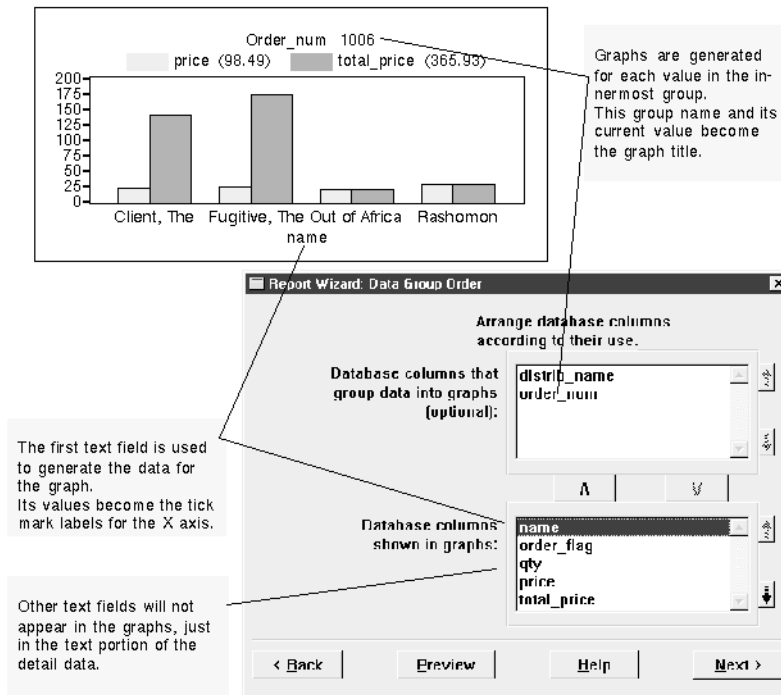
The contents of Columns Already Chosen tells the report wizard which columns to include in the report and the order in which to display them.

3. Add or remove items from Columns Already Chosen as desired.
4. To add more tables and their columns to the report, repeat steps 1-3.
5. To change the display order of columns, reorder the items in Columns Already Chosen:
  - Select the columns that you want to move
  - Use the Up or Down reposition buttons. The highlighted items shift up or down one position at a time.
6. When done, choose Next.

## **Grouping Data**


In the Data Group Order dialog, you specify which columns to use for grouping data. Groups can help clarify relationships among data; they also let you subtotal data. For some types of reports such as matrix reports, you must also select row and column headings.

This dialog contains two list boxes. The bottom list box contains the columns chosen in the previous dialog. The list box above it, initially empty, shows the columns selected to define data groups. The first item in this box defines the primary group. Each subsequent item defines another group that is subordinate to the one listed above it.



**Figure 5-9** In the Data Group Order dialog, select the columns that will sort the data.

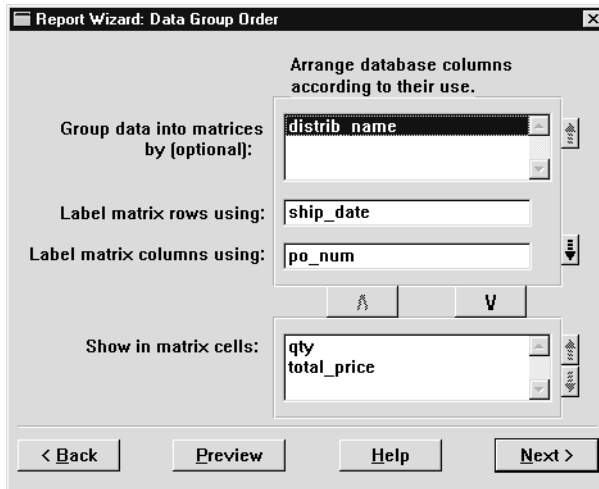
## How to Specify Columns for Grouping Data

1. Select a column from the bottom list box. Click+drag or Shift+click to select contiguous items, and Ctrl+click to select non-contiguous items.
2. Move the columns to the list box above by choosing .
3. Define the group hierarchy by reordering items in the top list box:
  - Select the columns from the list that you want to move.
  - Use the Up or Down reposition buttons. The selected items shift up or down one position at a time.
4. When done, choose Next.

## Matrix Data Groups



To create matrix reports, you must select labels for the data displayed in the matrix in addition to any group fields. For each group field, the report generates a separate matrix.

By default, the wizard lists all numeric columns in the bottom list box (Show in Matrix Cells):



**Figure 5-10** For matrix reports, choose row and column headings along with the data groups.

### How to Specify Columns for Grouping Data

1. Select one or more columns from the bottom list box to be a data group or a column or row label.
2. Move the selection up to Label Matrix Columns Using by choosing .
3. Select the next column from the bottom list box and move it up by choosing .

The previous selection moves up to the Label Matrix Rows Using list box.

4. Repeat step 3 until all groups and labels are selected.

5. Reorder selections so that the labels and group fields are in the desired order. Use the Up or Down reposition buttons to move items within a list.

In the final report, a separate matrix is created for the entry in the Group Data Into Matrices By list box. If more than one entry is listed, the first item in the group list specifies the primary group. Items below it specify subordinate groups in descending order.

Within each matrix, the cell entries are determined by the column label and the row label. When choosing the column label, you might want to consider the number of unique entries for that database column. The amount of space needed to report on each unique entry might exceed the width of the report. If this occurs, ellipses indicate the missing entries.

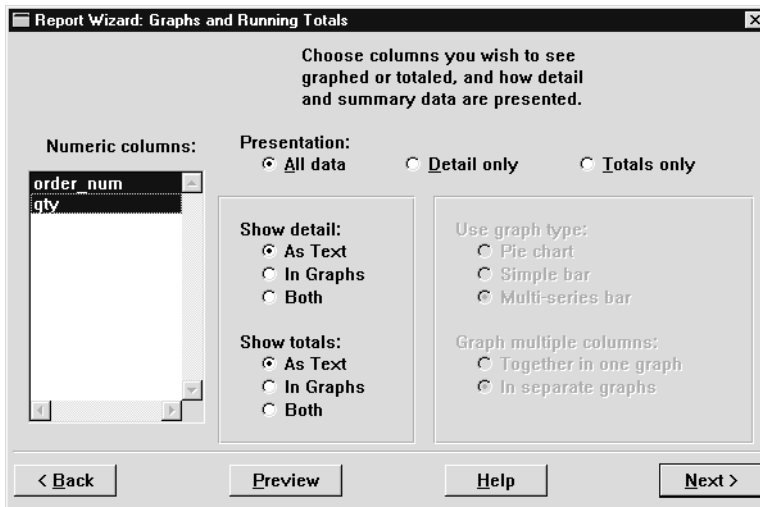
6. Reorder the items in Show In Matrix Cells:
  - Select the items that you want repositioned.
  - Use the Up or Down reposition buttons to shift highlighted items up or down one position at a time.

When this report executes, row and column totals appear for each numeric column.

7. When done, choose Next.

## Including Totals and Graphs

In the Graphs and Running Totals dialog, you decide which data to total and how to present the data. After numeric columns are selected, the default choices specify the display of all detail data with group totals. You can choose to view only the totals or only the detail data. You can also view data in graphs. The options that you can access in this dialog depend on the report type.



**Figure 5-11** The Graphs and Running Totals dialog lets you decide how to present report data.

## How to Include Running Totals in Your Report

1. From the Numeric Columns list box, select one or more columns. Each column that you select has subtotals calculated for all data groups.  
Click+drag or Shift+click to select contiguous items, and Ctrl+click to select non-contiguous items.
2. The options available for this report type become active.

## How to Specify the Report's Presentation Format

1. In Presentation, select one of these check boxes:
  - All data—Include both detail data and the totals.
  - Detail only—Include only detail data.
  - Totals only—Include only totals. This option is only available for Column and Graph reports.
2. In Show Detail, specify whether the data appears as text, graphs, or both.

3. In Show Totals, specify whether the totals appear as text, graphs, or both.  
For matrix reports, Show Totals is the only available option because each cell in the report already totals all database rows matching the cell's row and column headings.
4. For graphs, you can specify additional graph options:  
Use graph type
  - Pie chart—The data is displayed in a pie chart.
  - Simple bar—The data is displayed in a bar chart.
  - Multi-series bar—Sets of data are displayed using multiple bar charts for each set.Graph multiple columns
  - Together in one graph—The values for all columns are in one graph.  
If you choose Pie Chart or Simple Bar, each graph shows the data of one fetch. If you choose Multi-series Bar, the graph plots the history of all fetches for the innermost group.
  - In separate graphs—The values for each data set are in separate graphs. Each one shows the history of the data column's values for all fetches in the innermost group.
5. When done, choose Next.

## Finishing Up

In the Final Report Settings dialog, you set some final presentation options. The content of this dialog varies for each report type:





**Figure 5-12** Add the final touches to your report on the **Final Report Settings** dialog.

### How to Add Final Touches to Your Report

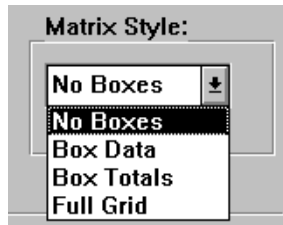
1. Edit the Report Title entry. This title appears in the page headers.
2. For row reports, specify how to handle data overflow—that is, when the length of the row exceeds the width of the report.



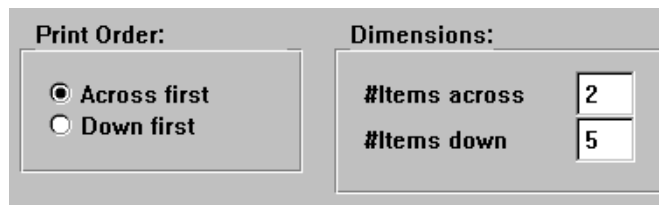
Choose Wrap or Elide:

- Wrap—Wrap the overflow data onto the next line.
- Elide—Display one line of data and enter ellipses to indicate overflow.

3. For matrix reports, select a style from this list box:



- Box Data—Place a box around the detail data.
  - Box Totals—Place separate boxes around the data and the totals.
  - Full Grid—Display the data in a grid.
4. For address label reports, select the print order and dimensions:



- Print Order determines if each set of data, after being sorted, is printed across the page first or down the page first.
  - Dimensions determines the number of labels both across and down.
5. Choose Done.

The screen editor resumes control and displays the layout window for the completed report. To test the report output, connect to a database and choose either Report→Preview Report or File→Test Mode.

---

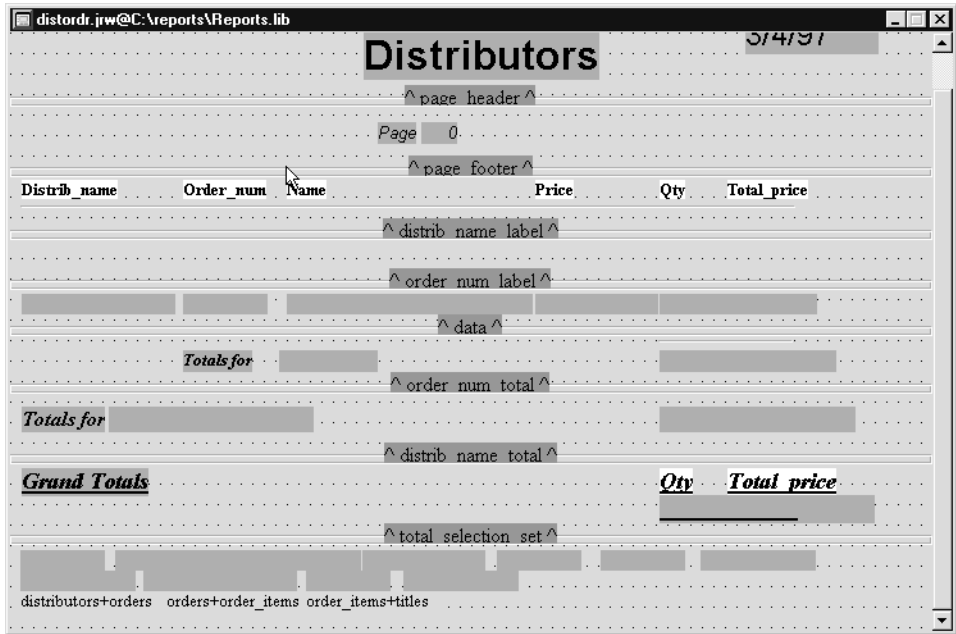
## How It Looks

---

When the report wizard is complete, the layout window for your report appears in the editor workspace. You can save it and use it immediately, or you can continue to work on it—rearrange widgets, add decorations, or change property settings.

### Layout Window

The layout window contains the layout areas created for the report. A horizontal line across the entire window defines the end of each layout area; the name of each layout area appears at the center of this line. The order of the layout areas is independent of how they appear in the report, which is determined by the report structure.



**Figure 5-13** The layout view contains the layout areas formatted as they will appear in the report.

Generally, the layout window for wizard-created reports contains layout areas for these report components:

- Page header—Contains a pixmap with the Panther logo, the report title, and the current date.
- Page footer—Contains the page number.
- Group header labels—Depending on the report type, there are one or more label areas. Most reports have a separate label area for each group level, named after the database column that defines the group.
- Detail output—Contains the format of the detail data section.
- Chart—Contains a graph widget. Note that the properties that determine data sources for this graph are set at runtime.

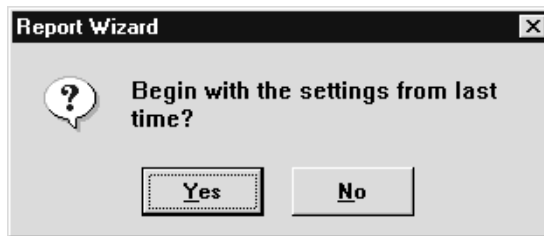
- **Group totals**—Depending on the report type, there are one or more total areas used to output group footers. A total area is created for each column selected in the Graphs and Running Totals dialog. The layout area `total_selection_set` outputs grand totals.

All widgets that output data are located in layout areas. The unnamed area of a layout window—that is, the space below all named layout areas—contains widgets that do not output data. The wizard places link widgets and history widgets in this area. They are needed to calculate and fetch data for the report, but they are not needed in the report layout itself.

For further information on report layout, refer to Chapter 2, “Introducing Report Layout,” in *Reports*.

## Report Structure Window

The report structure window displays a diagram of the report structure. Each report element—format, data groups, detail data, and the report itself—has a corresponding node in the structure:



**Figure 5-14** The report structure window schematically shows how a report executes.

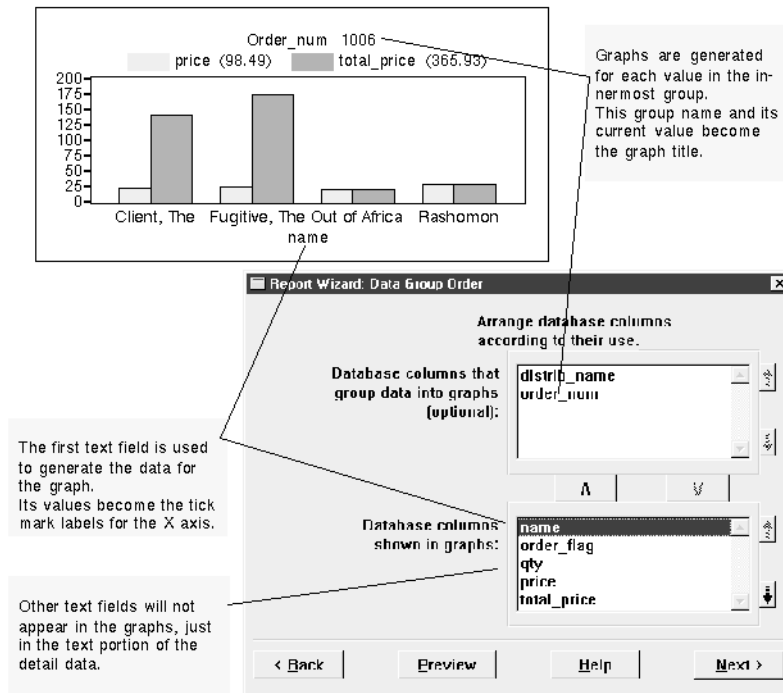
The main report node appears at the top of the report structure; its page format node is directly below it. The detail node specifies how the data is fetched for the report. Each data group level has a corresponding group node whose footer performs the calculations for the group totals. The print nodes attached to the detail and group nodes have associated layout areas to provide the report output.

For further information on report structure, refer to Chapter 3, “Introducing Report Structure,” in *Reports*.

# Including Graphs

Numeric data in your report can be presented as a graph. Both pie and bar charts are available and can be specified in the Graphs and Running Totals dialog.

The selection of data groups determines which column values are selected to appear in the graph. The graph is created for the innermost data group.



**Figure 5-15** This graph, which is for an order in the vbizplus database, shows the film titles that are included in the order and the price and total price for each film title.

Because graphs display numeric data, text columns in the data set are discarded if only graphing options are chosen for the detail data section.

If your report includes two or more numeric data columns, you have a choice of graphing options. You can choose to plot them on a single graph. Alternatively, you can graph each separately, and so plot the column's values for an entire group in one graph.

The image shows a screenshot of the 'Report Wizard: Graphs and Running Totals' dialog box. The dialog is titled 'Report Wizard: Graphs and Running Totals' and contains the following elements:

- Choose columns you wish to see graphed or totaled, and how detail and summary data are presented.**
- Numeric columns:** A list box containing 'price', 'total\_price', and 'qty'. 'price' and 'total\_price' are selected.
- Presentation:** Radio buttons for 'All data' (selected), 'Detail only', and 'Totals only'.
- Show detail:** Radio buttons for 'As Text', 'In Graphs', and 'Both' (selected).
- Show totals:** Radio buttons for 'As Text', 'In Graphs' (selected), and 'Both'.
- Use graph type:** Radio buttons for 'Pie chart', 'Simple bar', and 'Multi-series bar' (selected).
- Graph multiple columns:** Radio buttons for 'Together in one graph' and 'In separate graphs'.
- Buttons: '< Back', 'Preview', 'Help', and 'Next >'.

Callouts provide additional information:

- Top-left:** 'The value of each item determines the height of the bar.' (points to the y-axis of the graph preview).
- Top-right:** 'Use Graph Type determines whether it is a pie chart, a bar chart, or a multi-series bar.' (points to the 'Use graph type' section).
- Middle-left:** 'Totals for the selected columns appear in the graphs.' (points to the 'Show totals' section).
- Bottom-left:** 'Other numeric fields will not appear in the graphs, just in the text version of the detail data.' (points to the 'Numeric columns' list box).
- Bottom-right:** 'Select whether totals and graphs appear in the report output.' (points to the 'Show detail' and 'Show totals' sections).

**Graph Preview Data:**

Order_num	Client, The	Fugitive, The	Out of Africa	Rashomon
price (98.49)	~25	~25	~25	~25
total_price (365.93)	~140	~175	~25	~25

**Figure 5-16** In this report, the detail section of the report will contain text output and business graphs while the totals section will contain only business graphs

**Note:** A report that runs in character mode omits display of graphs. However, a layout area is created for the graph, which displays on other platforms.





---

---

# Part III User Interface Components

Defining Screen Properties

Defining Service Components

Defining Widget Behavior

Manipulating Widgets

Controlling the Way Things Look

Specifying Colors

Providing Help Facilities

Display Widgets

Data Entry Widgets

Grid Widgets

Tab Controls

Framesets and Splitters

ActiveX Controls

Push Button Widgets

Selection Widgets

Graphics Widgets

Table Views and Links



# 6 Defining Screen Properties

The role of the screen is determined, in part, by where it is stored. You can create a screen and store it:

- In a library where you and other developers can conveniently store many screens and access them at runtime.
- In a repository where it can function as a template or as a source of inheritance for objects used throughout your application.

This chapter describes:

- The different means provided to create a screen, ensuring that it is stored in the appropriate place on your system.
- General design considerations when creating and editing a screen with the screen editor.
- How you can document the contents of a screen.

When you design the interface to your application, there are at least two considerations to keep in mind:

- Portability from character-mode Panther to a GUI environment and vice versa, from a GUI environment to a web environment, and from one GUI environment to another. This can affect size, colors, and GUI-specific controls as well as other issues.
- Maintainability.

---

# Creating, Opening, and Saving Screens

---

Your Panther screens are created in the editor and then stored in a library or a repository. An screen can be:

- A client screen—Defines the user interface.
- A repository entry—Allows for an inheritance link when you use the repository screen and its objects to create your application screens.

You can create a screen by:

- Opening a new, untitled screen in the workspace.
- Opening an existing screen in a library and saving it with a new name.
- Opening an existing screen in a library as a template.

You can populate a screen with widgets by:

- Using the screen wizard.
- Creating new widgets.
- Copying widgets from one screen to another.
- Copying widgets from a repository.

## Creating a New Screen

A new, untitled screen is open by default when you start up the editor. This screen can be a client screen or a screen in a repository depending on where you choose to store it. Now you can begin populating the screen with widgets, or defining the screen's properties.

**Note:** As of Panther 4.5, a special type of screen, a frameset, can contain multiple screens. For more information, refer to Chapter 17, “Framesets and Splitters.”

## How to Create a New Screen

1. Choose File→New→Screen (or the New Screen button on the toolbar). The New Screen Tool dialog box appears and prompts:



2. Choose how you want the screen to be created:
  - Yes—Panther guides you through the creation of a screen using imported database tables in a repository (a repository must be open). Refer to Chapter 4, “Screen Wizard,” for details on using the screen wizard.
  - No—An empty, untitled screen opens in the screen editor workspace.
  - Cancel—The new screen operation is cancelled.

## Identifying Screens

Since you can have different types of application components open in the editor at the same time, you can identify screens in the following ways:

- The Screen Type property. Under Identity, the Screen Type property is set to Screen.
- The Type field in the Properties window. If a screen has focus (by clicking on a blank area of the screen), the Type field displays *Screen*.
- The screen's Title bar.
  - For newly created screens, the title bar displays *Untitled#*.
  - For screens stored in libraries, the title bar displays *screenName@libraryName*.

- For screens stored in repositories, the title bar displays `screenName@[Repository]`.

## Creating a Screen from a Repository

You can create screens that are derived from your database by following the procedure for importing database tables to a repository (refer to “Populating a Repository with Database Objects” [on page E-25](#)). Once you import the database tables, you can use the resulting repository entries to create your application screens.

### How to Add Objects to Your Screen from a Repository

1. Choose File→Open→Repository if the repository is not open.
2. To open a repository entry, do either of the following:
  - Choose View→Repository TOC. Select the desired repository entry from the Repository (TOC) and choose Open.
  - Choose File→Open→Repository Entry. Select the desired repository screen from the Open Repository Entry dialog box. Choose OK.
3. Select the desired widget or widgets on the repository screen and drag them to your application screen.

The copied (child) object in your application screen has property settings that match its parent in the repository. Inherited property values are indicated in reverse video in the Properties window. The position (Start Row, Start Column, End Row, and End Column properties) are not inherited; and although the initial value for the Name and Inherit From properties are derived from the parent object, there is no inheritance relationship for these property settings.

The Inherit From property for the child widget indicates the source of inheritance—the name defined for the parent object and its repository entry in the format `repository_entry!widget_name`. Refer to “Controlling Inheritance” [on page E-29](#) for ways of controlling inheritance.

## Creating a Dialog Box

A screen that has the Dialog property set:

- Cannot be resized, maximized, or minimized—At runtime, these options are not available on the screen's system menu, and the border (specifically in GUIs) has no maximize/minimize buttons. However, it can be moved.
- Is modal—At runtime, the user is forced to enter data, respond to, or acknowledge the dialog box before proceeding with the application. The menu bar and controls strings outside of the dialog box cannot be accessed while the modal dialog box is active.

**Note:** Dialog screens are not available in web applications.

For additional information about window styles in Windows applications, refer to [page 6-25](#), “Specifying Styles under Windows.”

## How to Define a Screen as a Dialog Box

1. Select the screen.
2. Under Identity, set the Dialog property to Yes.

The screen, in application/test mode, is now an application modal window. Consider providing some mechanism, like an OK button, to allow the user to continue or exit the screen.

For instructions on creating a tab dialog in Windows applications, refer to [page 16-5](#), “Creating a Tab Dialog Screen.”

## Opening and Saving Screens

For information about opening screens, refer to [page E-4](#), “Opening Application Components.”

For information about saving screens to a library, refer to [page E-16](#), “Saving an Application Component.”

For information about saving screens to a repository, refer to [page E-23](#), “Saving Application Components to a Repository.”

---

# Resizing a Screen

---

By default, new screens have a predefined Height and Width property setting—18 (grid rows) by 48 (grid columns). These properties define the initial dimension of a screen.

## Defining the Screen Size

Since Panther screens are virtual screens, you can create screens as large as 254 lines by 254 columns. When a virtual screen is larger than the physical display can accommodate, then the screen is viewed through a mechanism called a *viewport*.

## How to Change the Size of the Current Screen

Do either of the following:

- Drag the screen edge or corner to the desired dimension.
- Set explicit dimensions in the Geometry properties, Height and Width.

The grid size and the size of the font can have an effect on screen size as well.

## Conventions for Controlling Screen Size at Runtime

You can control the initial size of the screen as well as define what users can do to resize the screen.

## Changing the Viewport

### How to Control Whether Users Can Resize a Screen at Runtime

1. Select the screen.



2. Under Geometry, set the `Resizable` property:
  - **Yes**—(default) The user can change the size of the screen by dragging on its corner or edge. Vertical and horizontal scroll bars are automatically displayed if the viewport is not large enough to display the screen's entire contents.

In addition, when a user changes the size of a screen, you can control how widgets will grow or shrink to take advantage of the screen's new size. The screen editor Properties window is a good example of how widgets can resize and move depending on the window's size. Refer to “Controlling a Widget's Size at Runtime” on [page 9-9](#) for details on controlling widget size.

- **No**—The initialize size of the screen is maintained. At runtime, the user cannot resize the screen, and the screen's system menu `Size` option (if applicable to the platform) is not available.

Panther provides a mechanism for resizing widgets based on the size of the screen, but you can also write a function that Panther calls when the viewport size is changed. The function can control how widgets, under a GUI platform, are resized and rearranged when the user changes the viewport size.

## How to Attach a Custom Resize Function

1. Select the screen that requires the function.
2. Under Geometry, specify the function name in the `Resize Function` property.

## Maximize and Minimize

In a GUI environment, you can control whether the standard maximize and minimize buttons are available to the user. In addition, you can define whether a screen starts off in a maximized or minimized (iconified) state. By default, when a screen is invoked, it starts up in the size you specify at design time.

At runtime, a user restores a window to its original size by:

- Double-clicking on the minimized icon to reopen the window.
- Choosing the `Restore` button in the upper-right corner of the screen, or choosing `Restore` from the screen's system menu to restore a maximized window to its previous size.

## How to Define a Screen's Startup State

1. Select the screen.
2. Under Geometry, set the Startup property to the desired setting:
  - **Maximized**—Invokes the screen to occupy the entire display (entire MDI frame in Windows) when the screen is first displayed.

**Note:** The Maximized value has no effect under Motif.

  - **Iconified**—Invokes the screen in a minimized, iconified state. A default operating system icon (an empty blank square box in Windows) is used if you have not set the Icon property (under Display) or if the specified pixmap in the Icon property cannot be found.
  - **Normal**—Invokes the screen in the application's current state of maximization.

## How to Let Users Maximize and Minimize a Screen

1. Select the screen.
2. Under Geometry, set the Max/Min property under Geometry to one of the following settings:

**Note:** While in Edit mode in the screen editor, all screens have a minimize button only, regardless of the property setting. This allows you to iconify screens while you are designing. In Test mode, screens reflect the actual property specifications.

  - **Maximizeable**—Provides a maximize button on the screen's title bar, and the Maximize option on the screen's system menu. This setting prevents the user from minimizing or iconifying the screen at runtime; there is no minimize button, and the Minimize option on the system menu is unavailable.
  - **Minimizeable**—Provides a minimize button on the screen's title bar, and the Minimize option on the screen's system menu. This setting prevents the user from maximizing the screen at runtime; there is no maximize button and the Maximize option on the system menu is unavailable.
  - **Both (default)**—Provides both maximize and minimize buttons, and both options are available on the screen's system menu. Only the minimize button appears while you are in Edit mode.

- Neither—Prevents the screen from being maximized or minimized by removing the maximize and minimize buttons; both options are unavailable on the screen's system menu. The minimize button remains on the screen while you are in Edit mode.

**Note:** If the Title Bar property is set to No, maximize and minimize buttons are not available nor is the system menu.

## Iconifying a Screen

On a GUI platform, when a screen is minimized at runtime, a default operating system icon is provided by Motif, and Windows uses an empty square. You can provide your own icon that is displayed when the screen is minimized.

You can browse libraries with the Open button. Windows only supports \*.ico formats for icon usage. Motif will display BMP (.bmp), GIF (.gif), JPEG (.jpg), XPM (.xpm, .xpm1, .xpm3) and XBM (.xbm) style files.

**Note:** Motif does not scale images, so if they are too large for the icon space, the icon image may appear truncated.

## How to Assign an Icon To Display on a Minimized Screen

1. Select the screen.
2. Under Display in the Icon property, enter the filename or choose More to select the file from the Select Library Member dialog box. Choose OK.

In addition, the Select Library Member dialog box provides the capability to browse libraries with the Open button, and to add icon image files to a library with the Add button. If you choose the Add button, the Select File dialog box opens and you can select the icon file that you want to include in a library.

3. To see how it looks, choose File→Test Mode (or the Test Mode button on the toolbar).
4. Choose the Minimize button on the upper right corner of the screen's title bar.

The icon displays if the file is in one of your open libraries.

To facilitate portability and provide maximum flexibility, omit the filename extension when you specify an icon file. At runtime, Panther searches for files

in all open libraries using all possible image extensions supported on the local platform.

If you have icon images of the same name but of different types (for example, `logo.ico` is the Windows icon format, while `logo.jpg` is a photograph), include the extension to ensure that the correct image is displayed at runtime.

**Note:** Under Windows, if you compile the icon into your resource-definition file, the resource id must match the Icon property setting exactly.

---

## Using the Screen Grid

---

Panther uses a grid to determine widget location. The height and width of the entire screen is measured in grid units, expressed in grid rows and grid columns, respectively.

**Notes:** The horizontal grid marks indicate different units of measurement: in character mode, the space between each grid mark represents two grid units; in GUI modes, it represents one unit.

In character-mode Panther, the grid is fixed; each character occupies one grid cell.

If you explicitly set the Grid Height and Grid Width properties, under a GUI platform, Panther uses these; otherwise, it uses the screen's font size to determine the grid size (refer to “Specifying Fonts” [on page 10-7](#) for more information on setting font preferences).

To toggle the display of the grid, choose Options→Grid (or the Grid button on the toolbar).

## How to Control the Size of the Positioning Grid

Select the screen. Under Geometry in the Grid Height and/or Grid Width properties, enter the desired measurement (default is grid units). (Refer to Table 9-1 [on page 9-7](#) for a list of valid units of measurement.)

This setting overrides the screen's font size as a determining factor for sizing the positioning grid.

---

## Controlling Screen Location

---

When invoking a screen, you can specify parameters that control the screen's location and size. If you omit these parameters, Panther uses its own defaults. By default, Panther displays the screen in its entirety. When you invoke a screen:

- As a form, Panther displays it at the physical display's upper-left corner; this excludes the menu bar, which remains visible.
- As a window, Panther tries to leave the calling screen's last cursor position visible.
- If the screen size exceeds the actual display area, Panther creates a viewport. By default, the viewport's upper left corner (1,1) initially displays the screen's upper-left contents, unless this prevents display of the cursor. Panther ensures that the cursor's initial position in a viewport—usually the first field—is visible. If necessary, it adjusts the screen offset within the viewport accordingly.

Refer to “Displaying Screens” [on page 18-3](#) in *Application Development Guide* for detailed instructions on specifying screen position in control strings or runtime library functions.

---

# Controlling Geometry Changes Across Platforms

---

Panther recalculates a screen's size and the positions of its widgets when you open a screen in an environment that is different from the one in which it was created—for example, a screen created in character mode that is opened in Motif. When this happens, Panther uses a positioning algorithm that makes several assumptions about screen size and the correct placement of widgets. By default, these assumptions include maximizing the use of whitespace while maintaining the relative positions of widgets to each other.

The screen's Positioning properties let you:

- Control spacing between widgets.
- Set a minimum margin within the screen's border.
- Control adjustments in screen size and widget positions.

**Note:** Positioning properties are recognized only in GUI environments.

## Spacing Widgets for Portability

### How to Control the Amount of Space Between Widgets

1. Select the screen.
2. Under Geometry, in the Positioning subheading, set either or both of the following properties to the desired unit (default is grid units) of measurement required between widgets on the screen. (Refer to Table 9-1 [on page 9-7](#) for a list of valid units of measurement.)
  - Min Horiz Space—Determines the minimum amount of space maintained between widgets that are horizontally contiguous. If you place widgets closer than the distance specified by this property, Panther does not try to

increase the space between them. When no value (default) is specified, widgets are allowed to touch.

- **Min Vert Space**—Determines the minimum amount of space maintained between widgets that are vertically contiguous. If you place widgets closer than the distance specified by this property, Panther does not try to increase the space between them. When no value (default) is specified, widgets are allowed to touch.

If all available whitespace is used up between widgets, Panther moves them or adjusts the screen size in order to maintain the minimum spacing required.

## Setting the Screen Margin

### How to Control How Much Whitespace Exists Between the Screen's Inner Border and Its Contents

1. Select the screen.
2. Under **Geometry**, in the **Positioning** subheading, set the **Region Margin** property to the amount of space (grid unit is the default) desired between the border and the widgets on the screen. (Refer to Table 9-1 [on page 9-7](#) for possible units of measurement.)

**Note:** The default 0 allows widgets to touch the screen border. If you place widgets within the margin, Panther does not try to increase the space between them and the screen's border.

## Controlling Widget Positions and Screen Size

When you open a screen in an environment different from the one in which it was created, Panther attempts to minimize differences between the presentation of the same screen on different platforms. Panther typically adjusts the widget's start positions and the screen's dimensions by shrinking or expanding the grid.

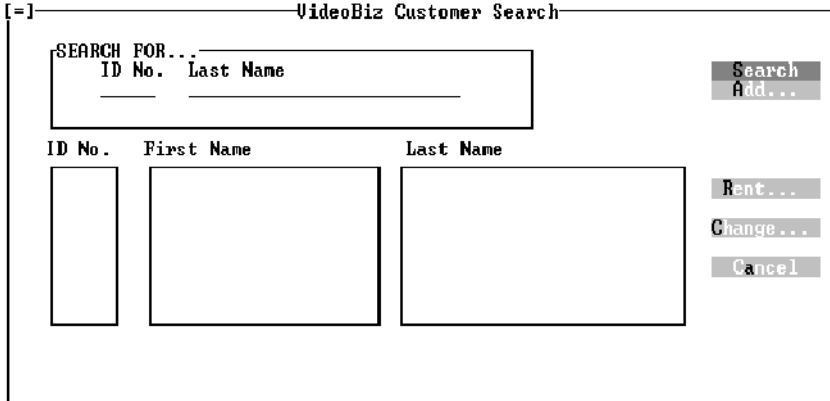
### How to Control How Widgets Move and the Screen Resizes

1. Select the screen.

2. Under Geometry, in the Positioning subheading, set the Horizontal and/or Vertical Shrinking properties to one of the following:
  - Decrease region size—Adjusts widget positions and screen size according to changes in the font and point size. This is the default setting.
  - Keep region size—Keeps the screen size unchanged. Panther adjusts widget start positions to simulate their original proximity; when moving from character to GUI modes, this is likely to leave extra whitespace at the screen's borders.
  - Prevent grid shrinking—Keeps the screen size and widget start positions unchanged. When moving from character mode to GUI modes, the whitespace between widgets is liable to increase.

## Example of Vertical and Horizontal Shrinking Properties

To illustrate how these settings work, three versions of the following screen were created in character mode. Each has different vertical and horizontal shrinking properties settings. In character mode, all versions appear to be identical (Figure 6-1).



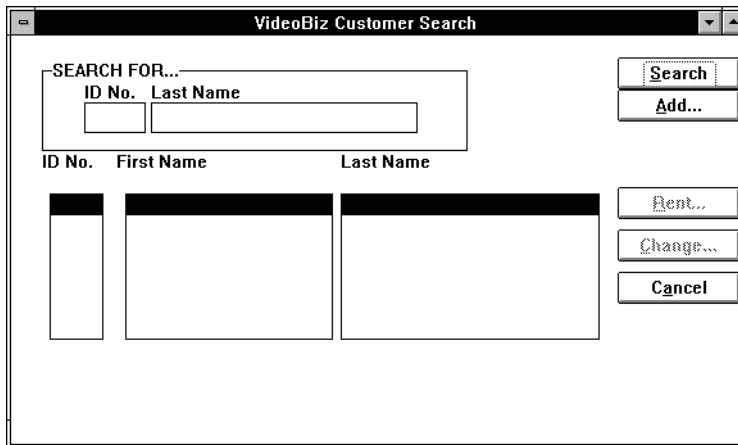
**Figure 6-1** Screen created in character mode.

When you port these screens to Windows, however, the differences become evident.

- Screen with Horizontal and Vertical Shrinking properties set to Decrease region size—Because the default screen font is proportional in Windows, the widgets require fewer columns and rows. Panther adjusts the widgets' start positions

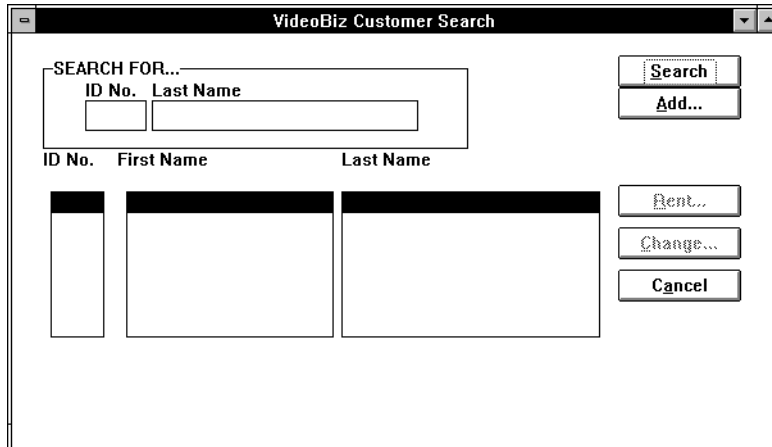


accordingly and shrinks the screen size to remove the extra whitespace that would otherwise be left over (Figure 6-2).



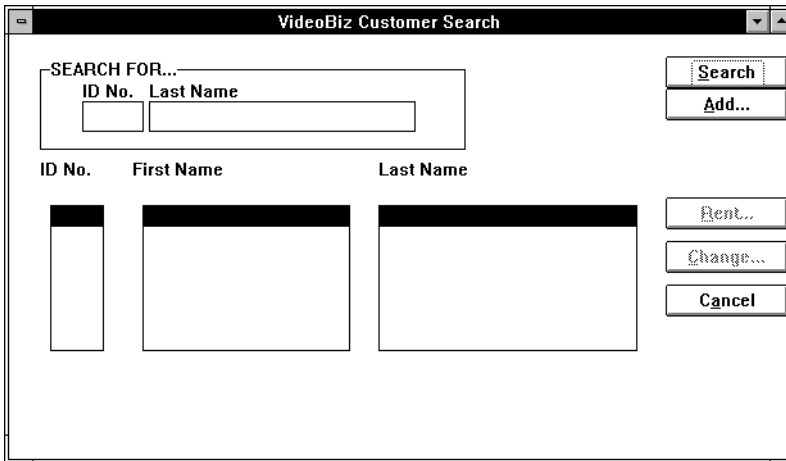
**Figure 6-2** Vertical and Horizontal shrinking properties were set to Decrease region size for this screen ported to Windows.

- Screen with Horizontal and Vertical Shrinking properties set to Keep region size—Panther repositions the widgets in order to maintain their relative proximity. However, the screen size remains unchanged, so extra whitespace is left in the screen's bottom and right margins (Figure 6-3).



**Figure 6-3** Vertical and Horizontal Shrinking properties were set to **Keep region size** for this screen ported to Windows.

- Screen with Vertical and Horizontal Shrinking properties set to Prevent grid shrinking—The screen size and widget positions remain the same. Because widgets occupy less space, the space between them is greater. Notice in Figure 6-4 that Panther repositions the widgets within the Search For box. Grid shrinking can occur within a box independently of the screen because a box is an autonomous positioning region with its own vertical and horizontal property settings—in this case, set to Decrease region size. For more information about using boxes as positioning regions, refer to [page 21-6](#), “Using Boxes as Positioning Regions.”



**Figure 6-4** Vertical and Horizontal Shrinking properties were set to Prevent grid shrinking for this screen ported to Windows.

Refer to “Controlling a Widget’s Position at Runtime” on page 9-18 for information on controlling widget position when a screen resizes.

---

## Including Screen Wallpaper

---

Instead of displaying a screen background color, you can display a picture, drawing, or photograph as the screen background. This is referred to as *wallpaper*.

Panther supports BMP (.bmp), GIF (.gif), and JPEG (.jpg) files under Windows and Motif as well as XPM (.xpm, .xpm1, .xpm3) and XBM (.xbm) files under Motif. Images appear in the size in which they were created.

### How to Assign a Wallpaper as a Screen Background

1. Select the screen.

2. Choose Options→Grid or toggle the Grid button on the toolbar to turn the grid off.

**Note:** The wallpaper does not display in Edit mode when the grid is on.

3. Under Display in the Wallpaper Pixmap property, enter the filename or choose More to select a file in a specific library from the Select Library Member dialog box. Choose OK.

The image displays immediately if the file is in one of your open libraries.

In addition, the Select Library Member dialog box provides the capability to browse libraries with the Open button, and to add image files on the system to a library with the Add button. If you choose the Add button, the Select File dialog box appears and allows you to select the files that you want to include in a library.

To facilitate portability and provide maximum flexibility, omit the filename extension when you specify an image file. At runtime, Panther searches for files in all open libraries using all possible image extensions supported on the local platform.

If you have images of the same name but of different types (for example, `dog.xpm` might be a bitmapped image while `dog.jpg` is a photograph), include the extension to ensure that the correct image is displayed at runtime.

**Note:** Under Windows, if you compile the image into your resource-definition file, the resource id must match the Wallpaper Pixmap property setting exactly.

4. Under the Wallpaper Pixmap property, set the Style subproperty to the desired position of the wallpaper image on the screen:
  - Center—Positions the image on the center of the selected screen. If the image is smaller than the screen dimensions, whitespace appears around the picture. If the image is larger than the screen dimensions allow, the image is cropped equally on all borders.
  - Tile—(default) The image appears in the upper left corner of the selected screen. If the image size is smaller than the screen dimensions, the picture is repeated as many times as necessary to fill the window. If the image size is larger than the screen dimensions, the right and bottom borders of the picture are cropped.

## How to Specify a Wallpaper under Windows

To specify a wallpaper for Panther's MDI parent window in Windows applications, set the appropriate variables in your Panther initialization (`*.ini`) file. For details on settings in the initialization file, refer to `MDIWallpaperPixmap` on page 3-5 in *Configuration Guide*.

---

## Specifying a Mouse Pointer

---

In general, the mouse is the primary method for navigating in GUI applications. For the most part, the native window manager handles the way your mouse pointer looks and behaves. In Motif, the default pointer shape is an arrow, but can be specified in the `pointerShape` resource; in Windows, the default cursor is also an arrow. These are considered application-wide pointers. However, you can also specify a mouse pointer shape for a screen that is different from the default.

In addition, you can use the library function `sm_delay_cursor` to change mouse pointer shapes at runtime.

## How to Define a Unique Cursor Shape for a Screen

1. Select the screen.
2. Under Display, enter the name of the GUI-specific cursor shape in the Pointer property. Refer to Table 6-1 for Motif specifications and to Table 6-2 for Windows specifications.

**Note:** Motif and Windows each have pointer shape specifications that are unique to their environments. Therefore, they are not portable across platforms.

**Table 6-1 Pointer shape specifications for Motif\***

---

arrow	icon
-------	------

**Table 6-1 Pointer shape specifications for Motif\***

based_arrow_down	left_ptr
boat	leftbutton
center_ptr	question_arrow
clock	sb_up_arrow
coffee_mug	spider
cross	spraycan
dot	target
double_arrow	top_left_corner
fleur	umbrella
hand1	watch

\*The entire list can be found in the file  
/usr/include/X11/cursorfont.h (the XC\_  
prefix is not required in the Pointer property setting).

**Table 6-2 Pointer shape specifications for Windows**

IDC_APPSTARTING	IDC_SIZE
IDC_ARROW	IDC_SIZEALL
IDC_CROSS	IDC_SIZENESW
IDC_HAND	IDC_SIZENS
IDC_HELP	IDC_SIZENWSE
IDC_IBEAM	IDC_SIZEWE
IDC_ICON	IDC_UPARROW
IDC_NO	IDC_WAIT

## Custom Pointer Shapes

In Windows, you can create your own cursor shapes and designate your custom cursor in the Pointer property for the screen.

### How to Create and Specify a Custom Pointer Shape

1. Create the cursor shape using any image/bitmap editor.
2. Define the image in your resource-definition file (.rc file) using the following format:

```
pointer CURSOR image_file
```

where *pointer* is a name you devise to identify the image; *image\_file* is the bitmap filename of the image you created.

3. Recompile to include the new definition.
4. In the screen editor, open and select the desired screen.
5. Under Display, specify your *pointer* in the Pointer property.

---

## Including Borders and Decorations

---

By default, a screen has a border and is fully decorated with an area for the screen title (title bar), minimize and maximize buttons (in GUIs only), and a system menu. Scroll bars appear in the border only if they are necessary, that is, if the screen is larger than can be displayed in the physical monitor or you choose to display the screen as a viewport.

In addition to designing screens with and without borders, you can control what appears on a border and control the title bar components. For example:

- Display the screen's title in the title bar.
- Suppress the display of the title bar all together.

- Display the system menu and specify its content.

## Designing the Screen Borders

Borders are included, by default, on all newly created screens. Borders can add visual interest and help distinguish the edges of a screen when it is opened on top of another screen of the same color. This is particularly useful in character mode applications. Borderless screens, on the other hand, have no title bar, system menu, or minimize and maximize buttons.

### How to Define a Screen's Border Display

1. Select the screen.
2. Under Display, set the Border property.
  - Yes—(default). All newly created screens have a border.
  - No—At runtime, the screen is borderless and has no resize handles. The screen's title bar does not display, nor does the system menu, minimize and maximize buttons. The screen cannot be moved, resized, minimized, or maximized with a mouse; however, shortcut keys can still perform these actions.

**Note:** The Border property and its subproperties are ignored on Windows applications.

### How to Define the Border Style for Character Mode Applications

1. Select the screen.
2. Under Display, set the Border property to Yes. Additional border properties are displayed.
3. Set the Border Style property to the desired style; styles are numbered 0 through 9 (style 1 is the default). The designated style will surround the screen.

**Note:** The numeric styles correspond to characters and are defined in your video file. You can modify them in the video file to control the actual character set associated with a particular border style. (For more information, refer to “Borders and Line Drawing” [on page 7-44](#) in *Configuration Guide*.)



4. Set the FG (foreground) and/or BG Color Type properties for the Border. In addition, you can assign other display attributes to the screen border—such as intensity, reverse video, and blinking. (Refer to “Setting Display Attributes” on [page 11-6](#) for detailed information on setting screen border colors and attributes.)

## Displaying a Screen Title

While you are working in the editor, the title bar for your application screen displays the filename and library you specified when you saved the screen, or, when unsaved, it is `Untitled` followed by a number. However, to provide a title for your screen in application mode or test mode, you need to set the Title property.

### How to Display a Title in the Screen's Border/Title Bar

1. Select the screen.
2. Under Identity, enter the screen title in the Title property.  
In application/test mode, the title is displayed in the screen's border or title bar.
3. To suppress the display of a title in the title bar, leave the Title property value blank.

**Note:** The Title property has no effect in character mode if the Border property is set to No. In the GUIs, there is no effect if the Title Bar property is set to No.

## Assigning a Mnemonic

Identifying a mnemonic for screens running in character mode provides users with an alternative way, and GUI-like means, of bringing focus to a screen. At runtime, the user can bring focus to a screen by choosing the Windows menu bar option, and then selecting the desired screen from the drop-down menu by typing the mnemonic that is highlighted. The mnemonic is designated as one of the characters in the screen's title.

### How to Identify the Keyboard Mnemonic for the Screen

1. Select the screen.
2. Under Identity, enter a screen title in the Title property.

3. Under Identity, in the Mnemonic Position property, enter a letter or a number corresponding to the letter's position in the screen's title.

For example, if the screen's title is `Customer Information`, you can specify any of its letters as the mnemonic. Enter an upper case `C` (or `1`), or lower case `u` (or `2`). Uppercase `C` or the number `1` defines the first letter in the string to be the mnemonic, while lowercase `u` or the number `2` defines the second letter in title to be the mnemonic.

The property setting displays the position of the character followed by the character. For example, if you enter the mnemonic `C` for the `Customer Information` screen, the Mnemonic Position property displays `1 `C'`—the first character in the title.

## Displaying a Title Bar

All newly created screens have a title bar. The title bar includes the screen's title, the system menu in the upper left corner, and minimize and maximize buttons in the upper right corner.

**Note:** The Title Bar property has no effect in character mode if the Border property is set to `No`.

## How to Define the Display of the Screen's Title Bar

1. Select the screen.
2. Under Display, set the Title Bar property.
  - **Yes**—(default) All features and decorations as well as the screen's title are displayed on the screen's title bar, unless other title bar-related properties have been turned off.
  - **No**—The screen's title bar and, therefore, its title are not displayed. In addition, the following title bar features are also suppressed:
    - Minimize and maximize buttons (on GUIs).
    - System menu.
    - At runtime, the screen cannot be moved, minimized, maximized, or closed. Consider providing alternative user options, such as a Cancel button, OK button, or some other mechanism for allowing the user to continue.

## Displaying a System Menu

A screen's system (or control) menu is displayed in the upper left corner of the screen border (displays as [=] in character mode). By default, all newly created screens have a system menu. In general, the system menu includes the following options: Restore, Move, Size, Minimize, Maximize, and Close.

**Note:** In Motif, you can define the contents of your system menu in the `.mwmrc` file.

Most options on the system menu have other means of access, such as resize handles, and maximize and minimize buttons.

**Note:** The System Menu property has no effect if Border is set to No in character mode or Title Bar is set to No

## How to Prevent Users from Closing a Screen from the System Menu

1. Select the screen.
2. Under Display, set the Close Item property to No.

**Note:** This property setting is particularly useful on dialog boxes where user input may be required before proceeding.

## How to Suppress the Display of a System Menu

1. Select the screen.
2. Under Display, set the System Menu property to No.

---

# Specifying Styles under Windows

---

As of Panther 4.5, there are new options for opening windows under the Windows operating system. Previously windows could either be opened as MDI windows, or as dialogs. An MDI window cannot be moved outside the MDI frame. A dialog can be

moved outside the MDI frame, but blocks access to the MDI frame – when a dialog is opened, focus cannot be given to any MDI windows, nor can the MDI menu be accessed until the dialog is closed.

The new options are implemented by the properties **Keep in Frame** and **Topmost**. These are screen-level properties and are found under the **Identity** section in the **Properties** window.

## The Keep In Frame Property

The **Keep in Frame** property is a subproperty of the **Dialog** property. If a window is specified as a dialog, the new option doesn't apply.

The default value for **Keep in Frame** is **Yes**. A non-dialog window with **Keep In Frame** set to **Yes** will behave exactly like the MDI windows Panther has always supported. Setting **Keep In Frame** to **No** will make the screen open as a non-MDI window, that is not a modal dialog. Such a screen can be moved outside the MDI frame, but does not block access to the MDI windows nor to the MDIs menu bar.

When **Keep in Frame** is set to **No**, two further properties appear in the **Properties** window. These are **Keep On Top** and **Parent Window**. The **Parent Window** property is relevant only to windows that have **Keep in Frame** set to **No**, and **Keep On Top** also set to **No**. The **Parent Window** property will not appear in the **Properties** window if the **Keep On Top** property is set to **Yes**.

Unlike dialogs, screens that have **Keep in Frame** set to **No** can be minimized or maximized. The behavior when they are minimized is determined by the **Parent Window** property.

Windows outside the MDI frame do not appear in the **Windows** menu list in the standard MDI application menu. And they cannot be accessed by function keys like **Ctrl+F6**. They can, however be accessed by **Windows** shortcuts such as **Alt+F6**.

## The Parent Window Property

Windows that have the **Keep in Frame** property set to **No** and are not designated as **Topmost** windows can either have no parent or they can be parented by the MDI frame. If the parent is designated as **None** then the screen can be hidden by the MDI frame when the latter is given focus. If the screen is designated as parented by the MDI frame, then the screen will always appear on top of the MDI frame. In this case, even if focus

is given to the MDI frame, or one of the windows inside the MDI frame, the non-Keep in Frame window will always be displayed, and will not be obscured behind the MDI frame. Note that a non-Keep in Frame window whose parent is the MDI frame can still be hidden behind other windows that are not part of the Panther application. To designate that a window should never be obscured behind another window, whether that other window be part of the Panther application or not, use the Keep on Top property.

When a non-Keep In Frame window is minimized, the location of the icon representing it depends on the window's parent. A non-Keep In Frame window with no parent will be represented by an item on task bar. A non-Keep in frame window whose parent is the MDI frame will be represented by an icon in the desktop's lower left, just above the taskbar.

## The Keep On Top Property

A non-Keep in Frame window can be designated as a Topmost window by setting the Keep On Top property to Yes. A Topmost window will generally be rendered on top of any other window on the desktop, no matter which application is active. The only windows that can be placed on top of a Topmost window are other Topmost windows.

## Runtime Access to the Window Options Properties

At runtime the properties `PR_KEEP_IN_FRAME`, `PR_KEEP_ON_TOP`, and `PR_WINDOW_PARENT` are read only. Once a window has been opened, its status with regard to these settings cannot be changed.

---

# Attaching a Menu Bar

---

Menu bars provide users with a convenient method for accessing a variety of commands and functions.

- In character mode, the menu bar is displayed at the top of the physical screen.

- Under Motif, the menu bar/toolbar is displayed at the top of the base window.
- Under Windows, the menu bar/toolbar is displayed at the top of the MDI frame.

When you click on a menu option, a submenu drops down. Drop-down menus display items that can either invoke dialog boxes or carry out commands.

## How to Attach a Menu to a Screen

It is assumed that you have already created the menu and/or toolbar and that it resides in a menu script. For details on creating menu bars, refer to Chapter 25, “Menu Bar Editor.” For information on loading menu script files, refer to Chapter 15, “Including Menus and Toolbars,” in *Application Development Guide*.

1. Select the screen.
2. Under Focus, enter the name of the menu in the Menu Name property.

At runtime, Panther checks the screen-level location to see if the named menu is in the installed menu script, and then checks application-level. If the named menu is not in an installed menu script, proceed to step 3.

3. Under Focus, in the Menu Script File property, enter the filename or choose More to select the file (\* .menu) from the Select Library Member dialog box.

This specification loads the named script into this screen's memory. All other menus defined in this script are available for display with this screen.

If you do not supply a menu name in the Menu Name property, Panther displays the first menu in the specified script.

You can also load menu scripts and install menus by calling `sm_mnscript_load` and `sm_menu_install`, respectively. Refer to the *Programming Guide* for a description of these functions.

## Assigning a PopUp Menu

A popup menu can be opened when the user presses the right mouse button. By default, the screen's menu bar is displayed as the popup.

## How to Display a Popup Menu that is Different from the Screen's Menu Bar

1. Select the screen.
2. Under Help, enter the name of the menu in the Popup Menu property.

The menu must be in a script that is already loaded in memory. Panther checks the screen-level location first, and then application-level for the named menu. For details on creating menu bars, refer to Chapter 25, “Menu Bar Editor.” For information on loading menu script files, refer to Chapter 15, “Including Menus and Toolbars,” in *Application Development Guide*.

At runtime, the user sees the specified popup menu when the right mouse button is pressed anywhere in the screen area. However, if the mouse is over a field that has its own popup menu designation, the field's popup menu is displayed instead. For information on attaching a popup menu to a widget, refer to [page 8-6](#), “Invoking a Popup Menu.”

## Testing Menus

Once you attach the menu, you can test it with the screen.

1. Bring focus to the screen.
2. Choose File→Test Mode or choose the Test button on the toolbar.

The screen is displayed in Test mode with its menu bar.

---

# Documenting Screens

---

Two properties under Identity in the Properties window help you document the contents of your screens:

- Comments property—Provide a brief description of the screen.

- Memo Text property—Attach up to nine lines of text for additional comments or for programmatic use.

## Adding Comments

By including a brief description of your screen, you can provide information to other developers about the screen, its content, and its purpose. This description is displayed when you view the Repository TOC and scroll through each of the repository entries.

### How to Add or Edit Comments on Your Screen

1. Select the screen.
2. Under Identity, select the Comments property. The Comments dialog box opens.
3. Do one of the following to enter or edit text:
  - Type the text directly in the text area.
  - Choose Editor to access your local editor (as defined in the [SMEDITOR](#) variable).
  - Choose Read File to read in an external file located on your system.
  - Choose Save File to save the comments to an external file.
4. Choose OK to save the comments and return to the Properties window.

## Including Additional Information

### How to Include Additional Information on a Screen

1. Select the screen.
2. Under Identity, expand the Memo Text subheading. Nine memo lines are displayed.
3. Type additional information, comments, and specifications that you can't indicate or specify as a screen property.



At runtime, you can access the screen's Memo Text property to determine what action to take depending on the property's content. For information on accessing screen properties, refer to “Properties” on page 19-40 in *Application Development Guide*.

---

## Defining XML Tags for Screens

---

As of Panther 5, you can import and export XML data from a Panther screen. To use this feature, you must:

- Define properties for the screens and the widgets to be included in the XML data
- Call the appropriate functions to write data to the buffer or file holding the XML data

This section describes the property specifications for screens. For information on widget properties, refer to [page 8-31](#), “Defining XML Tags for Widgets.” For details on using the XML interface, refer to Chapter 22, “Using XML Data,” in the *Application Development Guide*.

### How to Set XML Properties for Screens

1. Select the screen.
2. Under XML, enter a value in the XML Tag property.

- *or*

Under XML, enter a value in *both* the XML Prefix and XML Postfix properties.

**Note:** If the XML Prefix property is empty, the generated XML begins with the following line:

```
<?xml version="1.0"?>
```

3. (Optional) If you entered an XML tag for the screen, you can enter additional attributes in the XML Attributes property.

---

## Testing Screens

---

For information on testing screens, refer to Chapter 38, “Testing Application Components,” in *Application Development Guide*.

# 7 Defining Service Components

For COM and EJB components and three-tier applications, you must create and deploy service components in addition to your screens. Even though all distributed application architectures use service components, each technology needs product-specific settings and defines services differently. For developers, this means that a service component created for one product will need modifications before it can work with another technology.

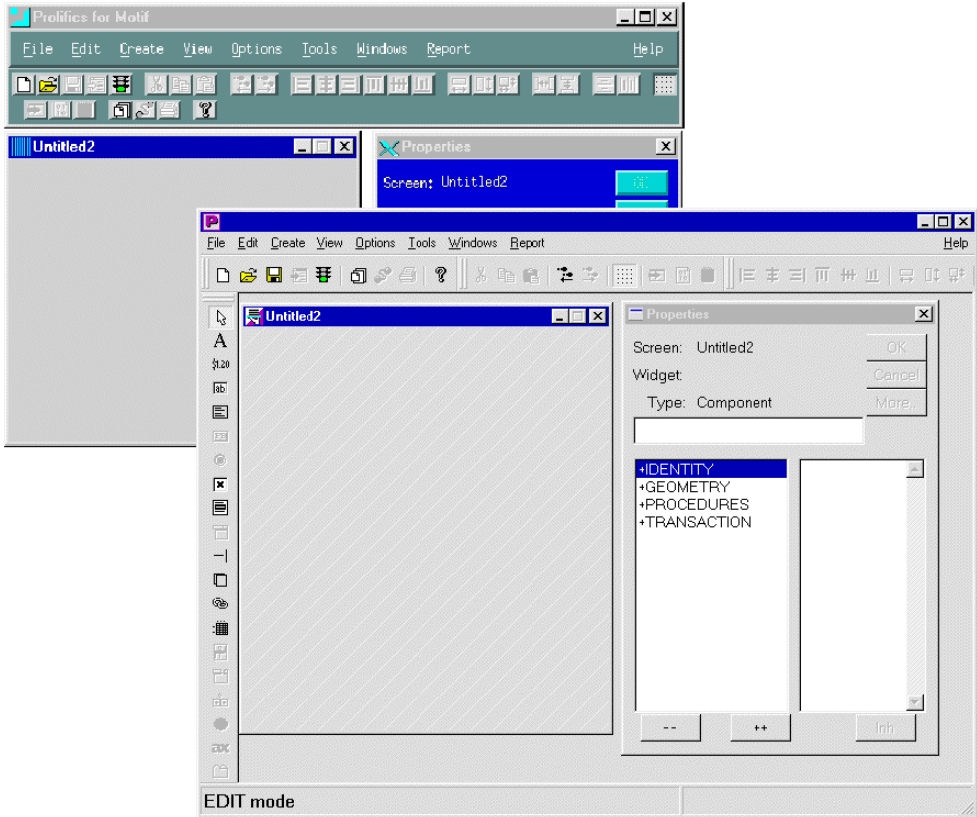
This chapter lists general instructions for creating and defining service components. Additional information can be found in the *Application Development Guide* as well as the manuals for each of the different products.

---

## Creating and Saving Service Components

---

When you create a new service component in the editor, it displays a window in which you define the widgets participating in the service and add programming code.



**Figure 7-1** In all environments, a service component has characteristics to distinguish it from a screen. Note the icon in the upper left corner of the window and the different wallpaper.

## How to Create a Service Component

1. Choose File→New→Service Component. An untitled service component opens.
2. Create the data members by placing widgets from the Create menu or Create tool bar onto the component.

3. Add code to the component. JPL procedures can be added by choosing Procedures→JPL Procedures.
4. For COM and EJB, the component interface must be defined. From the menu, choose View→Component Interface. This brings up a window where you can define methods and properties and specify product-specific settings.

## Creating with the Screen Wizard

In JetNet and Tuxedo applications, the screen wizard can generate service components at the same time that it is creating client screens. Each service component is identical to the client screen except that it has no push buttons and no menu/toolbar controls since there is no direct interaction between a user and a service component.

Services for the service component are defined for the root table view of the service component. For more information on wizard-generated output, refer to page 4-25, “Output for JetNet and Oracle Tuxedo Applications.”

## Identifying Service Components

Since you can have different types of windows open in the editor at the same time, you can identify service components in the following ways:

- The Screen Type property. Under Identity, the Screen Type property is set to Component.
- The Type field in the Properties window. If a component has focus, the Type field displays `Component`.
- The service component's icon and wallpaper.

## Opening and Saving Service Components

Additional information is available:

- For information about opening service components in a Panther library, refer to [page E-4](#), “Opening Application Components.”
- For information about saving service components to a library, refer to [page E-16](#), “Saving an Application Component.” Information specific to COM

components is in “Saving COM Components” on page 7-11 and to EJBs is in “Saving Service Components and Generating EJBs” on page 7-14.

---

## Defining the Component Interface

---

Part of creating the service component is defining the component interface, which is the set of methods and properties the component will support. From the menu, choose View→Component Interface. This brings up a window containing settings for methods and properties as well as the necessary settings for the specific technology.

**Note:** JetNet and Tuxedo applications do not use the Component Interface window.

### Defining Methods

Methods are functions or procedures available to the component. These procedures define actions to be performed when the methods are invoked, and they usually receive and/or return values.

The Methods section displays the methods defined for a particular component. Each row in the grid corresponds to a method. The first column shows the method's return type. The second column shows the method's name. The third column shows the method's parameters (in order, and prefixed by their kind and type).

A method can be implemented by a JPL procedure that is in scope when the form is opened at runtime or by a C function (the name of the JPL procedure or C function, however, *does* have to be the same as the component method's name).

Java processing is associated with the component by specifying the Java class in the service component's Java Tag property.

### How to Add or Change a Method

1. Choose Change.

The Change Methods dialog opens.

2. To add a method:

- In the New Method column, select the name of any existing component-level JPL procedure and use the arrow buttons to move items between the columns.
- In the New Method field at the bottom, type the name of a new method and press Enter.

The Add Parameters for Method dialog opens.

3. To change a method:

- Select an existing method and choose Copy or Modify or double-click.

The Copy Parameters for Method or the Change Parameters for Method dialog opens.

## How to Set the Method's Return Type

You set the return type on the Add, Change, or Copy Parameters for Method dialogs by selecting one of the following: Void, String, Integer, Boolean, Double, or Object.

## Parameters

Parameters are arguments that are passed between the client application and the service component. They may be passed only in one direction, or they may be passed back and forth.

## How to Add a Parameter to the Method

1. Enter the name of a new parameter in the New Parameter field and press Enter, or select a parameter from the Service Widgets column. You can also select an existing parameter and choose Modify or double-click it.

The Add Parameter or Change Parameter dialog opens.

2. Define the following:

- Type—Set to String, Integer, Boolean, Double, or Object.

- **Array**—Select the Array check box if applicable. Parameters may be arrays of data.
  - **Kind**—Set to In/Out, In, or Out, depending on whether the parameter is to be provided by the client, returned to the client, or if a value is to be sent and another returned in its place.
3. Choose OK.

## **How to Generate a Parameter List**

The Template button on the Add Parameters for the Method dialog writes template JPL for the method to the clipboard; it can then be pasted to another file.

## **Defining Properties**

Properties are variables that provide state information about the application.

The Properties section displays all the properties supported by the component. Each row in the grid corresponds to a Panther variable, either a widget on the component or a JPL variable. The order in which the properties appear does not affect the component's functionality.

The first column shows the property's type. The second column shows whether or not the property is read-only. The third column shows the property's name.



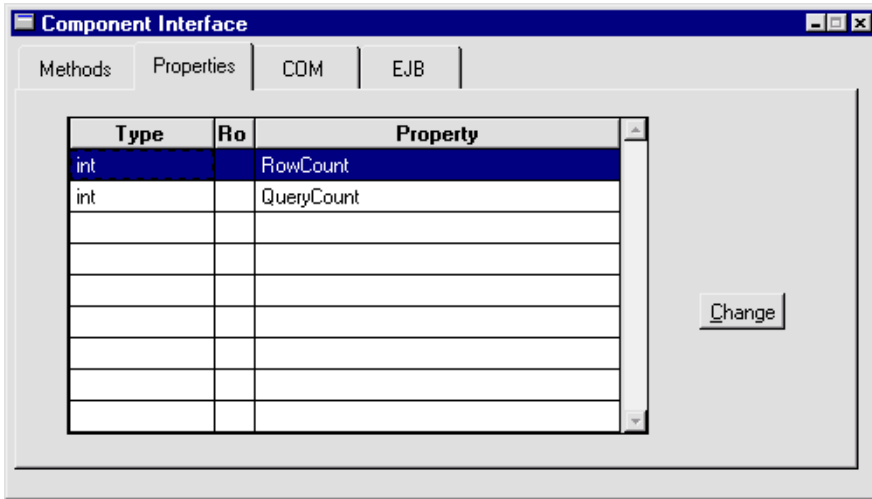
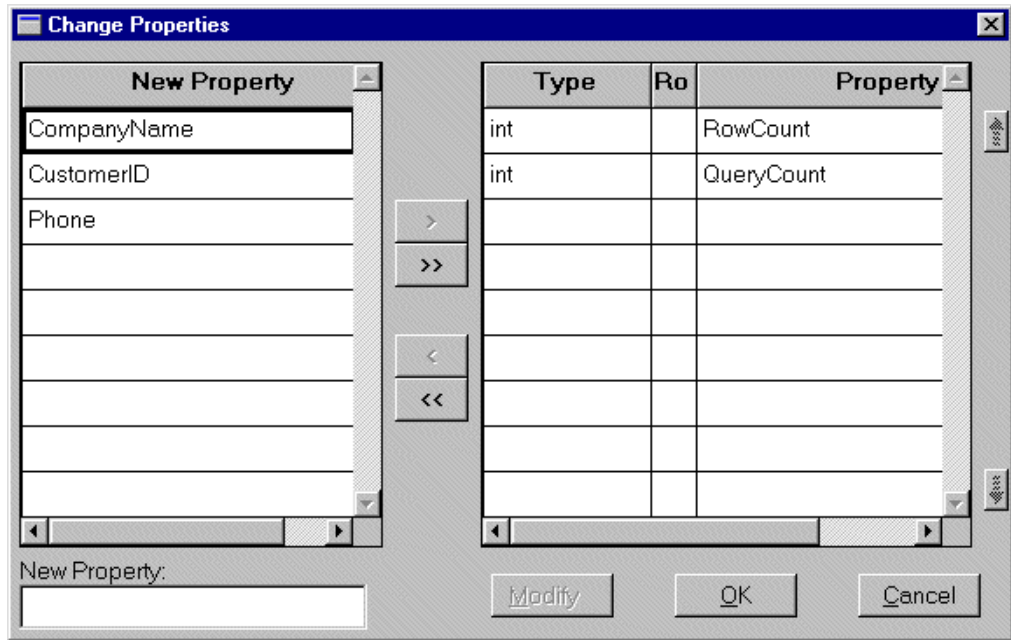


Figure 7-2 The Properties tab card displays type, name and read-only information for the properties of a component.

## How to Add or Modify a Property

1. Choose Change.

The Change Properties dialog opens.



**Figure 7-3** The Change Properties dialog allows you to add, insert or modify the property settings.

2. To add new properties:
  - In the New Property column, select an existing item and use the arrow buttons to move items between columns.
  - In the New Property field at the bottom, type the name of a new property and press Enter.  
The new property name appears in the grid.
3. To change the settings for a property, select the property in the grid and choose Modify or double-click on the property.  
The Change Properties dialog opens.
4. Define the following:
  - Type—Set to String, Integer, Boolean, Double, or Object.

The value in the field or JPL variable that implements the property will be converted to this type and handed to the client when a client requests the value of the property.

- Array—Select the Array check box if applicable.
- Read-only—Select the Read-only option. Check the box if applicable.

If the property is marked Read-only, clients can get its value but not set it.

---

## COM Applications

---

A Panther COM component consists of an implementation file (.dll) and a DCOM installation file (.inf). These all have the same base name as the service component. The DLL is generated from a template called `Pr1Server.dll` (in the `config` directory of the Panther installation directory). It is copied to the application directory and renamed to the name of the component. The application directory and template DLL can be changed in the COM section of the component interface screen.

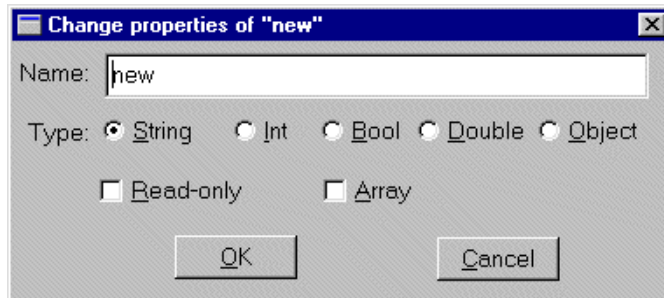
- When you save service components for COM, they all must be saved in the same directory. The Directory field allows you to specify the directory in which the .dll file will be stored.
- You have the option of specifying a version number. If you change the version number, the resulting component can be registered on a system along with the previous version. Client applications can call a specific version; if no version number is specified, the latest version will be accessed.
- Under Advanced Options, the `Pr1Server.dll` field allows you to use a .dll other than the default `Pr1Server.dll`. This allows you to add custom code by installing your own C functions. The source to `Pr1Server.dll` can be found in the `comlink` directory.
- Under Advanced Options, you have the option of changing the component's Class ID (CLSID). This identifier is automatically generated when the component is first created. You should change it only if you are generating a

new component from an existing one and they must have distinct identities. For example, the CLSID should be changed when changing the version.

## How to Generate a New CLSID

Press the New button on the COM tab card. You need to do this if:

- You modify an existing method or property of a distributed COM component. An interface is a contract between the component and the other software components that access it. If you modify the interface of a distributed component being used by other programs, you must generate a new Class ID. Adding a new method or property does not require generating a new Class ID, though updating the version number is recommended.
- You use one component as a template for another, and wish to save the component in the editor under a new name. Note that merely changing the name is not enough to define a new component. The Class ID is the unique identifier that COM uses to distinguish one component from another.
- If you change the version and want the old version to remain available, you should use a new CLSID for the new version.



**Figure 7-4** The COM tab card allows you to set DLL directories and to change the CLSID.

## Saving COM Components

To save a service component, choose File→Save As→Library Member, enter a name, and choose the library to save it in, generally `server.lib`. When you save a service component, you are asked whether to generate a new type library (`.tlb`) for it. If you are saving the component for the first time, or you have changed the interface definition for the component since the last time the component was saved, you should choose Yes. Otherwise, you can choose No.

You can view the contents of a type library using Microsoft tools such as COM/OLE Object Viewer (`oleview.exe`, part of Microsoft's Visual C++ compiler).

## Deploying COM Components

For more information on instantiating a COM component in a client screen, refer to Chapter 4, “Building Client Screens,” in *COM/MTS Guide*.

For more information on deploying COM components, refer to Chapter 5, “Deploying COM Components,” in *COM/MTS Guide*.

---

# Enterprise JavaBeans

---

A Panther Enterprise JavaBean consists of a series of Java classes, home and remote interfaces, a deployment descriptor and any environment settings needed for deployment. The Java classes are compiled and packaged in a JAR and then the EJB is installed with the WebSphere middleware so that it can be located in response to client requests.

The corresponding Panther service component must be located in one of the libraries specified in an application's `SMFLIBS` directory, or, by default, in `server.lib` in the server's base directory, if it is to be found at runtime.

The EJB's methods will be implemented either by JPL, Java or C code accessed from that form. To use C or C++ functions to implement an EJB's methods, the functions must be built into a shared library and that library must be loaded, and the relevant functions installed, using the functions `sm_slib_load` and `sm_slib_install`.

For more information on Panther EJBs, refer to *Panther for IBM WebSphere Developer's Studio*.

## The Panther EJB Tab Card

The Panther EJB card has three sections.

- **General**—Specify the directory into which the files for the component should be generated. Optionally, you can also specify a package name and a JNDI home name. Panther uses the service component name as the default JNDI home name.
- **Transaction**—Specify the transaction attributes for the bean. Optionally, you can also specify transactions for specific methods.
- **Environment**—Optionally, specify environment variables for the bean.

### General

#### How to Specify General Settings for the Bean

1. Specify the directory name for component file generation.
2. Optionally, specify the Java package name for the EJB's bean class, home interface, and remote interface if you wish the files to be compiled into a specific package.
3. Optionally, specify the Java Naming and Directory Interface (JNDI) name by which clients will invoke the EJB at runtime. Panther uses the service component name by default.

## Transaction

### How to Change the Transaction Attributes of the Bean

1. Choose the Transaction Attribute from the drop down menu.

- TX\_BEAN\_MANAGED
- TX\_MANDATORY
- TX\_NOT\_SUPPORTED
- TX\_REQUIRES\_NEW
- TX\_REQUIRED
- TX\_SUPPORTS

For descriptions of the attributes, refer to “Transaction Attribute Settings” on [page 5-14](#) in *Panther for IBM WebSphere Developer's Studio*.

2. Choose the Isolation Level from the drop down menu.

- SERIALIZABLE
- REPEATABLE\_READ
- READ\_COMMITTED
- READ\_UNCOMMITTED

For descriptions of the isolation levels, refer to “Isolation Level Settings” on [page 5-14](#) in *Panther for IBM WebSphere Developer's Studio*.

3. Optionally, double click on a specific method name to specify values other than the default for the transaction attribute and isolation level.

## Environment

### How to Add a New Environment Variable for a Bean

1. Specify Name and Value.
2. Choose Set.

## Saving Service Components and Generating EJBs

To save a service component, choose File→Save As→Library Member, enter a name, and choose the library to save it in, generally `server.lib`.

When you save a service component, you are asked if you wish to generate the files needed to deploy the component. You can choose to generate the EJB's Java files, and, in addition, to compile and package the EJB, depending on whether IBM WebSphere Application Server is installed locally or remotely.

You can also select Tools→Generate Component to generate the EJB.

For more information on building and generating EJBs, including a description of the files generated, refer to Chapter 5, “Building Enterprise JavaBeans,” in *Panther for IBM WebSphere Developer's Studio*.

## Deploying EJBs on IBM WebSphere Application Server

The specifics of deployment depend on whether or not IBM WebSphere Application Server is installed locally. Panther installs the EJB if the application server is available; if not, additional steps are necessary.

To call EJBs built with Panther, the deployed JAR file, built or added to when the EJB is packaged and compiled, needs to be on the `CLASSPATH` for the client. Users deploying applications must copy these JAR files to every machine that they wish to act as EJB clients.

For more information on deploying Panther EJBs, refer to Chapter 6, “Deploying Enterprise JavaBeans in WebSphere,” in *Panther for IBM WebSphere Developer's Studio*.



---

# JetNet and Oracle Tuxedo Applications

---

## Defining Services in JetNet and Oracle Tuxedo applications

Services in JetNet and Oracle Tuxedo applications are the subroutines necessary for an application to access a resource manager, such as a database. Service components for JetNet and Tuxedo are visual representations of these services. For more information, refer to Chapter 5, “Defining Services in JetNet and Oracle Tuxedo Applications,” in *JetNet/Oracle Tuxedo Guide*.

## Using Wizard-Generated Service Components

When you generate a client screen and a service component, they are identical because a service component interacts with its client screen counterpart to transfer data to and from the appropriate services. The client screen and service component pair must be kept parallel. Thus, if you modify your client screen, you must also modify its corresponding service component in a similar fashion. This is true of selection screens and their corresponding selection service components.

**Note:** Visual modifications (such as color, fonts, and pixmaps) on the client screen need not be duplicated on the service component.

## Testing Service Components

You can test a service component to see if the appropriate data is passed from the database to the service component. This requires a direct database connection rather than connecting to the database by way of the application server.

For more information on testing service components, particularly in JetNet and Oracle Tuxedo applications, refer to “Testing Screens and Service Components” [on page 38-5](#) in *Application Development Guide*.



# 8 Defining Widget Behavior

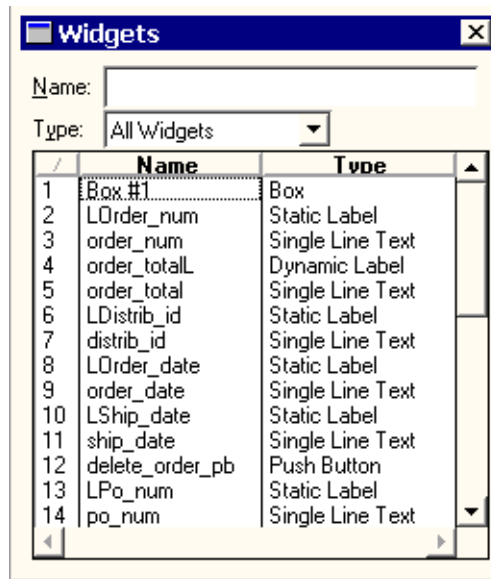
By default, Panther controls much of how widgets behave—each widget type having its own unique behavior. However, this chapter describes those attributes, via the Properties window, that allow you to control the way a widget behaves, such as:

- Using the widget's name to tell Panther which widget to act on.
- How to use mnemonics to provide users with an alternative method for focusing on a field.
- How to initialize a widget's state and protect its data so the user can't erase or overwrite information.
- Determining what happens when you double-click on a widget.
- Defining the tabbing order.
- How to synchronize scrolling arrays.
- How to set properties to generate XML

## Identifying and Naming Widgets

Panther automatically assigns internal field numbers whenever you add a widget to a screen. However, to maintain maximum control of what happens on a screen at runtime and facilitate field referencing in programming, assign a name to each widget on a screen—under Identity, in the Name property.

The widget's name appears at the top of the Properties window when the widget is selected. You can also see the name (and number) assignments for all widgets on a screen by viewing the [Widget List](#).



**Figure 8-1** Provide widgets with easy-to-identify names to facilitate referencing them programmatically.

Panther automatically numbers each object on a screen as follows:

- Widgets that allow data entry and all onscreen occurrences of arrays are numbered sequentially from left to right, and top to bottom—as `Field #1`, `Field #2`, and so on.
- Box and line widgets are numbered sequentially from left to right, and top to bottom—as `Box #1`, `Box #2` and `Line #1`, `Line #2`, respectively.
- Grid widget and graph widgets are number sequentially from left to right, and top to bottom—as `Grid #1`, `Grid #2` and `Graph #1`, `Graph #2`, respectively.
- Groups are numbered sequentially in the order in which they were created on the screen—as `group1`, `group2`, and so on.

**Note:** Only widgets that are assigned field numbers (`Field #n`) are accessible at runtime through the `field_num` property.

Referencing by field name is generally safer than referencing field numbers, because field numbers can change each time the screen is changed; for example, when a record is inserted.

## Assigning a Widget Name

A widget name identifies a field on the screen. The name must be unique; that is, no other widget or group on the screen can share the name. Individual array occurrences are referenced by widget name and occurrence number. Widgets that are created as the result of importing database tables are automatically named after the database column with which it is associated.

## How to Assign a Name to a Selected Widget

- Under **Identity**, enter a name in the **Name** property.

Names can be up to 31 characters long, and must start with an alphabetic character, an underscore, a dollar sign, or a period. The rest of the name can contain alphanumeric characters, underscores, dollar signs, and periods. The names are case-sensitive; therefore, a widget named `city` is different from a widget named `City`.

Use the widget name to refer to a widget both within the screen editor (e.g., to specify the **Next Tab Stop** property) and in application code. In addition, a widget must be named if:

- Its contents are to be shared with the LDB (local data block).
- Inheritance from a parent object in a repository entry is to be maintained—repository widgets must be named.

When you copy a widget within a screen, the copy retains all properties of the original except the name—otherwise a duplicate name would result. The new widget has no name.

When you copy a widget from one screen to another, the widget name is copied as well. Widgets on different screens can share the same name, but it is good practice to share names only when the widgets will share the same data or properties. In that way, data can be shared through the LDB, and properties can be inherited from the repository, respectively.

## Assigning a Mnemonic

You can identify which character on a widget's label is treated as the widget's mnemonic. The mnemonic provides users with an alternative way of activating or bringing focus to a field, or related field, if the widget is protected. The following rules apply when you assign a mnemonic:

- To a button (push button, check box, radio button, or toggle button)—The mnemonic can be used to “press” the widget via the keyboard.
- To a dynamic label widget—When the user types the mnemonic, the focus goes to the field specified in the label's Next Tab Stop property. If there is no tab stop specification, focus goes to the next field to the right of the dynamic label that can receive focus.

The mnemonic is designated as one of the characters in the widget's label. For example, if you have a check box on the screen with the label New Movies, you can specify any of its letters or a position as the mnemonic. You can specify upper case N or lower case w, or position 1 or 3, respectively. In this example, upper-case N defines the first letter in the string to be underlined (or displayed in a contrasting color in character mode), lower-case w defines the third letter in the word New to be designated as the mnemonic.

### How to Identify the Keyboard Mnemonic on a Widget

1. Select the widget.

2. Under Identity, enter a label in the Label property.
3. Under Identity in the Mnemonic Position property, enter the letter or a numeric position to be identified as the access key.

The property setting returns the position of the character in the label. For example, if you enter the mnemonic A for a push button with the Label property setting Add, the Mnemonic Position property is defined as 1 `A'—the first character in the Add label.

## Using Mnemonics

You can always activate a widget mnemonic from a non-data entry field, such as a radio button or toggle button, by pressing the mnemonic key by itself. If a data entry field has focus, you can activate a menu bar or widget mnemonic by pressing the Alt and mnemonic keys together. In Windows and character mode applications, the screen's menu bar is searched for an active item that uses that mnemonic; if none is found, the mnemonic is sought among the widgets on the current screen. In Motif applications, only menu bar items are searched.

## Specifying the Widget's Data Type

When you import database tables, the widgets that represent the table's columns are automatically assigned a data type by Panther. For the most part, the C Type property does not affect the look or behavior of the widget in your application. For example, a C Type property (under Identity) set to Double, with the Precision property set to 3, does not set or effect a widget with a numeric property specification or perform any validations when the user enters data into the field.

You can use the C Type property to control the way Panther formats data before passing it to a database (refer to “Colon Preprocessing” [on page 30-1](#) in *Application Development Guide* for information). In this way, you can specify the database format as opposed to Panther's format.

Possible C type values are: Default (two decimal places), Omit, Char String, Int, Unsigned Int, Short Int, Long Int, Float, Double, Zoned Dec, Packed Dec, and Hex Dec.

The Omit specification means that the field should be omitted from the data structure.

If you select Float, Double, Zoned Dec, or Packed Dec, you can also specify the Precision property. The default is 2 decimal places.

You can also define zoned and packed decimal types to be either signed or unsigned. In the Sign property, enter Yes to indicate signed; enter No for unsigned.

---

## Invoking a Popup Menu

---

A popup menu is a vertical menu bar. The menu bar that appears on the current screen can be accessed as a popup menu by clicking the right-mouse button. The popup menu appears at the current mouse cursor position.

### How to Invoke a Popup Menu for a Widget or Specific Widgets

1. Select the widget or widgets.
2. Under Help, enter the menu name in the Popup Menu property.  

The specified menu must be in a script that is already loaded in memory. Panther checks the screen-level location first, and then application-level for the named menu.

At runtime, when a user clicks the right-mouse button on the widget, the specified popup menu is displayed. If the widget's Popup Menu property is unspecified, the screen's menu bar is displayed as the popup.
3. To load a popup menu script into memory, under Help in the Popup Script File property, enter the filename or choose More to select the file (\*.mnu) from the Select Library Member dialog box.  

If you do not supply a name in the Popup Menu Name property, Panther displays the first menu in the specified script.



---

## Active Versus Inactive Widgets

---

Active widgets can get focus and respond to user-generated actions. Their appearance indicates that they can be selected or pressed. Widgets that can be made inactive, or temporarily deactivated, are those that have a label—specifically, push buttons, toggle buttons, radio buttons, and check boxes.

You can inactivate specific widgets in a selection group. You must set the Active property under Identity to No for all members of the group to make the group inactive.

### How to Initialize a Widget's State

1. Create or select the widget (dynamic label, push button, toggle button, radio button, or check box).
2. Under Identity, specify the widget's initial state in the Active property.
  - Yes—(default) The widget is active and can receive focus at runtime.
  - No—Sets the widget's initial state to be inactive. The widget appears grayed, and cannot receive focus.

If you define that a dynamic label be initially inactive, consider deactivating its corresponding data entry widget (if one exists) at the same time. Under Focus, set the Focus Protection property to Yes for the data entry widget.

### How to Change a Widget's State at Runtime

For information on accessing and setting a widget's property values programmatically, refer to “Properties” [on page 19-40](#) in *Application Development Guide*.

---

# Protecting Widgets

---

Some widgets, by default, are protected and can only be unprotected at runtime via library functions. For example, dynamic labels are protected from getting focus; the user cannot type data directly into a dynamic label.

There are four types of protection properties available. You can set one or more of these on a widget.

## Focus Protection

When a widget has focus protection, at runtime the user cannot Tab, mouse click, or arrow into the field. However, you can use library functions, for example, `sm_gofield` to move the cursor into the field.

You can set focus protection for the following widget types:

- Single and multiline text
- Option menu
- Combo box
- Scale
- Grid widget

## How to Control a Widget's Focus Protection

1. Select the widget.
2. Under Focus, set the Focus Protection property accordingly.
  - Yes—Protects the selected widget from being accessed at runtime.

Widgets that have horizontal and/or vertical scroll bars can be scrolled and shifted, but the contents of the field cannot get focus. All members of a grid, regardless of their individual specifications, are protected.

Focus-protected widgets, while not included in the tabbing order unless the protections are changed programmatically at runtime, can affect the tabbing order if the Next Tab Stop or Previous Tab Stop properties are specified. Refer to “Setting Tabbing Order” on page 8-12 for information.

- No—(default) The widget is active and can be accessed either by pressing Tab, mouse click, or arrow keys.

**Note:** Individual focus protections for members of a grid are honored at runtime even if the grid widget's Focus Protection is set to No.

## Validation Protection

Validation occurs under the following circumstances:

- The cursor leaves the field for any reason—by pressing Enter, Tab, Backtab, arrow keys, or mouse clicking out of the field. You can limit validation to occur only when the user presses Tab or Enter by setting the setup variable `IN_VALID` to `OK_NOVALID` either as a default or by using the library function `sm_option` to temporarily change the behavior.
- The library function `sm_fval` is called, which validates a field.
- The XMIT key or any widget that you have assigned to execute XMIT is used. All fields on the screen are validated. XMIT is the only key that causes all fields to be validated. You can assign other keys to perform the same function by calling the library function `sm_keyoption`.
- The library function `sm_s_val` is called, which validates all fields on a screen.
- The library function `sm_validate` is called causing the field, the screen, or the tab card/tab deck the field is in to be validated.

You can set validation protection for all widget types except static and dynamic labels and graphics widgets (including lines and boxes).

## How to Protect a Field from Being Validated When the User Exits the Field

1. Select the widget.
2. Under Validation, set the No Validation property accordingly.

- Yes—The selected widget is not validated at runtime, even when the screen as a whole is validated. The keystroke filter specification, if applicable to the widget, is still enforced, but other validation functions associated with the widget are not executed. Library functions cannot be used to validate a field that has the No Validation property.
- No—(default) Validation will occur on field exit and under the conditions stated above.

## Input Protection

When a widget is protected from input, the field rejects all characters that the user types, issuing a beep. The CLR, FERA, DELE, DELL, and INSL keys do not work in fields that are protected from input.

The Input Protection property can be set for the following widget types:

- Single line and multiline text widgets
- Combo boxes

## How to Control Data Entry to a Widget

1. Select the data entry type widget.
2. Under Input, set the Input Protection property accordingly.
  - Yes—Protects the widget from accepting new or changed data.  
Shifting and scrolling text widgets function appropriately; however, the user cannot modify the field content.
  - No—The user can edit and add data to the selected widget.

## Clearing Protection

A field is protected from clearing if any of the following occur:

- Clear Field (CLR) key is pressed.
- Field Erase (FERA) key is pressed.

- Delete Line (DELL) key is pressed.
- Insert Line (INSL) key is pressed.

There are library functions that you can use to clear screens and fields at runtime; however, widgets that are protected from clearing are not affected.

The Clearing Protect property can be set for the following widget types:

- Single line and multiline text widgets
- Combo boxes
- Scales
- List boxes

## How to Define Whether a Widget's Data Can Be Cleared or Not

1. Select the widget.
2. Under Input, set the Clearing Protect property accordingly.
  - Yes—The user can enter data in the field, and can delete data. However, the logical keys that clear data from screens and fields will not affect protected widgets. In addition, if you programmatically clear data from the screen, these fields are protected.
  - No—Data in unprotected fields can be cleared. A scale widget's value is returned to its minimum range (not to its initial value).

---

# Double-Click Events

---

A double click event occurs when the user quickly presses and releases the mouse button twice while over the following widget types:

- Dynamic label
- Single line and multiline text

- List box that is a select-any type or is defined as a selection group
- Combo box

To make something happen when the user double-clicks on a widget, you assign a control string to the widget.

## How to Assign a Double-Click Event

1. Select or create the widget.
2. Under Validation, enter the control string in the Double Click property. Refer to Chapter 18, “Programming Control Strings,” in *Application Development Guide* for details on control string syntax.

At runtime, when the user double-clicks on the widget, the control string is executed. The control string causes one of the following actions or events to take place:

- A named screen displays as a stacked window, sibling window, or form.
- A JPL or C function is executed.  
If you want the double-click event to trigger validation, call `sm_fval` or `sm_validate` as part of the function's processing.
- An operating system command or program is executed.

---

## Setting Tabbing Order

---

Tabbing order refers to the order in which the cursor moves from field to field in response to the TAB and BACK (Backtab) keys or when a character is entered into the last position of a field that has the Autotab property set to Yes. (Refer to “Autotabbing” on page 8-19 for an explanation of auto tabbing.)

The cursor can only move to tab-accessible fields; that is, those widgets that have the Focus Protection property set to No. However, if widgets that are focus-protected have specifications for the tabbing order (in the Next Tab Stop or Previous Tab Stop properties), those specifications affect the tabbing order.

## Forward Tabbing Order

The forward tabbing order is the order of cursor movement when the user presses Tab or when auto tabbing occurs. By default, the forward tabbing order is left to right, and top to bottom. This corresponds to the way Panther internally numbers widgets on a screen.

## Backward Tabbing Order

The backward tabbing order is the order of cursor movement when the user presses the BACK key. By default, the back tabbing order is the reverse of the default for ward tabbing order—that is, right to left, and bottom to top.

## Changing the Tabbing Order

You can change the tabbing order in two ways:

- Setting the Next Tab Stop and Previous Tab Stop properties

Next Tab Stop and Previous Tab Stop settings control cursor position only when the user tabs, auto tabs, or backtabs out of a field. The property specifications have no effect on the cursor positioning keys, mouse clicking, or the NL (newline) key.

- Setting the Focus Protection property

If Focus Protection is set to Yes, the cursor will bypass this widget.

**Note:** If the Next Tab Stop or Previous Tab Stop properties are specified for widgets that are focus-protected, those specifications affect the tabbing order.

## How to Define the Tabbing Order for a Widget or Group

1. Select the widget or group (from the [Widget List](#)).
2. Under Focus in the Next Tab Stop and/or Previous Tab Stop properties, enter the name of the widget or group that should be accessed when the user tabs or backtabs out of the selected object. Choose OK and the Alternate Tab Stop subproperty is displayed.

The Next Tab Stop property controls where the cursor goes next when the user presses the TAB key (or as a result of auto tabbing) in the current field. Panther uses the default forward tabbing order to position the cursor when no Next Tab Stop is defined.

The Previous Tab Stop property controls where the cursor goes when the user presses the BACK key while in the current field. Panther uses the default backward tabbing order to position the cursor when no Previous Tab Stop is defined.

3. (Optional) In the Alt Tab Stop property, enter the name of the widget or group that should be accessed in the event that the named object you identified as the Next/Previous Tab Stop does not exist at runtime.

## Specifying Tab Stops

The following steps illustrate how to specify a tab order from widget A to widget B:

1. Select widget A.
2. Under Focus in the Next or Previous Tab Stop property, enter widget B's name (the name is case-sensitive, so enter the name exactly as specified in its Name property).

**Note:** You can also specify a widget's field number, although field numbers should be used with caution since field numbers can change if you move a widget.

Use the [Widget List](#) to find widget B's name or select widget B on the screen; its name appears in the Widget field at the top of the Properties window. If a field number displays, you can name the widget—in the Name property under Identity.



## Using a Field Number

If you specify a field by number, rather than the widget's name, precede the number with a # (pound) sign. For example, #12 refers to field number 12.

You can use relative referencing to refer to a field relative to the current field. Precede relative references with a plus or a minus sign. The field just before the current field is -1. The fifth field following the current field is +5. To refer to the current field, use +0 or -0. Using a widget name and relative referencing is not a valid designation; i.e., `usr_id +2`.

The following are all valid settings for Next or Previous Tab Stop property settings as well as their Alt Tab Stop settings:

#23	specifies field #23
+12	specifies the 12th field after the current widget
-4	specifies the 4th field preceding the current widget

## Specifying an Occurrence

You can designate a particular occurrence in an array by appending the occurrence number in square brackets to the widget's name or field number. The occurrence number can be absolute or relative.

If you specify an array without giving an occurrence number, the cursor is positioned in the designated array at the same occurrence number it was in when it left the previous array. For example, if the cursor is currently in `item[3]` and the next tab stop is `unit_price`, tabbing out of the current field, specifically `item` at its third occurrence, will move the cursor to `unit_price` at its third occurrence, `unit_price[3]`.

If the current field is not an array, the cursor moves to the first occurrence of the designated array.

The following are all valid designations for specifying a particular occurrence in an array as the Next or Previous Tab Stop:

<code>title[4]</code>	specifies 4th occurrence in the widget <code>title</code>
-----------------------	---

---

<code>title</code>	specifies the array widget <code>title</code> ; cursor moves to the occurrence that matches the occurrence number of the current field
<code>title[-3]</code>	specifies the array widget <code>title</code> ; cursor moves to the third occurrence prior to the occurrence number of the current field

---

Refer to “Element and Occurrence Numbering” on page 14-6 in *Application Development Guide* for more information.

## Specifying a Group

If you specify a group name as the next or previous tab stop, at runtime the cursor moves to the first field of the group. To move the cursor directly to a specific member of the group (for example, the *n*th field of the group), append [*n*] to the group name. For example, `city[3]` refers to the third member in the group `city`.

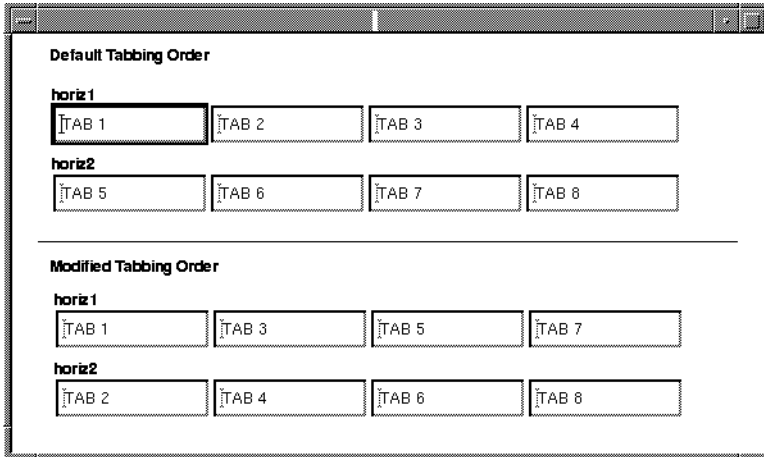
By default, the cursor does not move when the user makes a selection. Setting the `Autotab` property to `Yes` causes the cursor to leave the group once a selection is made.

## Changing the Tab Order

This section includes some examples that illustrate how you can control the tab order.

### In Horizontal Arrays

By default, when `TAB` is pressed in a widget defined as a horizontal array, the cursor moves from left to right in the array. In Figure 8-2, the top portion of the screen illustrates the default tabbing order when there are two horizontal arrays on a screen: a widget named `horiz1` and another `horiz2`. The cursor tabs through `horiz1` before moving to `horiz2`.



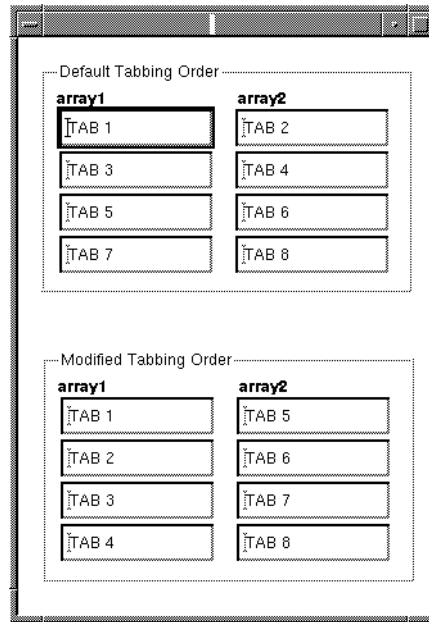
**Figure 8-2** The default tabbing order in horizontal arrays can be modified to alternate between the two arrays.

To modify the tab order so that tabbing moves the cursor from top to bottom, and from left to right (as illustrated in the bottom portion of the screen in Figure 8-2):

1. Select the widget `horiz1`.
2. Specify `horiz2` (with no subscript) as the Next Tab Stop property setting.  
This means when the user presses `TAB` while in `horiz1`, the cursor moves to the same occurrence of `horiz2`.
3. Select the widget `horiz2`.
4. Specify `horiz1[+1]` as the Next Tab Stop setting.  
When the user presses `TAB` while in `horiz2`, the cursor moves to the next occurrence in `horiz1`.

## In Vertical Arrays

As the top portion of the screen in Figure 8-3 illustrates, the default tabbing order causes the cursor to move from the first occurrence of `array1` to the first occurrence of `array2`, to the second occurrence of `array1` to the second occurrence of `array2`, and so on. Scrolling through the occurrences of an array requires the user to press `Enter` or the `DOWN ARROW` key.



**Figure 8-3** The default tabbing order for vertical arrays can be modified to cause the cursor to move through one array before moving to the next.

To change the default tabbing order so that the cursor moves through the screen in columns (as illustrated in the bottom portion of the screen in Figure 8-3):

1. Select the widget `array1`.
2. Specify `array1[+1]` in the Next Tab Stop property under Focus.  
This means that when the user presses TAB, the cursor moves to the next occurrence of the same array. In addition, Panther scrolls an array, if necessary, to make offscreen occurrences visible.
3. Specify `array2[1]` in the Alt Tab Stop subproperty that displays under the Next Tab Stop property. This designation instructs Panther what to do when the user reaches the end of `array1` (just beyond the maximum occurrence number), where its next field designation will fail.

The Alt Tab Stop designation causes the cursor to move to the top of the second array, and scroll both arrays so that their first occurrences are displayed in the top field of each.

4. Select the widget `array2`.
5. Specify `array2[+1]` in the Next Tab Stop property.
6. (Optional) Specify the name of the field to move to after tabbing through `array2` in the Alt Tab Stop property.
7. (Optional) For both widgets, use the Previous Tab Stop property to make BACK have the opposite effect.

## Autotabbing

When a user fills a field to its last position, by default, the cursor remains in the field until the user presses Tab, Enter, or mouse clicks out of the field. You can force the cursor to automatically move onto the next field when a user enters the required data in a field or makes a selection.

The Autotab property can be set for the following widget types:

- Single line and multiline text
- Option menus and combo boxes
- Selection groups

## How to Control Autotabbing

1. Select the widget or group.
2. Under Focus, set the Autotab property accordingly.
  - Yes—At runtime, Panther automatically tabs (moves the cursor) to the beginning of the next field under the following conditions:
    - For single line and multiline text widget (default Yes)—When the current field is filled to its maximum length specification and passes validation.
    - For option menus and combo boxes (default Yes)—When the user makes a selection, or, in the case of a combo box, when the text portion of the widget is filled to its maximum length specification.
    - For grouped selection type widgets (radio buttons, check boxes, toggle buttons, and list boxes) (default No)—Once the user makes a selection, the cursor leaves the group. In a list box, this property setting causes the

cursor to move to the next item in the list.

- No—(default for selection groups) The user must press Tab, Enter, some other cursor movement key, or mouse click to leave the field. The cursor causes a beep to be issued and remains in the last position of the field or group; for a grouped list box, the focus remains on the last selected item.

---

## Synchronizing Scrolling Arrays

---

Scrolling arrays can be synchronized so that they scroll together. This helps manage related information in table-like screens. Panther automatically synchronizes arrays when they meet either of the following conditions:

- The widgets are grid members within a grid widget.
- They are database-derived widgets that belong to the same table view or to different table views that are joined by a server link.

You can manually synchronize arrays that do not meet the above conditions by making the widgets members of a synchronized scrolling group. Refer to “Creating and Modifying Synchronized Arrays” [on page 8-22](#) for instructions.

### Array Behavior

The behavior of synchronized arrays is largely determined by the following rules:

- If a keypress causes a single array to scroll, then that key press causes all members of the synchronized group to scroll.
- Because each array in the synchronized group has the same number of allocated occurrences, if a key press allocates (de-allocates) an occurrence of an array, then the same occurrence is allocated (de-allocated) for every member of the synchronized group.

You can also prevent one or more arrays within a synchronized group from deletion and insertion of occurrences by setting the Clearing Protect property to Yes for the array you wish to protect. This lets you synchronize a protected array that contains an unchanging sequence of numbers with an adjoining unprotected array whose data grows and shrinks

**Note:** Setting an array's Clearing Protect property to Yes does not prevent deletion and insertion of occurrences in the array when it has focus.

## Logical Key Behavior

The movement of the cursor from one occurrence to the next in an array occurs as follows:

- NL (or Enter) moves the cursor through each visible occurrence of the array that contains data. When the cursor is in the last element of the scrolling array, the data scrolls, bringing the next occurrence into view, and also causes other arrays synchronized with the current field to scroll.
- Up and Down arrow keys move to the next element in the array. In a scrolling array, the Up and Down arrow keys cause the cursor to scroll through all allocated occurrences of the array before advancing to the next field.

## Automatic Synchronization

If you are accessing arrays with the transaction manager, all the widgets that are members of the same server view must have the same number of occurrences if those widgets are participating in SQL INSERT, UPDATE, and DELETE statements that occur as a result of the SAVE command. Otherwise, the transaction manager displays an error message that it is unable to synchronize the table view.

This automatic synchronization of arrays by the transaction manager is attempted:

- If the widget belongs to an updatable table view.
- If the widget's Synchronization property under Transaction is set to Default or Yes.
- When the Synchronization property under Transaction is set to Default, if the widget has the Use In Update, Use In Insert, or Use In Select properties (under Database) set to Yes.

Synchronization of arrays is necessary for commands that alter the database—**NEW**, **COPY**, and **SELECT**. If it fails, an error message is displayed. Synchronization of arrays is also attempted for the **VIEW** command; however, no error message is displayed if it fails.

If you get a synchronization error for the table view and you do not want to change the number of occurrences, you can review the Use In Update, Use In Insert and Use In Select properties for the widgets in the server view. Also, the Synchronization property for a widget can be set to No. However, changing the Synchronization property will not change the way Panther displays data in arrays. Panther fetches the number of rows equal to the smallest number of occurrences for the widgets in a server view whose Use In Select property is set to Yes.

## Creating and Modifying Synchronized Arrays

To ensure that arrays are synchronized regardless of where they are on the screen, you can create a synchronized scrolling group. (You do not need to do this for arrays in a grid widget, or for those that are synchronized by virtue of the table view to which they belong.)

### How to Synchronize a Set of Scrolling Arrays

1. Select the arrays.
2. Ensure that the value set in the Onscreen Rows or Array Size property is the same for all widgets in your selection set (a number in the setting indicates that the setting is the same; three question marks (???) indicate that the setting is different).
3. If you are synchronizing a group of Multiline Text widgets, ensure that the Word Wrap property (under Format/Display) is set to No.
4. Choose Edit→Group→Create→Synchronized Scrolling.

All selected arrays will scroll together.

### How to Change the Size and/or Behavior of a Synchronized Scrolling Group

1. Select the group by doing either of the following:



- Select one of the members of the group. Then choose Edit→Group→Select Groups. If the widget belongs to more than one group, the Select Group dialog box opens. Select the Sync. Group from the list.
- Select the Sync. Group from the [Widget List](#).

The Properties window displays Sync. Group-specific properties.

2. Under Geometry, set any of the following properties for the group:
  - Array Size—Enter the desired number of onscreen occurrences. All members of the synchronized group adjust to the specified size.
  - Max Occurrences—Enter the total number of occurrences. Set the property to blank to allow for the maximum number of occurrences possible (the actual maximum is platform-specific). A value greater than that specified in the Array Size property allows the array to scroll.
  - Scroll Increment—Specify the number of occurrences by which all members should scroll when the user presses Page Up or Page Down. The default is one less than the number of onscreen occurrences.
  - Circular—Set the property appropriately.
    - Yes—Causes the first element to redisplay after the last element in the group is reached, and vice versa.
    - No—(default) If the user presses the arrow keys to move the cursor to the last/first element of any member of the group, the cursor moves to the next or previous field. If the user presses Page Down/Up to scroll the occurrences, Panther displays the message “No more data” when the first/last element is reached.

## Identifying Members of the Synchronized Scrolling Group

### How to Identify All Members of a Synchronized Scrolling Group

1. Do either of the following:
  - Select one or more members of the group.
  - Select the Sync. Group from the [Widget List](#).

2. Choose Edit→Group→Select Members. All widgets associated with the synchronized group are selected.  
  
If you select widgets associated with two or more groups, the Select Group Member dialog box opens. Continue to step 3.
3. Select the group from the left hand column and choose OK. All members that comprise the group are selected.

## **Changing Members of a Synchronized Scrolling Group**

### **How to Add a Scrolling Array to an Existing Group**

1. Select all members of the existing group by doing one of the following:
  - Shift+click on each member of the group.
  - Select at least one member of the group, and choose Edit→Group→Select Members.
  - Select the Sync. Group from the [Widget List](#).
2. Shift+click to add the desired array to the selection set.
3. With all the widgets selected that are to be grouped, choose Edit→Group→Update Group Members.

The existing group now includes the additional arrayed widget.

### **How to Isolate an Array That is Synchronized Because It Belongs to a Table View or to Table Views Joined by a Server View**

1. Select the scrolling array you do not want to be synchronized.
2. Under Transaction, set the Synchronization property to No.

### **How to Isolate an Array That is Part of a Synchronized Scrolling Group**

1. Select members of the group that you wish to keep in the existing group. Do this by either:

- Clicking on the desired widgets in the group. Omit the array that you want to remove from the group.
  - Selecting at least one member of the group, and choosing Edit→Group→Select Members. Then deselect the array you want to isolate.
2. With the widgets selected that are to remain synchronized, choose Edit→Group→Update Group Members.

The group is redefined to include the new set of widgets.

## Sorting Data in Synchronized Scrolling Arrays

You can sort data in arrays by specifying a custom sort order function or by calling the C function `sm_obj_sort`.

Calling `sm_obj_sort` for a widget in the array will use the order specified in the widget's Sort Order property: Lexicographic, Numeric, Date/Time, or Custom.

If you choose to write a custom function, it must conform to certain requirements. For more information, refer to [page 15-12](#), “Writing a Custom Sort Function.”

## Using an Alternative Scroll Driver

Panther uses memory-based scrolling by default. If you want to save memory at the expense of speed, you can set up a disk-based scroll driver by writing your own scrolling function. Attach the scrolling function to your arrays via the Alt Scroll Func property.

---

# Performing Calculations and Validating Numbers

---

You can attach math and check digit calculations to widgets; they are performed at runtime when the field is validated.

- Math calculations can access and change the content of any widget, group, or LDB widget.
- A check digit calculation is used to validate a widget according to a standard check digit algorithm.

## Math Expression Syntax

### How to Attach a Math Calculation to a Selected Widget

Under Validation, define the math expression in the Calculation property.

You can attach one or more math calculations to a widget; separate expressions with a semicolon. The math expression takes the following format:

```
[%m.n | %tm.n.] destination_widget = calculation
```

The square brackets indicate an optional specification. Do not include the brackets in the specification.

The optional size specification is followed by the destination widget designation (e.g., field name or number), an equal sign, and the body of the expression. The optional floating point size specification defines the destination widget.

**Table 8-1 Specifications for math calculations**

Specification	Description
<code>%m</code>	The total number of characters in the output. If no size is supplied, the total length defaults to the length of the destination widget.
<code>n</code>	The number of digits after the decimal point. If no size is supplied, it defaults to the number of decimal places in the destination widget's Numeric format specification; if there is no Numeric specification, default precision is determined by the <code>DECIMAL_PLACES</code> option (defined in a setup file). You can change the setup specification with a runtime call to <code>sm_option</code> .
<code>%t</code>	Forces truncation rather than rounding of the result.
<code>destination_widget</code>	Designates the destination widget by name or by number. <p>Field numbers must be preceded by a # sign; for example, #12 refers to field number 12.</p> <p>To refer to the current widget, use #+0 or #-0.</p> <p>You can use relative referencing to designate a widget relative to the field number of the selected widget. Relative references must be preceded by a plus or a minus sign. The widget just before the current widget is #-1. The fifth widget following the current widget is #+5.</p> <p>Designate a particular array occurrence by appending the occurrence number in square brackets to the widget name; for example, <code>title[4]</code> denotes the fourth occurrence of the array <code>title</code>. The number can be absolute or relative. If a designated widget is part of an array <i>and</i> the designation contains no occurrence subscript, then the current occurrence number is used as the array index. The current occurrence number is used as an index to fetch the desired value. However, if the current occurrence number is greater than the number of occurrences in the designated array, an error results.</p>

**Table 8-1 Specifications for math calculations (Continued)**

Specification	Description
<i>calculation</i>	Can contain numeric constants, widget designations, parentheses, and the arithmetic operations +, -, *, / and ^ (raise to a power). For example:
	<pre>%8.0 amount = line_item * 12 + 2</pre>
	<pre>flda[2] = (flda[1] - 6.235) / fldb[1]</pre>
	For more information on operators and expressions, refer to “Data Types, Operators, and Expressions” on page 19-46 in <i>Application Development Guide</i> .

## Using Functions

There are three special functions that you can use in math expressions:

- **@sum**—Calculates the sum of all the occurrences in a given array. For example, `@sum(array1)` totals the values in `array1`; `@sum(#2)` totals the values in the array containing field number 2.
- **@date**—Calculates the number of days between January 1, 1753 and the argument to `@date`. The argument must be either a widget name or field number that has a Data Formatting property set to Date/Time and a format specification that expects MM/DD/YY or MM/DD/YYYY format, or a literal date in the format specified in the message file under the constant [SM\\_CALC\\_DATE](#). The default literal format is `%m%d%4y (MON/DATE/YR4)`. (For information on specifying Date/Time properties, refer to [page 10-17](#), “Creating a Date and Time Field.”)

The following is an example of the `@date` function:

```
@date(3/31/1985)
```

An error occurs if the destination widget does not have the proper format. The number resulting from the calculation is interpreted as the number of days elapsed since 1/1/1753. If a destination widget has a date format, then the result is displayed according to its format; otherwise the date is displayed as the number of days since 1/1/1753. If `fieldA` and `fieldB` have the Data Formatting property set to Date/Time, the following expression

```
fieldB = @date(fieldB) + 30
```

sets the date in `fieldB` to 30 days past the date in `fieldA`.

The following expression sets `daysgoneby` to the number of days between `field1` and `field2`:

```
daysgoneby = @date(fieldB) - @date(fieldA)
```

- `@length`—Counts the number of characters, including embedded blanks, in one or more string arguments, and returns an integer. For example, the following expression counts the number of characters in `lname` and `fname`:

```
@length(fname, lname)
```

## Check Digit Calculations

A check digit calculation is used to validate a data entry widget (single line text, multiline text, and combo box widgets).

### How to Attach a Check Digit Calculation to a Widget

1. Select the data entry widget that requires the check digit validation.
2. Under Input, set the Keystroke Filter property to Digits Only.
3. Under Input, enter the modulus, 10 or 11, in the Check Digit property. Panther supports the mod-10 and mod-11 algorithms. The Minimum Digits subproperty is displayed.

**Note:** The source code to the check digit validation function `sm_ckdigit` is provided and can be modified to support other check digit algorithms.

4. (Optional) Under Check Digits in the Minimum Digits property, enter the minimum number of digits that must be entered by the user before the algorithm is invoked.

# Documenting Widgets

---

Two properties under Identity in the Properties window help you document the contents of your widgets:

- Comments property—Provide a brief description of the widget.
- Memo Text property—Attach up to nine lines of text for additional comments or for programmatic use.

## Adding Comments

By including a brief description of your widget, you can provide information to other developers about the widget, its content, and its purpose. This description is displayed when you view the Repository TOC and scroll through each of the repository entries.

## How to Add or Edit Comments for Your Widget

1. Select the widget.
2. Under Identity, select the Comments property. The Comments dialog box opens.
3. Do one of the following to enter or edit text:
  - Type the text directly in the text area.
  - Choose Editor to access your local editor (as defined in the `SMEDITOR` variable; refer to [page 2-6 in Configuration Guide](#)).
  - Choose Read File to read in an external file located on your system.
  - Choose Save File to save the comments to an external file.
4. Choose OK to save the comments and return to the Properties window.



## Including Additional Information

### How to Include Additional Information for a Widget

1. Select the widget.
2. Under Identity, expand the Memo Text subheading. Nine memo lines are displayed.
3. Type additional information, comments, and specifications that you can't indicate or specify as a screen property.

At runtime, you can access the widget's Memo Text property to determine what action to take depending on the property's content.

---

## Defining XML Tags for Widgets

---

As of Panther 5, you can import and export data from a Panther screen to a buffer or file. Starting in Panther 5.40, you can also create XML reports. To use these features, you must:

- Define properties for the screens and the widgets to be included in the XML data.
- Call the appropriate functions to write data to the buffer or file holding the XML data.

This feature is available for the following widget types:

- Text widgets: dynamic labels, single line text widgets, multiline text widgets, option menus, combo boxes and listboxes
- Container widgets: boxes, tab decks, tab cards and grids

This section describes the property specifications for widgets. For information on XML screen properties, refer to [page 8-31](#), “Defining XML Tags for Screens.” For details on using the interface, refer to Chapter 22, “Using XML Data,” in the *Application Development Guide*.

## How to Set XML Properties for Widgets

1. Select the widget.
2. Under XML, enter a value in the XML Tag property.
3. *(Optional)* If you entered an XML tag for the widget, you can enter additional attributes in the XML Attributes property. This information will appear in the XML opening tag.
4. *(Optional)* Enter values in the XML Prefix and XML Postfix properties.

## Generating the XML for Widgets

When the XML is generated for the screen, container widgets are handled differently than data entry and selection widgets.

For container widgets (boxes, tab decks, tab cards and grids), Panther generates the following:

- If specified, the XML Prefix property value is output.
- The opening tag and any attributes are output.
- XML is generated for the widgets within the container.
- The closing tag is output.
- If specified, the XML Postfix property value is output.

A container is included in the generated XML if any of its widgets have XML property values.

For data entry and selection widgets (dynamic labels, single line text widgets, multiline text widgets, option menus, combo boxes and listboxes), Panther generates the following:

- If specified, the XML Prefix property value is output.
- The opening tag and any attributes are output.
- The data from the widget is output.
- The closing tag is output.
- If specified, the XML Postfix property value is output.

For a sample file, refer to “Sample XML File” [on page 22-5](#) in *Application Development Guide*.

## Processing XML for Multiple Occurrences

When a grid is converted to XML, the maximum number of occurrences of each grid member having an XML Tag property determines the number of occurrences to output. All offscreen data will be written to the XML.

## Importing XML to Screens

Panther screens can import XML files. For this feature to work, the tags in the XML file must match the tags specified for the screen and the widgets.



# 9 Manipulating Widgets

The chapter describes some basic procedures needed for widgets in the Editor.

---

## Selecting a Widget

---

You must select a widget or widgets to set properties at the widget-level. A selected widget is displayed with a changed border—on GUI platforms small, black boxes, or grab handles are displayed; in character mode, the widget is surrounded by square brackets ([ ]). A selected grid widget member is displayed in reverse video from unselected members.

When more than one widget is selected, the first one you select is considered the *dominant selection*; all other widgets in the selection set are indicated with hollow boxes in GUIs and curly braces ({ }) in character mode. The position and size of the dominant widget determines how the other widgets in your selection set will align or resize.

**Note:** If you are using Panther in character mode without a mouse, refer to Appendix A, “Keyboard Interface,” for details on using the keyboard to manipulate widgets and navigate in the Editor.

## Using a Mouse

You can select widgets in the following ways:

### Mouse Click

Click on a widget with the mouse. Only one widget can be selected using this method. All previously selected widgets on the screen are deselected.

(For selecting grid widgets and their members, refer to [page 15-2](#), “Manipulating a Grid Widget and Grid Members.”)

### How to Select Multiple Widgets

- Press Shift+click to select and deselect widgets without affecting the selection state of other widgets.
- Press Ctrl+click to select a new widget and make it dominant.

**Note:** In character-mode Panther, Shift+click and Ctrl+click are not available. Choose Options→Multiple Select Mode (default setting). You can then click on multiple widgets; the first widget you select is considered the dominant selection. To change the dominant widget, deselect all widgets and choose the one that should be dominant, then reselect all others.

### Rubberbanding

With the mouse button pressed, drag a bounding box, also known as a rubberband, around the desired widgets. This method allows you to select contiguous widgets. Any widgets that fall within the rubberband boundary are selected. Only one widget in the selection set is dominant.

Press Ctrl+click on any widget in your selection set to make that widget the dominant selection.

Shift+rubberband to toggle the selection state of all widgets within the bounding box.  
Ctrl+rubberband to select any widgets that are not already selected; this does not change the dominant selection.

## Deselecting Widgets

### How to Deselect Widgets

Do either of the following:

- Shift+click to toggle selection for a specific widget in a selection set.
- Click in an empty area of the screen to deselect all widgets.

## Using the Widget List

Use the Widget List to:

- Select one or more widgets in the current screen.
- Select invisible widgets in order to access their properties, such as table views and groups.
- Locate widgets that are hidden behind other widgets or are located outside of the screen's viewport.
- Select the screen, report, service component, or frameset (in Panther 4.50 and later) by deselecting any selected widgets.

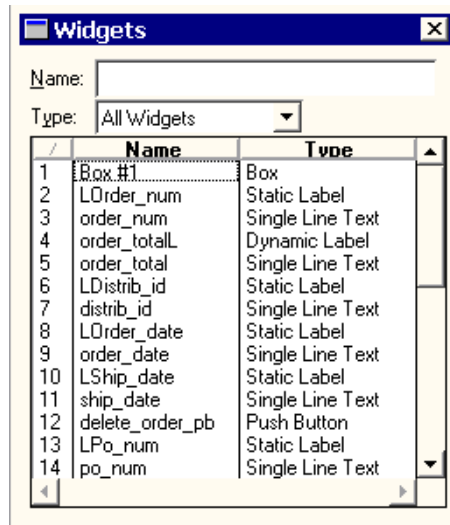
You can use the Type option menu to only view widgets of one widget type. In a GUI, you can click on the Name and Column labels to change the sort of the Widget List.

You cannot move widgets in the Widget List window; however, once widgets have been selected, you gain access to their properties and you can move, copy or delete the selected widgets together.

### How to Select Objects Using the Widget List

1. Choose View→Widget List. The Widget List is displayed.

The widgets on the current screen are listed in top to bottom and left to right order. The left column lists the widget numbers; the middle column lists the widgets by name (or by field number or contents if the widget is not named); the right column lists the widget type.



**Figure 9-1** The Widget List's left column is the widget number; the middle column is the widget name (or field number or contents) and the right column is the type.

2. Scroll through the list of widgets by using the scroll bar or the Up and Down arrow keys.
3. Select a widget by clicking on it in the Name column; Shift+click to select contiguous widgets and Ctrl+click to select and deselect noncontiguous widgets.
4. Use the Type option menu to display only widgets with a selected Widget type.
5. In a GUI, click on the Number, Name and Type columns to sort the columns.
6. The Delete key will delete any widgets that are currently selected.



---

## Resizing a Widget

---

You can resize, copy, and move widgets:

- By using the mouse.
- Through the Properties window.
- By using Edit menu commands and toolbar buttons.

**Note:** Some selection methods are unique to the platform on which Panther is running.

If you are using Panther in character mode without a mouse, refer to Appendix A, “Keyboard Interface,” for details on using the keyboard to manipulate widgets and navigate in the screen editor.

You can resize and control the size of a widget by:

- Dragging the widget to the desired size.
- Specifying its size in the Properties window as well as defining how much or how little it should grow at runtime if the screen size changes.
- Selecting more than one widget and making all widgets in your selection set the same size.
- Adjusting the widget size to fit its textual content.

Use the method that works best for your immediate requirements.

If the size of the widget is inherited from a repository widget, Panther reminds you that inheritance for that Geometry property will be removed. You can choose to either change the size or retain the inherited size. Refer to “Controlling Inheritance” [on page E-29](#) for details.

## Defining a Widget's Size

A widget's horizontal dimension (with the exception of scales, lines, and boxes) is defined by its Length property setting. This value defines the amount of data, in columns, that the widget can display on the screen. Its vertical dimension is defined by the number of onscreen occurrences and the size of the font that the widget uses.

Height and Width properties define a widget's physical size. And, a widget's length is not necessarily dependent on its width specification. So, you can have a widget appear wider than the actual amount of data it will display.

Default-sized widgets do not have Height and Width specifications. When you resize a widget, Panther automatically applies values to these properties.

## Dragging to Size

### How to Resize a Widget

1. Select the widget.
2. Position the mouse pointer on one of the widget's grab handles (or brackets) and drag it until the widget is the size you want.
3. Release the mouse button.

## Specifying a Size

To resize one or more widgets to a specific size, use the Properties window:

Select the widget or widgets that you want to resize. Under Geometry, enter a value (default is grid units) in the Height and/or Width properties. Refer to Table 9-1 for a list of valid units of measurement.

For example, 5c means five characters wide, 5 (without a unit of measure) denotes five grid units.

The width and/or height of the selected widgets adjusts according to your specification.

**Table 9-1 Units of measurement for sizing and positioning widgets**

Unit	Use	Description
characters	c	Based on the widget's font size (average character width and height). A setting of 5c means five average-size characters in the widget's font, while grid units specify average-size screen font. Characters and grid units are the most portable units of measure, since they are sensitive to the font in use. For GUIs, you can specify whole numbers plus fractions (such as 5.5c).
grid units	g	The default unit of measure and the only valid specification for character mode. A grid unit is based on the average character size of the screen font. A setting of 5 or 5g means five standard grid cells. Grid units and characters are the most portable units of measure, since they are sensitive to the font in use. For GUIs, you can specify whole numbers plus fractions (such as 4.25).
inches	in	Specifies size or position in inches.
millimeters	mm	Specifies size or position in millimeters.
pixels	p	Must be specified in whole numbers. These measurements depend upon screen resolution.

## How to Size a Widget To Fit Its Content

1. Select the widget (applies to widgets that have a label: static and dynamic labels, push buttons, radio buttons, check boxes, and toggle buttons).
2. Under Geometry, set the Size to Contents property appropriately.
  - Yes—(default for static labels) Trailing and leading blanks are eliminated from labels, extra space width eliminated from grid widgets, and any previously set Height and Width property values are removed.

The widget (except for static labels) dynamically resizes at runtime if its content changes.

If you increase or decrease the number of characters in the widget's Label property, the widget's length adjusts appropriately to fit the content. If you

reset both Height and Width properties, the Size to Contents property is ignored.

- No—(default) The widget's size remains fixed at runtime regardless of its content.

**Note:** If you change the Length property (either by setting the property value or by resizing the widget itself) or specify a picture to display on the label, the Size to Contents property is automatically reset to No.

## Unifying Widget Size

You can resize all widgets in your selection set to be the same size by using the Size command; adjust both the width and height of the widgets.

### How to Uniformly Size Widgets

1. Select the widgets that you want to resize.
2. Determine and indicate the dominant widget—that is, select the widget that all other widgets will match in size.



3. Choose Edit→Size and the directional dimension or the appropriate Size button on the toolbar.
  - Adjust Widths—Resizes the onscreen widths (or visible columns) of the selected widgets to match the onscreen width of the dominant widget.
  - Adjust Heights—Resizes the onscreen heights (or visible rows) of the selected widgets to match the onscreen height of the dominant widget.
  - Both—All widgets in the selection set resize to match the height and width of the dominant widget.

The widgets in your selection set match the height and/or width of the dominant widget.

---

## Controlling a Widget's Size at Runtime

Only in GUIs does Panther provide properties for certain widgets which allow them to grow and shrink proportionally when the user resizes the screen at runtime. The following widget types can be configured to resize:

- List box
- Grid widget
- Single line (resize horizontally only) and multiline text.
- Horizontal lines (resize horizontally) and Vertical lines (resize vertically).
- Graphs

The screen's `Resizable` property (under `Geometry`) must be set to `Yes` to allow widgets to resize.

**Note:** The `Resizable` property is supported only in GUIs.

### How to Define the Vertical Resizing Behavior for Capable Widgets

1. Under `Geometry`, set the `Auto Vert Resize` property to `Yes`. Additional vertical resize subproperties are displayed.
2. (Optional) Specify the maximum size to which this widget can grow vertically by setting the `Max Size` (`vert_max_size`) property.
3. (Optional) Specify the minimum size to which this widget can shrink vertically by setting the `Min Size` (`vert_min_size`) property.

For valid units of measurement, refer to Table 9-1 on page 9-7.

### How to Define the Horizontal Resizing Behavior for Capable Widgets

1. Under `Geometry`, set the `Auto Horiz Resize` property to `Yes`. Additional horizontal resize subproperties are displayed.
2. (Optional) Specify the maximum size to which the widget can grow horizontally by setting the `Max Size` (`horiz_max_size`) property.
3. (Optional) Specify the minimum size to which the widget can shrink horizontally by setting the `Min Size` (`horiz_min_size`) property.

For valid units of measurement, refer to Table 9-1 [on page 9-7](#).

### General Rules for Resizing at Runtime

This section explains how the auto-resize properties affect behavior at runtime. Review the following rules when designing your screen; they will help you take advantage of Panther's automatic resizing capability. In addition, if you find that Panther's built-in resize capability does not satisfy a specific requirement, you can attach a custom resize function via the screen's `Resize Function` property.

When configuring Panther for your desired resize behavior, think in terms of each resizable widget forming a path across the screen. The path occupies the area that the widget would take up if it were resized horizontally to its maximum in either direction. If one or more resizable widgets share the same `Starting Row`, `Height`, and `Vertical Anchor`, then these widgets share the same path. If widgets with either `Horiz` or `Vert Auto Resize` property set have an overlapping, but non-identical path, then the widgets cannot resize.

#### Resize Processing Rules for Panther's Built-in Resize Capability

- When a screen is made smaller, Panther preserves any white space margins that exist along the bottom and right side of the screen and applies the screen resize percentage directly to the resizable widgets.
- When stretching or shrinking widgets, Panther first determines the new size of these widgets, and then applies its positioning algorithm to re-orient the remaining widgets around those which have been resized.
- When widgets grow, white space is preserved. The screen's layout is maintained as originally presented, except that certain widgets are bigger.
- If one or more resizable widgets share the same path, Panther preserves their relative widths after a resize. For example, if a screen with a one-inch wide widget next to a two-inch wide widget has its horizontal dimension increased by three inches, the widgets' widths increase to two inches and four inches, respectively. Thus, the widgets maintain the size ratio of 1:2 which existed before the screen was resized.
- When a widget reaches its maximum (or minimum) resize restriction, all other widgets on the screen stop growing (or shrinking). This restriction assures that the resize ratio within resize groups is preserved.

---

# Moving and Copying Widgets

---

Alternative methods for moving and copying widgets in character mode are also described here, since character mode Panther is unable to take advantage of Shift+click and Ctrl+click to move and copy objects.



When you move or copy a widget, its properties and all other pertinent information go with it, including its inheritance.

## Within a Screen

### How to Move a Widget Within a Screen

1. Position the mouse pointer anywhere on the widget, except on a grab handle or bracket.
2. Drag the widget to the desired location on the screen.

### How to Copy a Widget Within a Screen

- Ctrl+drag your selection to the desired location.

Alternatively and in character mode:

1. Select the widget.
2. Choose Edit→Copy (or the Copy button from the toolbar).
3. Then choose Edit→Paste (or the Paste button).

The copied widget is pasted down and to the right of the original widget by one grid unit.

**Note:** In character mode, if you are in Multiple Select Mode, remember both the source and copied widgets are selected and it is difficult to carry out an operation without affecting both widgets—in other words, both widgets will move together.

## From Screen to Screen

### How to Move a Widget From One Screen To Another

- Shift+drag the widget.

Alternatively and in character mode:

1. Select the widget
2. Choose Edit→Cut (or the Cut button on the toolbar).
3. Bring focus to the destination screen
4. Choose Edit→Paste (or the Paste button).

The widget is pasted in the same location in the destination screen as it was in the source screen. If another widget is in the destination location, the widget is positioned down and to the right by one grid unit.

### How to Copy a Widget To Another Screen

- Drag and drop the widget to the destination screen.

## To and From a Repository

You can add a widget to your application screen by dragging (copying) a named widget from a repository entry, and thereby inherit properties from a single source.

You can also do the reverse operation: copy named widgets from your application screen to a repository entry. The original widgets become the children and are automatically defined to inherit from the copied widgets in the repository.



The inheritance relationship between parent objects in the repository and their child objects can be retained, thus ensuring that standards are established and that changes in repositories are propagated to your application appropriately. Refer to “Controlling Inheritance” on [page E-29](#) for more information on repositories and inheritance.

## Deleting Widgets

### How to Delete or Remove a Widget

Select a widget. Do either of the following:

- Choose Edit→Cut or the Cut button from the toolbar.

This places the removed widget in a temporary buffer. You can then paste the widget in the same screen or in another screen. The cut object remains in the buffer until it is overwritten by the next Cut operation.

- Choose Edit→Delete or press the Delete key.

The selected widget is not placed in a temporary buffer; it cannot be restored to the screen (except by choosing Undo).

---

## Arranging Widgets

---

You can arrange widgets along vertical and horizontal axes; you can align or space them in relation to one another or arrange them at specific positions on the screen. You can do this by:

- Using Align and Space commands on the Edit menu or the appropriate Align and Space buttons on the toolbar.
- Specifying coordinates under the Geometry properties in the Properties window.
- Drag widgets to the position on the screen.

## Aligning Widgets

When you arrange widgets on your screen using the Align command, the dominant widget remains fixed, and the other widgets in your selection set align with it. The Align command doesn't change the size of the widgets.

### How to Align Widgets with Each Other

1. Select two or more widgets on a screen.
2. Determine the dominant widget—that is, select the widget that all other selected widgets should align with.
3. Choose Edit→Align and the orientation or the appropriate Align button on the toolbar.



- Left—Aligns the left edges of selected widgets with the left edge of the dominant widget. The vertical positions do not change.
- Center—Aligns the horizontal midpoint of the selected widgets with the horizontal midpoint of the dominant widget. The vertical positions do not change. (Toolbar button is not available.)
- Right—Aligns the right edges of selected widgets with the right edge of the dominant widget. The vertical positions do not change.
- Top—Aligns the top edges of selected widgets with the top edge of the dominant widget. The horizontal positions do not change.
- Middle—Aligns the vertical midpoint of selected widgets with the vertical midpoint of the dominant widget. The horizontal positions do not change. (Toolbar button is not available.)
- Bottom—Aligns the bottom edge of selected widgets with the bottom edge of the dominant widget. The horizontal positions do not change.

### How to Align Widgets to an Absolute Position or Coordinate

1. Select the widgets that you want to align.

In the Properties window, three question marks are displayed in the Start Row and Start Column properties (under Geometry) if the selected widgets are not already aligned on the same coordinate.

2. Under Geometry, enter a value and unit of measurement in the Start Row and/or Start Column property. Refer to Table 9-1 on page 9-7 for a list of valid units of measurement.

Changing the Start Row aligns the top edges of selected widgets on the specified unit; changing the Start Column aligns the left edges of the widgets on the specified unit.

3. Choose OK.

The selected widgets, including the dominant selection, align accordingly on the specified coordinate.

## How to Position a Widget on a Grid Coordinate

1. If the screen grid is not displayed, choose Options→Grid (or the Grid button on the toolbar) to toggle the screen grid display on.
2. Select the widget or widgets.
3. Choose Edit→Grid Align and the position:

**Note:** The Grid Align command is dependent on the screen grid display.

- Horizontal—Aligns the widget's horizontal anchor point to the nearest horizontal grid coordinate.
- Vertical—Aligns the widget's vertical anchor point to the nearest vertical grid coordinate (whole number).
- Both—Aligns the widget's horizontal and vertical anchor points to the nearest grid coordinate (whole number).

**Note:** To change a widget's anchor point, under Geometry, set the Horizontal and/or Vertical Anchor properties accordingly (refer to page 9-18, “Controlling a Widget's Position at Runtime”).

## How to Force All Widget Placement on a Grid Coordinate

Choose Options→Snap To Grid.

- With Snap to Grid on, a widget that you add or position on a screen automatically aligns, using the widget's anchor points, to the nearest vertical and horizontal grid coordinate.
- If Snap to Grid is off, you can place widgets between grid marks under GUI platforms. When you do so, their Start Row and Start Column properties have decimal values, such as 5.35.

In a character environment, all objects are aligned to a grid coordinate, so the Start Row and Start Column properties will have whole number values.

## Spacing Widgets

By using the Space command, you can position widgets on your screen so that there is an equal amount of space between the widgets in your selection set. You can let Panther automatically space the widgets evenly, or you can explicitly set the distance between widgets.

## How to Create Automatic Spacing

To create an equal amount of space between widgets:

1. Select three or more widgets.

The widgets that are outermost in your selection set—that is, the left- and right-most, or top- and bottom-most—are designated as left and right anchors, or top and bottom anchors, respectively. The anchors do not move when you use the Space command; the other widgets in the selection set adjust positions within the anchors' boundaries.



2. Choose Edit→Space and an orientation or choose the appropriate Space button on the toolbar.
  - Horizontal Spacing—Positions widgets evenly between the left and right anchors.
  - Vertical Spacing—Positions widgets evenly between the top and bottom anchors.

The result of the Space command is an equal amount of horizontal and/or vertical space between the selected widgets.

## How to Create Custom Spacing

To specify an absolute amount of space between widgets in a selection set:

1. Select two or more widgets on a screen.
2. Determine the dominant selection—the dominant widget is the anchor and does not move.
3. Choose Edit→Space→Custom. The Set Custom Spacing dialog box opens.
4. Enter the amount of space required between the widgets in the Distance field. Refer to Table 9-1 [on page 9-7](#) for a list of valid units of measurement.
5. Select a directional dimension.
  - Horizontal—Widgets to the left of the dominant selection reposition to its left. Widgets to the right of the dominant selection reposition to its right. The alignment of the left edges of widgets determine whether they position to the right or left of the dominant widget.
  - Vertical—Widgets above the dominant selection reposition above it. Widgets below the dominant selection reposition below it. The alignment of the top edges of widgets determines whether they position above or below the dominant widget.
6. Choose OK.

If there is insufficient space to arrange the widgets using the specified value, a message is displayed. You might correct the problem by trying any of the following:

- Decrease the value in the Distance field on the Set Custom Spacing dialog.
- Increase the size of the screen and try the custom spacing option again.
- Choose a different dominant widget in your selection set.

## Centering Widgets

The Center command treats multiple widgets as a set, and, therefore, their orientation to one another is not changed.



### How to Center Widgets on a Screen

Select one or more widgets. Choose Edit→Center and an orientation, or choose the appropriate Center button on the toolbar.

- Horizontally—Centers the selected widgets between the left and right screen borders.
- Vertically—Centers the selected widgets between the top and bottom screen borders.
- Both—Centers the selected widgets directly in the middle of the screen; this command positions the widgets horizontally and then vertically. (Not available on the toolbar.)

---

## Controlling a Widget's Position at Runtime

---

You can control where a widget is repositioned at runtime in the event that:

- The screen or widget font changes.
- The screen size increases or decreases.
- The screen is ported to another platform.

By default, a widget is positioned along its left edge and vertically along its middle axis. The anchor properties also determine how the widget retains its alignment to other widgets on the screen.

## How to Set a Widget's Position in Relation to Other Widgets

1. Select the widget or widgets.
2. Under Geometry, set the Horizontal Anchor property to one of the following:
  - Left—The default setting. If the widget grows, it remains anchored to its starting column, and only grows to the right.
  - Right—If the widget grows, it remains anchored to its ending or rightmost column, and only grows to the left.
  - Center—If the widget grows, it remains anchored at its center axis and grows equally to its left and right.
3. Under Geometry, set the Vertical Anchor property to one of the following:
  - Top—If the widget grows, it remains anchored to its starting row, and only grows downward.
  - Bottom—If the widget grows, it remains anchored in its ending or bottom row, and only grows upward.
  - Middle—The default setting. If the widget grows, it remains anchored at its middle axis, and grows equally upward and downward.

Alignment and anchoring go hand-in-hand. Left-aligned widgets that have a Horizontal Anchor of Left remain left-aligned. The same is true for all other similar combinations (e.g., right-align and right anchor). If you align widgets by choosing Edit→Align, Panther automatically sets the appropriate Anchor property position.

In addition, if you align widgets to a grid coordinate by choosing Edit→Grid Align, widgets align at their anchor specification along the nearest grid coordinate (refer to “How to Position a Widget on a Grid Coordinate” [on page 9-15](#) for more information).

---

# Undoing Actions

---

Panther keeps track of editing and formatting changes that you make on each open screen. If you inadvertently move a widget or change your mind, you can usually reverse the last several (10 by default) actions you took, including setting changes you make via the Properties window. You can also change the number of undo levels for your authoring environment.

Multiple actions are undone or redone in sequence. However, there are some actions that you can't undo, such as saving a screen or selecting widgets.



## How to Undo an Action

Choose Edit→Undo or the Undo button on the toolbar to reverse the last action. The Undo menu option identifies the last action that can be reversed.

## How to Redo an Action

Choose Edit→Redo or the Redo button on the toolbar to redo the last canceled action. The Redo menu option identifies the last action that can be redone.

## How to Set the Number of Undo Levels

1. Choose Options→Undo Levels. The Configure Undo Levels dialog box opens.
2. Enter a number from 0 to 99 to define the number of levels that can be undone and redone for each screen.
3. Choose OK. This setting takes effect immediately.



# 10 Controlling the Way Things Look

To some degree, when you define the way things look, you also define the way those things act. This chapter describes how you control what the user sees, for example:

- Changing one widget type to another.
- Which font to use for a selected widget or for the screen as a whole.
- How the text in a widget aligns, right or left.
- Null field content and whether data should be displayed or not, or whether the widget itself is displayed or not.
- Creating simple arrays as well as arrays that shift and scroll.
- Specifying a data format to define date and time fields as well as formats for numeric and currency data.
- Assigning a mouse pointer shape for the screen.

---

# Giving Widgets Initial Data

---

You can specify that a widget has initial data or displays some initial text. This data is what displays when the screen first opens; it's what the user sees. Therefore, initial text can be, for example, instructions for the user, a default or suggested selection, or an identifier. The property to enter this information varies by widget type.

## How to Define Initial Text or a Label for a Widget

1. Select the widget.
2. Depending on the widget type, you can define initial text by selecting the appropriate property:
  - For single line and multiline text widgets, list boxes, combo boxes, and option menus—Under Format/Display, select the Initial Text property. For widgets that have multiple occurrences, such as a list box or multiline text, enter data for each occurrence; press Enter after each entry.
  - For scale widgets—Under Input, select the Initial Value property. Enter the starting value which, in turn, defines the slider location on the scale.
  - For labels, including those on push buttons, check boxes, radio buttons, and toggle buttons—Under Identity, select the Label property.
  - For graphs—Refer to “How to Specify Data Values” [on page 13-24](#) for details on specifying values.

A property-specific dialog box opens.

3. You can enter data in the dialog box by doing any of the following:
  - Type the text directly in the window.
  - Choose Editor to invoke your local editor and enter the desired data.
  - Choose Read File to import an external file.
4. Choose OK to close the dialog box and return to the Properties window.

---

# Changing Widget Type

---

Through the editor you can change a widget from one type to another. For example, you can change a single line text widget into a list box, change a vertical scale to a horizontal scale, or a combo box into an option menu.

**Note:** Lines, boxes, graphs, grid widgets, and links cannot be changed to other widget types.

## How to Change Widget Types

1. Select the widget.
2. Under Identity, in the Widget Type property, select the desired type from the Setting field option menu.

The original widget is replaced with a default widget of the specified type. Any properties that you defined for the original widget that are common to the new widget type are applied; however, properties which are not valid for the new widget are lost. Panther attempts to position the new widget in the same place as the original.

Initial text can be lost, particularly in cases where you change a multiarrayed widget, such as a list box, to a widget with a single onscreen element, like a single line text widget or option menu.

---

# Formatting Text

---

Formatting text in a widget includes:

- Deciding whether text is visible or not.

- Determining the text position on the widget.
- Defining a font for the screen and the widgets.
- Defining what displays when a field has a null value.

## To Display or Not Display

Widgets designated as hidden are visible in Edit mode in screen editor, but are not visible in Test mode or at runtime.

## How to Hide a Widget at Runtime

1. Select the widget (except static labels).
2. Under Identity, specify the visibility state in the Hidden property:
  - Yes—Initializes the widget in an invisible state. You can programmatically make the widget visible at runtime.

For example, on a logon screen, you might initially hide a password field to those users who, in fact, do not require passwords. However, if the user's name indicates that a password is required, you can change the Hidden property (refer to “Properties” on page 19-40 in *Application Development Guide* for information on changing properties at runtime).

- Always—Specifies that the widget remain hidden always. This setting can not be changed at runtime. In addition, at runtime, Panther does not create “always” hidden widgets, therefore, performance is not encumbered with unnecessary widget creation and screen painting. Screen space is not allocated for these widgets, so when screens are ported to other platforms, the widgets require no special resizing considerations.

Widgets designated as always hidden are useful for storing data that are used in screen processing.

**Note:** A hidden widget, whether the property is set to Yes or Always, is protected from user input and is not included in the tabbing order while it is invisible. If a grid widget is designated as hidden, all members of the grid are hidden as well.

- No—Initializes the widget in a visible state at runtime. This is the default setting for all widgets (except links which are always invisible). Visible widgets can be made invisible at runtime.

**Note:** An array designated as the row titles (Row Titles property is set to First Column) in a grid widget is not affected by the Hidden property setting; that is, the rows' titles are always displayed regardless of the Hidden property setting for that column.

## How to Hide the Contents of a Field

1. Select the single line (including those within a grid widget) or multiline text widget.
2. Under Format/Display, set the Password Field property to Yes. The Password Char subproperty is displayed.

All initial text (as specified in the Initial Text property) is made invisible. At runtime, the field can be seen, however, characters typed or accepted into the field are not displayed. This property is useful for creating password fields.

3. (Optional) In the Password Char subproperty, enter a single character that you want to display when the user types.

For example, if you enter \*, each character the user types in the field is represented by an asterisk. All initial text (as specified in the Initial Text property) is replaced with the specified password character.

## Justification

In the English language, characters are usually entered from left to right; they are left-justified. By default, the characters in all widgets are left-justified, except for toggle buttons and push buttons, which have a centered label.

You can change the text justification on the following widget types: single line and multiline (without a word wrap specification) text widgets, dynamic labels, option menus, combo boxes, radio buttons, check boxes and list boxes.

## How to Change Text Justification in a Widget

1. Select the widget.

2. Under Format/Display, specify the desired position in the Justification property:
  - Left—(default) Data is entered from left to right in data entry widgets. On widgets that take label text (dynamic labels, option menus, combo boxes, radio buttons, check boxes, and list boxes), the text is aligned to the widget's left border.
  - Right—Data is entered starting at the rightmost position of the field. To get the best results with right -justified fields, use monospace (fixed width) fonts.

## How to Control the Text Justification on Box Widgets

1. Select a box widget.
2. Under Identity, enter a title for the box in the Label property. The Justification subproperty is displayed.
3. Specify the desired position for the text: Left, Right, or Centered (default).

## Aligning Button Labels in Motif

In Motif, you can change the default alignment of label text by making the appropriate entries in the resource file for your application. By default, all push button and toggle button labels are centered:

```
Prolifics*area*XmPushButton.alignment:ALIGNMENT_CENTER  
Prolifics*area*XmToggleButton.alignment:ALIGNMENT_CENTER
```

To left justify all push button and toggle button labels, set these resources to ALIGNMENT\_BEGINNING:

```
Prolifics*area*XmPushButton.alignment:ALIGNMENT_BEGINNING  
Prolifics*area*XmToggleButton.alignment:ALIGNMENT_BEGINNING
```

To right justify all push button and toggle button labels, set the resource values to ALIGNMENT\_END:

```
Prolifics*area*XmPushButton.alignment:ALIGNMENT_END  
Prolifics*area*XmToggleButton.alignment:ALIGNMENT_END
```

To left justify all push button labels on a specific screen, use the following resource specification in your resource file:

```
Prolifics*screen_name*area*XmPushButton.alignment:
ALIGNMENT_BEGINNING
Prolifics*screen_name*area*XmToggleButton.alignment:
ALIGNMENT_BEGINNING
```

For example, to left justify all push button labels on the screen named `myscr`, set the following resource:

```
Prolifics*myscr*area*XmPushButton.alignment: ALIGNMENT_BEGINNING
```

To align the label on an individual button on a screen, use the following resource specification in your resource file:

```
Prolifics*screen_name*area*button_name.alignment:
alignment_specification
```

For example, to right justify the label on the widget `mybutton1` only when it appears on the screen `myscr`, set the following resource:

```
Prolifics*myscr*area*mybutton1.alignment:ALIGNMENT_END
```

To align the label on a button throughout your application, use the following resource specification in your resource file:

```
Prolifics*area*button_name.alignment:alignment_specification
```

For example, the following resource specification will center the label on `mybutton2` wherever it appears throughout the application.

```
Prolifics*area*mybutton2.alignment:ALIGNMENT_CENTER
```

## Specifying Fonts

You can use fonts in your application that are:

- GUI-specific—Use a font name that is associated with your operating system. Each platform has its own style and resources for identifying typefaces.
- Panther-specific—Predefined (in the configuration map file) aliases that map to platform-specific fonts. In the event that your application is ported to other platforms, using these fonts ensures that they are portable as well.

When you create a screen, a default font is applied, derived either from a Panther distributed configuration map (`cmap`) file or from your GUI platform. Under the Font heading in the Properties window, all font-specific properties are set to Default for the

screen and for any widgets you add to the screen (except for those that have a font assignment of their own, because the widget is a copy or inherits its property values from a repository entry).

The Default setting means that at runtime Panther resolves which fonts to apply based on what is set at a higher level. Therefore, a widget's Default specification means it uses the screen font assignment. The screen's Default specification means it uses the application-wide specification.

You can define fonts at any of the following levels:

- As the application default—An application-wide font is defined in the `default_font` assignment of your `cmap` file (provided with your installation and specific to your terminal type). In the absence of an application default specification in the `cmap` file, the system default is used. For information on assigning an application-wide font, refer to “Default Font” on page 45-37 in *Application Development Guide*.
- As a screen-wide specification—The font defined for the screen as a whole overrides the system or application default font. In the absence of any other specification, this font setting is used by all widgets on the screen. The screen's grid size adjusts to accommodate the font specification.
- On a widget-by-widget basis—This widget-level specification overrides any defaults set at a higher level. The widget adjusts in size to accommodate the font specification.

grid widget members can each have a unique font specification. The font defined for the grid widget itself determines the font used for the column titles and row titles.

**Note:** Under Motif, the individual members of a grid as well as the column titles and row titles acquire the font defined for the grid widget as a whole.

Graph widgets have their own set of font name specifications (prefixed with the word Graph) in addition to Prolifics fonts and system fonts. A single font specification applies to the entire graph. The point size for graph components, such as graph titles, subtitles, legends, and labels, is defined under the Graph heading in the Properties window. Refer to “Controlling Graph Text” on page 13-7 for details on defining graph fonts.



## Assigning a Prolifics Font

Prolifics fonts include several standard font styles: Courier, Helvetica, Symbol, and Times Roman, etc. These are defined in your Panther configuration map file, and resolve appropriately to a GUI-specific font for your platform. Depending on the platform and the font selection, a point size and other display attributes are automatically assigned. (For details on editing the `cmap` file, refer to “Configuration Map File” [on page 45-25](#) in *Application Development Guide*.)

### How to Assign a Font to a Screen or to One or More Widgets

1. Select the screen or widget.
2. Under Font, specify the desired typeface in the Font Name property by doing one of the following:
  - Select a name from the Setting field drop-down list. The list includes:
    - Panther-specific fonts (begin with the word Prolifics) mapped to local system-specific fonts.
    - Locally installed fonts (such as TrueType fonts, display fonts, and any other installed fonts on your system) listed alphabetically.
    - Graph-specific fonts if the selected widget is a graph.

If the selected font has a default point size and other attributes associated with it, the other font subproperties are set accordingly, otherwise they remain set to Default and acquire their value from the container (screen for widgets, application/system font for screens). Proceed to step 3 to assign a point size and to steps 4 through 6 for other attributes.

- Choose the More button. A GUI-specific font selection dialog box opens. (Refer to “Assigning a GUI Font Name” [on page 10-11](#) for details on selecting a GUI-specific font.)

**Note:** If your setup does not point to a configuration map file, the Setting field option menu in the Properties window has only the Default value. You can type a font name. Panther will attempt to resolve the Font Name specification at runtime.

3. Under Font, specify the desired size of the typeface in the Point Size property. The drop-down list of font sizes includes point sizes that are available for the specified font and are defined in your `cmap` file, plus Default.

Under Windows, you can also enter a value if the font is scalable.

**Note:** If you specified a font that includes default attributes, such as bold, italic, or underline, the following steps may have no effect on the way the font looks.

4. Under Font, set the Bold property to one of the following:
  - Yes—Displays text in an increased weight in the specified font, making it appear bold.
  - No—Displays text in a non-bold style.
  - Default—Uses the specification of the container (the screen specification if your selection is a widget; the application/system specification if you have the screen selected).

**Note:** The Bold property has no effect if the specified Font Name does not have a bold qualifier or if it is only available in a bold style. For example, if you select Bookman Old Style (by default a bold font), setting the Bold property to No will not change the weight of the font.
5. Under Font, set the Italic property to one of the following:
  - Yes—Displays text in the specified font in an italicized style.
  - No—Displays text in a non-italicized style.
  - Default—Uses the specification of the container (the screen specification if your selection is a widget; the application/system specification if you have the screen selected).

**Note:** The Italic property has no effect if the specified Font Name does not have an italic qualifier or if it is only available in an italic style. For example, if you select Brush Script MT (by default an italicized font), setting the Italic property to No will not remove the slant.
6. Under Font, set the Underlined property (which has no effect under Motif) to one of the following:
  - Yes—Underlines text that displays on the widget or screen.
  - No—Does not underline text.
  - Default—Uses the specification of the container (the screen specification if your selection is a widget; the application/system specification if you have the screen selected).

---

## Assigning a GUI Font Name

GUI-specific fonts are the least portable. Your specifications will reside with the screen and you should map them as needed in the configuration map file. Selecting a font in this way, as described here, allows you to choose attributes that may not be included in your `cmap` file and see a sample of how the selected font looks.

### How to Assign a GUI-Specific Font

1. Select the screen or widget.
2. Under Font, select the Font Name property.
3. Choose More. A GUI-specific font selection window opens.
  - Under Windows—The Control Panel Font dialog box is displayed. You can select a font name and any other attributes.
  - Under Motif—A Selection Window dialog box is displayed.
    - To display all available fonts: Overwrite any string in the Filter field with an `*` (asterisk) wildcard. Then choose Filter. All X Logical Font Descriptions (XLFD) on your system are displayed.
    - To display a partial list: Enter a partial string including a wildcard (`*`) in the Filter field, for example `*helvetica*` will display all Helvetica fonts on your system, while `-adobe*` will display all Adobe foundry fonts.

Refer to your X documentation for the syntax used to identify fonts on your X server.

4. Select the desired font. The selection window shows a sample of how the font appears.
5. Choose OK from the font selection window once you have made your selection. You resume in the Properties window.
6. Choose OK on the Properties window to confirm your selection.

Under Windows, Panther sets the point size and other style options to Yes in the corresponding properties—Bold, Italic, and Underlined. If the styles do not apply, the properties remain set to Default.

## Displaying Null Values

To distinguish between when a field is null and when it has data that is not visible or available (for example, a field that is blank), you can specify that it should display a null indicator string when no data is present. A field is considered null if both of the following conditions are met:

- Its Null Field property under Format/Display is set to Yes.
- It contains no data.

When a field is null, the null indicator string is displayed in the field, and the field is, by default, selected on entry. The null indicator string is cleared when the user enters data.

The null indicator is defined as blank in the `SM_NULLLEDIT` variable in your Panther message file. If you prefer a different default null indicator, you can change the value of `SM_NULLLEDIT`. (For information on editing the message file, refer to “Creating and Modifying Message Files” on page 45-2 in *Application Development Guide*.)

To distinguish a blank field from an empty (null) one, you can either:

- Change the `SM_NULLLEDIT` variable to something other than blank, or
- Define a Null Text indicator for those widgets that must be distinguished as null.

## How to Allow a Widget to Accept and Display a Null Value

1. Select the widget.
2. Under Format/Display, set the Null Field to Yes. Additional null-related subproperties are displayed.
3. In the Null Text property, enter a character string (up to 256 characters long) that will be displayed at runtime when the field is empty or null.

For example, enter an x, an \* (asterisk), or null.

4. (Optional) To fill the field with the specified indicator string, set the Repeating property to Yes.

Whatever character or string you have entered in the Null Text property will fill the field to its maximum length. The Repeating property has no effect if you do not specify a Null Text property.

Widgets that have a Null Field setting are made null at runtime if any of the following occurs:

- The user clears the field by choosing the FERA (clear field) key.
- The user enters the null indicator in the field.
- The field has no initial content and is not populated with data from another source (LDB or database).

Use the library function `sm_null` to determine whether or not a field is null. The function `sm_getfield` returns the content of the field, which is the null text string (possibly replicated) if the field is, in fact, null.

## Controlling Grid Formats

To control the appearance of data in grid widgets, you can set the following properties:

- For grid widgets, set the Row Margin property, under Geometry, to control the space between rows.
- For grid columns, set the Column Click Action property, under Format/Display, to define the behavior that occurs when a user clicks on a grid column heading.

---

# Creating Shifting/Scrolling Fields

---

A widget must occupy at least one position on the screen. However, a widget can be larger, in two dimensions, than its onscreen capacity; a field can shift and scroll. Shifting extends a field's widget beyond its onscreen width. Scrolling permits a field to hold more data items that will fit onscreen.

## Creating an Array

Each widget is an array, although for the most part, an array of one occurrence. This is the base field. You increase the array to include additional occurrences by specifying that the widget has more than one element; the base field is the first element of the array.

### How to Create an Array with More than One Occurrence

1. Select the widget.
2. Under Geometry, set the property appropriately for the widget type:
  - Set the Array Size property to the desired number if the widget type is a single array widget (for example, single line text or dynamic label). Additional array-related properties are displayed.
  - Set the Onscreen Rows property to the desired number if the widget type, by default, has more than one occurrence (for example, list boxes and multiline text widgets).
3. (Optional) In the Spacing subproperty, enter the amount of space desired between each occurrence of the array. Refer to Table 9-1 [on page 9-7](#) for a list of valid units of measurement.
4. (Optional) To create a horizontal array, set the Horizontal subproperty to Yes. If the current screen size cannot accommodate the horizontal array, it resizes at runtime to best accommodate all or as many elements of the array as can display (providing scroll bars to view offscreen widgets). For information on controlling a screen's viewport, refer to “Displaying Screens” [on page 18-3](#) in *Application Development Guide*.

Each element of an array is assigned a unique field number, but shares its properties with the base field. You assign a name to the array as a whole, and not for each element. The array slots are called occurrences. The elements of the array are the mechanism through which the array's occurrences are viewed. A scrolling array can have more occurrences than elements. A simple array has the same number of occurrences as elements.

## Creating a Scrolling Array

Any widget that has either the Onscreen Rows property or Array Size property can be defined as a scrolling array. When you specify that a widget has a greater number of occurrences than can be displayed on the screen, the widget is referred to as a scrolling array.

For details on creating synchronized arrays, refer to [page 8-20](#), “Synchronizing Scrolling Arrays.”

### How to Define Offscreen Occurrences for a Selected Widget

1. Under Geometry, set the Scrolling property to Yes. Additional scrolling-related properties are displayed.
2. Under Scrolling, in the Max Occurrences subproperty, enter the total number of occurrences. By default, Max Occurrences is set to be equal to the number of onscreen occurrences. Set the property to blank to allow for the maximum number of occurrences allowed for the operating system/database driver.

The Max Occurrences value must be greater than or equal to the number specified in the Array Size property or Onscreen Rows property in order to scroll.

3. (Optional) Under Scrolling, in the Scroll Increment property, specify the number of occurrences by which the array should scroll when the user presses Page Up or Page Down. The default is one less than the number of onscreen occurrences.
4. (Optional for multiline text and list box widgets) Under Scrolling, the Vert. Scroll Bar property is, by default, set to Yes; set the property to No if you don't want vertical scroll bars.
5. (Optional) Under Scrolling, set the Circular property appropriately:
  - Yes—Causes the first element to redisplay after the last element in the array is reached, and vice versa.
  - No—(default) If the user uses the arrow keys to move the cursor to the last/first element, the cursor moves to the next or previous field. If the user presses Page Down/Up to scroll the occurrences, Panther displays the message "No more data" when the first/last element is reached.

**Note:** To disable the `No more data` message, set the message file entry `SM_MOREDATA` to the empty string `" "`.

## Creating a Shifting Field

If the maximum length of a widget is greater than the display length, the widget is referred to as a shifting field. By default, single line and multiline text widgets are shifting fields.

### How to Create a Shifting Field for a Selected Widget

1. Under Geometry, set the Length property to the number of columns of data that the widget can display at a time.  
**Note:** This number does not necessarily define the number of characters that are displayed, especially if your screen or widget uses a proportionally spaced (varied-width) font.
2. Under Geometry, in the Max Data Length property, enter a number greater than the Length property. This number defines the maximum number of columns that the widget can hold. Choose OK. Additional properties are displayed.
3. (Optional) Under Max Data Length, in the Shift Increment subproperty, enter the number of characters by which the data should be shifted when the user moves the cursor beyond the onscreen length.
4. (Optional for multiline text and list box widgets) Under Max Data Length, set the Hor. Scroll Bar property appropriately:
  - Yes—(default) If you want to provide horizontal scroll bar.
  - No—Display the widget without a horizontal scroll bar. The user can use arrow keys to shift the data in the field.



# Creating a Date and Time Field

You can assign a date and/or time format to almost any widget type (except static labels and scales). You can enforce a format for user-entered dates or times, or specify the format in which the system date or time is displayed.

## How to Set a Date/Time Format for a Selected Widget

1. Under Format/Display, specify Date/Time in the Data Formatting property. Additional date/time-specific properties are displayed.

**Note:** By default, the property is set to None. However, if the widget is derived from a database column and has a date/time edit, Panther sets the Data Formatting property to Date/Time.

2. In the Format Type subproperty (which for dates is `date_format` at runtime), select one of the following:
  - One of the ten predefined date/time formats (defined in the Panther message file). For information on creating custom defaults, refer to “Creating Date and Time Defaults” on page 45-16 in *Application Development Guide*.

Set Format Type to:	to get the following format:
MON/DATE/YR4 HR:MIN2	7/18/2016 10:06
MON/DATE/YR4	7/18/2016
MON/DATE/YR2 HR:MIN2	7/18/16 10:06
MON/DATE/YR2	7/18/16
HR:MIN2	10:06
DEFAULT5 to DEFAULT9	same as MON/DATE/YR2 HR:MIN2

- Custom (Refer to “Defining a Custom Date/Time Format” on page 10-18 for details.)
3. In the System Update property, specify whether or not the date/time is updated by the system.
    - No—Indicates that the user is expected to enter the data. Skip to step 5.
    - Yes—Indicates that the system updates the date/time. Additional properties are displayed.
  4. In the Frequency subproperty, specify how often the system date/time is updated. Indicate time interval in seconds. The default is zero.

If the interval is set to zero, the system date/time is updated only under the following conditions:

- When the screen first opens.
  - Panther receives a clear field command or the FERA key is pressed (and the field is not protected from clearing) while the cursor is on the field.
  - Panther receives a clear all fields command or the CLR key is pressed (and the field is not protected from clearing).
  - A new array occurrence is allocated.
5. In the Clock Type subproperty, specify a 12-hour or 24-hour clock.

**Note:** If you specify a 12-hour clock, consider a custom format that includes an AM/PM indicator.

## Defining a Custom Date/Time Format

In addition to the default settings that you can specify by name, you can format a date or time string to suit your needs by using combinations of literal strings and Panther-specific substitution variables (refer to Table 10-2 on page 10-19).

A literal character forces entry of that particular character. A substitution variable forces entry of a string that matches the specified format. For example, if you enter a Custom Format setting of `HR:MIN:SEC`, a runtime entry of `09:19:21` is accepted.

To embed punctuation, such as spaces, slashes, or colons, enter literal characters as part of the Custom Format setting. Table 10-1 illustrates samples of date/time format strings and input that is considered valid.

**Table 10-1 Examples of date/time format strings**

<b>Format String</b>	<b>Acceptable Input</b>
MON2/DATE2/YR2	03/03/89
DATE2/MON2/YR4	19/09/1956
DAYL MONL DATE, YR4	Saturday December 26, 1954
HR:MIN2AMPM	2:04PM

Table 10-2 lists the substitution variables you can use for customizing date/time formats.

**Table 10-2 Substitution variables for customizing date/time format**

<b>Substitution Variable*</b>	<b>Input</b>
DATE	day of month
DAYA	abbreviated day name
DAYL	long day name
WEEKDAY	numeric day of the week
MON	month number
MON2	month number, zero filled
MONA	abbreviated month name
MONL	long month name
YR4	4 digit year
YR2	2 digit year
YDAY	day of year
HR	hour
HR2	hour, zero filled

**Table 10-2 Substitution variables for customizing date/time format**

Substitution Variable*	Input
MIN	minute
MIN2	minute, zero filled
SEC	second
SEC2	second, zero filled
AMPM	am or pm

\*Substitution variables must be entered in upper case.

---

## Defining a Numeric Format

---

You can display data as a string of numbers with the appropriate punctuation, like a comma and decimal point, or assign the appropriate currency characters to display monetary data. The format you define is applied to the data during field validation, and non-numeric data is discarded.

### How to Set a Numeric Format for a Selected Widget

1. Under Format/Display, in the Data Formatting property, specify Numeric. Additional number-specific subproperties are displayed.
2. In the Format Type subproperty (which for Numeric is `numeric_type` at runtime), select either of the following settings:
  - One of the ten predefined numeric formats (defined in the Panther message file). Table 10-3 shows examples of how input data is displayed according to the predefined format. Skip to Step 6.

For more information, refer to “Creating a Default Numeric Format” on [page 45-21](#) in *Application Development Guide*.

- Custom. Additional numeric-specific properties are displayed.

**Table 10-3 Default numeric formats in the message file and sample output**

Entered Data	Local currency	2 dec w/comma	0 dec w/comma
12345	\$12,345.00	12,345.00	12,345
1234.56	\$1,234.56	1,234.56	1,235
123.456	\$123.46	123.46	124

3. (Optional) If you specify Custom in the Format Type property, set the Min Decimals and Max Decimals subproperties to a value (both default to 2) between 0 and 9, inclusive.

If Min Decimals is greater than the Max Decimals specification, numbers entered in the fields are rounded to the nearest whole number, and the number of decimal places specified is padded out with zeroes. For example, if Max Decimals is 0 and Min Decimals is 2, entering the number 22.546 displays as 23.00.

4. Define numeric and/or currency punctuation to be used in your custom format of the data.
  - In the Decimal Symbol subproperty, select the desired radix symbol: comma or period (default).
  - In the 000 Separator subproperty (`thousand_sep`), select the desired punctuation for indicating a thousandths separator: comma (default), period, Blank, or None.
5. (Optional) To display the data as monetary output, specify the desired currency format.
  - In the Currency Symbol subproperty, enter a one- to five-character symbol, such as a dollar sign (\$). If you do not enter a symbol, none will be used.
  - In the Placement subproperty, specify Left (default), Middle, or Right to indicate where the currency symbol should be positioned. If you select middle, the currency symbol appears where the decimal/radix symbol would appear, so it overrides the Decimal Symbol property specification.

6. In the Rounding property (a subproperty of a Numeric setting in the Data Formatting property), specify Round Up, Round Down, or Round Adjust (default is to round up above 5).

Refer to Table 10-4 for examples of how rounding options work.

**Table 10-4** Examples of rounding options

Number Entered	Round Up	Round Down	Round Adjust
2.056	2.06	2.05	2.06
-2.056	-2.05	-2.06	-2.06
2.055	2.06	2.05	2.06

7. Define what should occur if the numeric data is blank or equal to zero.
    - (Optional) In the Fill Character subproperty, enter a character, for example, an asterisk, if you want leading or trailing blanks to be replaced with a specific character. The character pads the field to its maximum length.
    - In the Zero Format subproperty, indicate if you want the specified format applied to the data when it is zero:
      - Yes—(default) Applies the format to data that is equal to zero, and the field remains empty (blank) if no data is entered.
      - No—The field remains empty if the data equals zero or no data is entered.
    - In the Empty Format subproperty, specify if you want the specified format to be applied even when the field is empty.
      - Yes—Applies the format to the field even when it is empty. For example, at runtime, when the field or screen is validated, if data was not entered in the field, the defined format is displayed—\$0.00 for instance.
      - No—(default) The field remains empty (blank) if no data is entered.
- Note:** The Empty Format subproperty has no effect unless the Zero Format property is set to Yes.

---

# Giving Screens a 3D Appearance

---

Panther allows you to provide a three-dimensional appearance to your screens and the widgets on those screens when you are running Panther or a Panther application under Windows.

**Note:** If you are running Panther under Motif, which provides native support for three-dimensional widgets, use the settings in your Motif environment to enable or disable the 3D feature.

## How to Implement the 3D Feature for a Screen

Select the screen. Under Identity, set the 3D property to one of the following:

- Yes—Displays the screen and its widgets in 3D.
- No—Does not display the screen and its widgets in 3D.
- Application—(default) Displays the screen according to the application-wide 3D setting defined in the initialization file (.ini file). Refer to [3D on page 3-4](#) in *Configuration Guide* for a description of the 3D option in your Windows .ini file.

If the 3D option is enabled in the .ini file for your application, message boxes and Windows common dialog boxes also take on a three-dimensional appearance. If this initialization option is not set, only those application screens that have the 3D screen property set to Yes appear in 3D.

The 3D screen property setting is ignored when Panther is running on a platform other than Windows. Thus, if your application will run on multiple platforms, Windows among them, set the 3D property to the value that will give the desired appearance under Windows; when the application runs on any other platform, screen appearance is unaffected.

## Three-Dimensional Screen and Widget Appearance

This section describes how screens, and the widgets on those screens, appear when the 3D feature is enabled under Windows.

### Screens

The background color of 3D screens and dialog boxes is determined by the Button Face color setting on the Windows control panel, and overrides the background color specified in the Properties window for the screen.

Dialog box title bars appear indented; title bars on non-dialog screens do not have a 3D appearance.

Screen size might increase slightly if additional space is needed to accommodate the 3D effects imposed on the screen's widgets.

### Widgets

The following widgets *do not* change appearance when 3D is enabled:

- Static and dynamic labels
- Scales
- Graphs
- grid widgets (the row and column title areas appear in 3D by default)
- Push buttons and toggle buttons (appear 3D by default)

All other visible widget types change in appearance, as described below. Unless otherwise noted, all widgets affected by the 3D setting take on the same background color as the screen on which they appear (that is, the Button Face color setting on the Windows control panel).

The amount of screen real estate needed by 3D widgets might increase slightly because of the border used to create the three-dimensional effect.

Text and multiline text widgets appear indented into the screen, with a 3D border around the entire widget.



List boxes appear indented into the screen, with a 3D border around the entire widget. Scroll bars are the screen background color if active; if inactive, they are the Window Background color, as set on the Windows control panel.

Option menus appear indented into the screen, with a 3D border surrounding both the text and the drop-down area.

The text area of the combo box appears indented into the screen. A 3D border surrounds the text area only and not the drop-down button.

The selection area on check boxes (square) and radio buttons (circle) appears indented into the screen; its color is the Window Background color, as set on the Windows control panel. The selection indicator (X or dot) is black. The label text does not appear 3-dimensional.

If the Line/Box Style property is set to Default, the line or box appears in the Etched In style. If it is set to any other style, the line or box is unaffected by the screen's 3D setting.

Boxes with a Line/Box Style property set to Default or Etched In take on the background color of the screen; if the Line/Box Style is set to any other value, the background color setting in the Properties window remains in effect.



# 11 Specifying Colors

Regardless of your terminal type or your development environment, you can design applications using all the colors provided by Panther (Panther's basic colors) as well as those that are unique to your platform.

All objects have default foreground and background colors described by an external color scheme. Panther provides a default scheme that specifies colors for each object. You can assign and change your application's colors by altering the scheme or by editing the objects' color properties.

**Note:** If your application is running under Windows and you have the 3D option enabled, screen and widget background colors are determined by settings on the Windows Control Panel. Refer to “Giving Screens a 3D Appearance” on [page 10-23](#) for an explanation of how the 3D feature affects screen and widget appearance.

---

## Color Property Types

---

There are three color types from which you can choose to define the colors for widgets and for the screen's background. Each color type offers its own variety of color and can ensure portability from one platform to another without having to reassign colors.

## Using Panther Basic Colors

Panther's basic colors are defined for use on all platforms. They include eight highlighted and eight unhighlighted colors plus a Container option. The Container option applies only to background color specifications and causes the selected widget to acquire its background color from the underlying object. For example, a screen is a container for a widget; therefore, a widget with a color specification of Container has the screen's background color.

The scheme defined in the distributed configuration map (`cmap`) file specifies that most objects have a background color of Container. Therefore, unless the `cmap` file has a different specification, all new objects start with their background matching the underlying background.

## Scheme Color Specification

The Scheme color type maps to a predefined set of color specifications that determines the foreground (and, sometimes, background) colors for each object type (for example, screens, push buttons, multiline text widgets, etc.) on an application-wide basis. All new widgets have an FG and BG Color Type property setting of Scheme.

When an object has Scheme defined as its color type, Panther determines what color the object should be by first looking in the platform-specific `cmap`, which contains definitions for objects' background and (if applicable) foreground colors. If the `cmap` file does not define a scheme or a specific object type's color property, the native environment's default color scheme is used. The environment's scheme is as follows:

- In character mode Panther, the scheme is a white foreground on a black background.
- In Windows, the Control Panel color specifications are used.
- In Motif, the `*fg` and `*bg` color resource settings are used.

The `cmap` file can also refer to the native scheme. For information on how to alter the scheme in the configuration map file, refer to “Defining Color Schemes” [on page 45-30](#) in *Application Development Guide*.

## Extended Color Specification

The Extended color type specification allows you to assign a GUI-specific color to an object. This setting allows you to assign a color beyond the Panther basic colors—either GUI-specific colors or GUI-independent color aliases. The alias colors are resolved in the `Colors` section of your configuration map file.

---

# Changing Color Properties

---

In the screen editor, you specify colors by either:

- Setting Color properties in the Properties window, or
- Selecting colors from the Color palette.

## Setting Color Properties

### How to Change an Object's Color Via the Properties Window

1. Select the screen or widget.
2. Under Color, set the FG (not available as a screen property) and/or BG Color Type property to one of the following types:
  - Scheme (continue to step 5).
  - Basic. The Color Name subproperty is displayed (continue to step 4).
  - Extended. The Color Name subproperty is displayed (continue to step 3).
3. In the Color Name subproperty, do one of the following to designate a GUI-specific color:
  - Select a color specification from the Setting field drop-down list. The list includes color alias names as defined in your configuration map (`cmap`) file.

- Choose More. A GUI-specific color selection dialog box opens.
  - Under Motif, you can select a literal color by name, for instance, `DarkOliveGreen`.
  - Under Windows, you can choose or define a color in the Windows Color dialog box. This sets the Color Name property to an RGB value.
- Type a numeric RGB value—for example, `255/128/0`.
- Type a GUI-independent color alias, like `myGreen`, which would be defined in the `Colors` section of your `cmap` file as, perhaps, `0/128/0`. When running on Windows, any object with a `myGreen` color assignment will map to a Windows-specific green color. Whereas, when your application runs on Motif, you might change the configuration map file to map `myGreen` to Panther's basic `GREEN`.

Continue to step 5.

4. In the Color Name subproperty, choose from the list of Panther basic colors: black, blue, green, cyan, red, magenta, yellow, white, brightblack, brightblue, brightgreen, brightcyan, brightred, brightmagenta, brightyellow, and brightwhite. Container is an additional option on the list of background color names.

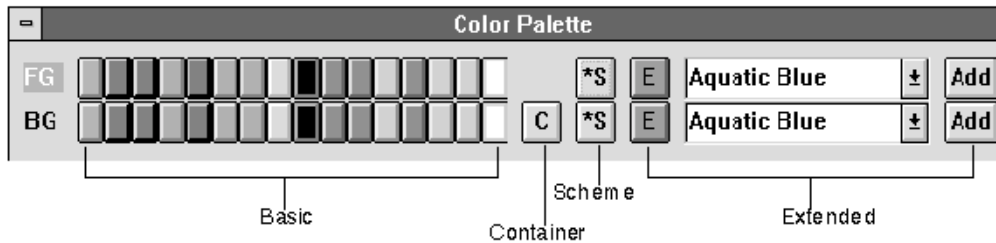
Additional display attribute properties are displayed when you choose an FG Color Type of Basic. For details on these properties, refer to [page 11-6](#), “Setting Display Attributes.”

5. Choose OK.

## How to Set an Object's Color with the Color Palette

You can choose any of the colors displayed on the palette.

1. Choose View→Color Palette. The Color palette opens in the workspace.



**Figure 11-1** To assign colors, you can choose any color that is displayed on the Color palette.

2. Select the widget or screen.
3. Click on the desired foreground and background color button on the palette.

Your selected object changes color and an asterisk is displayed on the specified colors on the palette. The Color Type and Color Name property values are updated appropriately.

To assign a color, you can:

- Select a Panther basic color (sixteen color buttons, from left to right).
  - Choose the C (container) button, located to the right of the basic colors, to assign the background color of the underlying container to the selected widget.
  - Choose the S button (scheme indicator) to assign the object's default scheme as defined in your `cmap` file. If it cannot be resolved there, Panther uses the native environment scheme. For more information on defining color scheme, refer to “Defining Color Schemes” on page 45-30 in *Application Development Guide*.
  - Choose the E (extended) button to assign the color as displayed on the button and named on the corresponding option menu (the name is either an alias or GUI-specific color as defined in your `cmap` file). To assign a different extended color, go to step 4.
4. To assign a different extended color than is indicated on the E button:
    - Select the foreground/background option menu, and select a color from the drop-down list. The option menus list any colors that are defined in the `Colors` section of your `cmap` file. They can be listed as GUI-specific colors or GUI-independent color aliases.

The E button adopts the color you selected.

- Choose the E button to assign the color to the selected object.

## How to Add an Extended Color that Is Not Listed

Extended colors that you add within the screen editor are temporary. They remain on the list for the duration of your editing session.

1. Choose the appropriate Add button (foreground or background) to the right of the option menus. The Add Extended Color Name dialog box opens.
2. Enter the local GUI color—by name (literal or alias) or RGB number.
3. Choose OK.

To ensure that the specified color is portable to other environments, assign a native color to the alias in the platform-specific configuration map file. Refer to “Defining Color Aliases” [on page 45-26](#) in *Application Development Guide* for information.

---

# Setting Display Attributes

---

You can assign other display attributes—reverse, blink, and dim—to all widget types, including lines and boxes, and for screen borders (seen in character mode only). You can set any of these properties on any terminal type, even those where the property is not supported. Panther attempts to simulate attributes. Almost all terminal types support reverse video; however, it is handled differently from terminal to terminal.

**Note:** The Blink and Dim attributes are not supported under the GUI platforms.



## How to Access Additional Display Attributes (Character Mode Only)

For borders in character mode screens and widgets, you can set additional display attributes.

1. Under Color, set the FG Color Type property to Basic. Additional color-specific properties are displayed.

**Note:** Display attributes only apply to the foreground.

2. Select a Panther basic color in the Color Name property and choose OK.
3. Under the Color Type property, set one or more of the display attribute subproperties by setting their values to Yes.
  - Reverse—Displays the text or border in reverse video.
  - Blink—Causes the text, label, or screen border to blink.
  - Dim—Displays the text, widget label, or screen border foreground at a low intensity.



# 12 Providing Help Facilities

To provide assistance to users of your application, you can include:

- Status line text—Information that describes the screen or any widget that currently has focus.
- Panther help screens—Internal Panther screens that display as a window in your application when the user chooses the Help menu option (or `HELP` key).
- Selection screens—Screens that display recommended entries from which the user can choose when the `ITSEL` (Item Selection) key or Help is chosen.
- Table lookup—Non-displaying screens that contain acceptable values that are used to validate the field.
- External help source—An external help file that can be accessed from within Panther via context identifiers associated with screen-level and/or widget-level help requests.

---

## Determining the Level of Help Requests

---

When a user requests help (by choosing Help) on a field that has focus, Panther proceeds as follows:

1. Determines if the External Help property has a context identifier. If it does, that help engine is invoked and the external help screen is displayed.
2. If there is no External Help property setting, Panther determines if the Help Screen property names an internal help screen. If it does, the help screen is displayed.
3. If there is no Help Screen property setting, Panther determines if the Selection Screen property names a selection screen. If it does, the selection screen is displayed.
4. If there no selection screen associated with the field, screen-level help is displayed.

If you do not define field-level help, you can include screen-level help. When a field has focus and no help is defined for it, Panther checks for screen-level help as follows:

1. Determines if the External Help property has a context identifier. If it does, that help engine is invoked and the external help screen is displayed.
2. If there is no External Help property setting, Panther determines if the Help Screen property names a help screen. If it does, the help screen is displayed.

---

## Providing Status Line Text

---

You can assign status text to screens and to any widgets that can get focus at runtime. Field-level status text is displayed when the cursor is in a field that has status text attached to it; status text associated with a screen is displayed as long as the cursor is not in a field that has its own status text.

You can embed code sequences in the status text that:

- Specify text display attributes.
- Display the physical key name assigned to a Panther logical key.
- Cause the terminal to beep.

These embedded codes are interpreted at runtime.

To define status text for a selected widget or screen, enter the text and any desired code sequences directly in the Status Line Text property under Help in the Properties window

The codes are described briefly below; for a more complete explanation, refer to the JPL `msg` command [on page 2-37](#) in *Programming Guide*).

## Adding Display Attributes to Status Text

To apply a display attribute to status text, include the string `%Annnn` anywhere in the status text. `nnnn` is a four-digit hexadecimal number; the corresponding display attribute is then applied to the remainder of the text, or until another `%A` is encountered. To combine attributes, add the numbers together in hexadecimal. The possible values for `nnnn` and the corresponding attributes are shown in Tables 52 and 53.

**Table 12-1 Status text attribute codes**

Attribute	Hex Code	Attribute	Hex Code
Non-display	0008	Blink	0040
Reverse	0010	Highlight	0080
Underline	0020	Dim (low intensity)	1000

**Table 12-2 Status text color codes**

Foreground Color	Hex Code	Background Color	Hex Code
Black	0000	Black	0000
Blue	0001	Blue	0100
Green	0002	Green	0200
Cyan	0003	Cyan	0300
Red	0004	Red	0400

**Table 12-2 Status text color codes**

Magenta	0005	Magenta	0500
Yellow	0006	Yellow	0600
White	0007	White	0700

## Displaying Key Names on the Status Line

By embedding key names in your application's status text, you can provide keystroke instructions to the user. To display a physical key name or Panther logical key name in the status text, include the string `%Kkkkk` anywhere in the status text. `kkkk` is the short name of a Panther logical key (such as `XMIT`, `EXIT`, `PF1`).

For a list of Panther logical keys, refer to Table 6-1 on page 6-7 in *Configuration Guide*.

If the specified key has a keytop defined in the key translation file used at runtime, the appropriate key label replaces the embedded code. For example, `%KEXIT` might be replaced with `ESC` when using the Panther-distributed key translation file for the IBM PC. If there is no keytop in the key translation file, the `%K` is stripped out, and the Panther logical name remains in the text.

## Sounding a Message Bell

To alert the user to a status line message, precede the status text with `%B`. This notation causes Panther to issue a beep on the terminal. The Status Line Text property setting might look like `%BDon't forget to check return/due date.`

---

# Using Panther Help Screens

---

A Panther help screen, or internal help, can be associated with any screen in your application as well as with any widget or menu item that can get focus. By using Panther internal help screens, you can ensure that your application help is environment- and platform-independent.

A Panther help screen can provide users with information about a field and its contents, a menu item, or the screen as a whole. A help screen that is attached to a widget can also be used to enter or change that widget's data.

Help is invoked at runtime when the user chooses Help or if you have specified that the help screen opens automatically on entry to a widget. Refer to “Determining the Level of Help Requests” on page 12-1 for the order of precedence that Panther uses to display help.

## Creating Internal Help Screens

Use the editor to create help screens as with any other screen in your application. There are several ways in which help screens can function and look. With Panther, you can create help screens that:

- Display information or instructions.
- Allow data entry and pass the results back to the calling screen.

In general, you want to make help screens distinctly different from your application screens. You might consider the following approaches and property settings:

- Set the screen BG Color to a contrasting color from the calling screen's background.
- Include a push button (such as OK) whose Control String property setting allows the user to exit the help screen, like `^jm_exit`.
- Build all help screens from a single repository entry and thereby ensure that all your help screens have a similar appearance. In addition, you then can easily

change help screen properties from a single source by propagating changes from a repository.

## Display-Only Help Screen

A help screen can be used for display-only data, such as information about the widget, menu item, or screen. A display-only help screen can contain any widget that conforms to these conditions:

- By design the widget is protected from input and clearing, for example, an array of dynamic labels.
- Its properties protect it from data entry (Input Protect property is set to Yes) and from clearing (Clearing Protect property is set to Yes).

You can populate a help screen's data in several ways:

- Enter data directly in the Label property (for dynamic labels), or in the Initial Text property of the Properties window.
- Fetch data at runtime from an LDB or from a database (refer to “Using Local Data Blocks” on page 25-7 in *Application Development Guide* for information about LDBs).
- Use a screen entry function that is specified in the help screen's Screen Entry property (under Focus).

To exit a Panther display-only help screen, the user can press any key or mouse click anywhere on the screen.

## Data Entry Help Screen

Users can enter data via a help screen that is invoked from a widget. You can do this by creating a help screen that contains one single line text widget, which is available for data entry.

When a widget opens a help screen, Panther copies the widget's value into the help screen's single line text widget. The user can then edit this widget's value. When the user chooses `XMIT` on the help screen with `XMIT`, Panther copies the content of the help screen widget back to the calling widget, then closes the help screen. If the user exits the help screen in any other way—for example, through `EXIT`—none of the data is copied and the calling widget remains unchanged.



**Note:** The control string `jm_keys XMIT` when attached to a push button causes an infinite loop. This occurs because the act of pressing `XMIT` actually activates a push button.

Panther's help function provides for the automatic passing of data whenever the help screen contains exactly one widget that is unprotected; that is, all protection properties (Input Protect and Focus Protect) are set to No. All other widgets on the help screen must be protected from input and focus.

**Note:** Scrolling widgets, like multiline text, can be scrolled even if they are protected.

The data entry widget on the help screen is usually defined to be the same length as the calling widget. If the help screen widget is too short, it is automatically made shifttable, and its length then matches the maximum length of the calling widget. If it is too long, then it is shortened—allowing only the number of characters that are defined for the calling widget.

Almost all properties that define the calling widget's input and format are adopted (not inherited) by the data entry widget on the help screen. For example, the following properties are adopted over any specifications for the help screen's widget:

- Keystroke Filter
- Check Digit
- Select on Entry
- Convert Case
- Range checks (Minimum and Maximum Values)
- Data Formatting specification
- Justification specification

Colors are not adopted; therefore, you can assign different colors to the calling widget and help screen widget.

## Multilevel Help System

A help screen can contain widgets that can access another level of help, somewhat like help-on-help. You can define up to five levels of help screens; the user can traverse the help system, moving from one help screen to another. The help screens maintain a separate stack from application screens. You attach help screens to other help screens by naming the subscreen in either a screen's or a widget's Help Screen property.

When the user requests additional information by choosing Help or when you set the Auto Help property to Yes, the lower level help screen opens.

You can provide multilevel help for:

- Data entry help screens by associating help with that screen's data entry widget or with the screen itself.
- Display-only help screens by associating help with that screen or with any widget on the screen that can receive focus (like a push button).

## Attaching Panther Help Screens

### How to Assign a Panther Help Screen to a Widget or a Screen

1. Select the object (screen or widget).
2. Under Help, enter the help screen name or choose More to select the screen from the Select Library Member dialog box. Optionally, you can specify the position of the screen in the Help Screen property. Choose OK. If you attach help to a widget, the Auto Help property is displayed.
3. In the widget's Auto Help subproperty, define the desired behavior:
  - Yes—The specified help screen displays automatically on entry to the widget.
  - No—(default) The help screen displays only on user request by choosing Help.

**Note:** If Auto Help is set to Yes, at runtime the help screen opens automatically unless the widget is validated. In this case, the user must choose Help to invoke the help screen.

To assign a Panther help screen to a menu item, enter the screen name in the item's Panther Help property (refer to Help Screen on page 25-10).

## Positioning Help Screens

You can control where and what appears when help is invoked by specifying the position and size of the help screen or selection screen directly in the Help Screen property or Selection Screen property, respectively. The full format of the position and size specification is:

*(row,col,height,width,vrow,vcol)*

If you do not specify a position, Panther attempts to display the entire help screen without hiding the field.

If you do not specify height and/or width, the help screen displays in the size specified in its Height and Width properties.

*row*

The row of the physical display at which to position the upper left corner of the help window's viewport. If *row* starts with a + or -, then *row* is the row offset, relative to the top left corner of the current screen, at which to position the upper left corner of the help window's viewport.

*col*

The column of the physical display at which to position the upper left corner of the help window's viewport. If *col* starts with a + or -, then *col* is the column offset, relative to the top left corner of the current screen, at which to position the upper left corner of the help window's viewport.

*height*

The size of the viewport in rows.

*width*

The size of the viewport in columns.

*vrow*

The row of the help window to be initially displayed in the upper left corner of the viewport.

*vcol*

The column of the help window to be initially displayed in the upper left corner of the viewport.

---

# Using Selection Screens

---

A selection screen, as well as a help screen, can be attached to any single line widget or multiline text widget (whose Word Wrap property is set to No). It is displayed under the following conditions:

- When the widget has focus and the user chooses the Item Selection (ITSEL) key.
- On entry to the widget if the Auto Item property is set to Yes.
- When the user chooses Help and a help screen (Panther or external) is not attached to the widget.

When the selection screen is displayed at runtime, the user can make a selection and it is automatically copied back to the associated widget on the calling screen. Cancelling the selection or pressing `EXIT` leaves the field unchanged.

Assigning a selection screen to a widget does *not* restrict data entry into the field, it only provides a list of possible choices. To restrict the choices to the entries in the selection screen, specify the same selection screen name in the Table Lookup property (under Input) as well. Refer to “Using Table Lookup Screens” [on page 12-13](#) for more information.

## Creating a Selection Screen

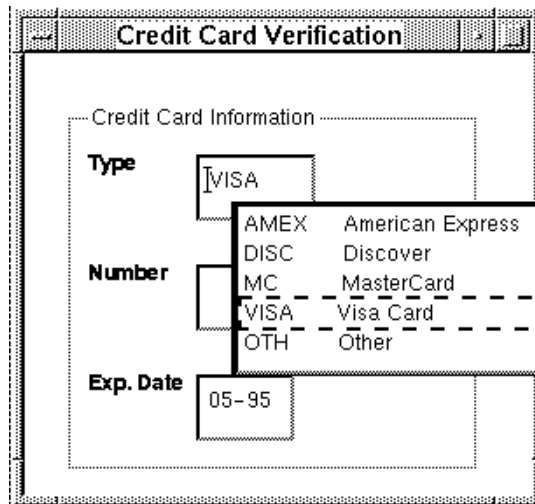
Use the screen editor to create selection screens, just as you would create any other screen in your application.

1. Use any widget that allows the user to make a selection; for example, a list box (set the Listbox Type property under Identity to Action) or an array of push buttons works well on selection screens.

The length of the widget on the selection screen need not be the same as the length of the associated text widget on the underlying screen. However, the length of text widget determines how much data does display. Therefore, if the selection widget length is greater than the calling text widget, the data might be

truncated if the Max Data Length property is not sufficient. However, by limiting the size of the widget, you can include additional information on the selection screen entry that will not be able to display in the calling text widget. Figure 12-1 illustrates an example of this type of selection screen.

If the underlying text widget has a format of right justified output (Justification property is set to Right) and has a Max Data Length of 4, for example, then only the rightmost part, or the first 4 characters, of the selection screen field is displayed in the underlying widget.



**Figure 12-1** Only the first four characters are returned to the field on the underlying screen. The remaining data serves as additional help.

2. Determine the source of the data that will populate the selection screen.
  - If the data are static, enter values for push buttons in the Label property, or in the Initial Text property for a list box.
  - If the data are dynamic, perhaps the result of a database query, consider using the library function `sm_svscreen` to make dynamically populated screens memory-resident.
  - If the data are passed from an LDB, remember to assign the same name to the selection screen widget and its corresponding LDB widget.

The selections can be a complete set of valid entries (validated by the screen specified in the Table Lookup property [on page 12-14](#)), or simply recommended or commonly used entries.

At runtime, you can shrink the screen to fit the number of selections by calling the library function `sm_shrink_to_fit`.

**Note:** In a three-tier architecture, you must create a selection service component for every selection screen. A selection service component is identical to its corresponding selection screen, except that it does not have push buttons or a title bar. It facilitates the transfer of data from the server to the selection screen on the client side of the application.

## Attaching a Selection Screen

### How to Assign a Selection Screen

1. Select the text widget.
2. Under Help in the Selection Screen property, enter the name of the screen that holds the valid choices or choose More to select the screen from the Select Library Member dialog box. Choose OK. The Auto Item subproperty is displayed.

You can control the selection screen's position by specifying the parameters in the property setting. For example, to display the screen `cc_code.itm` at row 5, column 25, enter `( 5 , 25 ) cc_code.itm` as the property setting. For more information on positioning screens, refer to [page 12-9](#), “Positioning Help Screens.”

3. Specify whether the selection screen should display in the Auto Item property:

- Yes—The specified selection screen will display on entry to the screen or widget.

**Note:** At runtime, Auto Item causes the selection screen to open automatically unless the widget has been validated. In this case, the user must choose the `ITSEL` logical key to invoke the selection screen.

- No—The selection screen will display only when the user chooses the `ITSEL` logical key.

4. (Optional) To provide a means of validating the input data to the text widget, enter the name of a lookup screen in the Table Lookup property under Input in the Properties window. Refer to the next section for details on using lookup screens.

---

## Using Table Lookup Screens

---

A table lookup screen is similar to a selection screen, except that it doesn't display at runtime. Instead, its contents are used to validate the entry in the field. You can assign a table lookup screen to any data entry widget: single line text, multiline text, and combo box.

For single line and multiline text widgets, it is common to specify the same screen in the Table Lookup property under Input as is indicated in the Selection Screen property under Help in the Properties window. This ensures that the list of entries on the selection screen, which *does* display at runtime, provides only valid choices.

At runtime, the table lookup screen binary is read, and screen entry and exit functions are called in the same way that other screens are processed.

## Creating and Attaching Table Lookup

Use the editor to create lookup screens just as you would create any other screen in your application. The screen can consist of any widget type that can display or accept data—data that is used to validate the contents of the widget on the underlying screen. This widget must not be focus protected; that is, the Focus Protection property (under Focus) must be set to No.

The data can be:

- Constant—created within the screen editor by entering values in the Label property or in the Initial Text property, depending on the type of widget.

- Dynamic—created at runtime, perhaps as the result of a database query. Consider using the library function `sm_svscreen` to make dynamically populated screens memory-resident.
- Returned via a screen entry function or through an LDB.

## How to Attach a Lookup Screen to a Data Entry Widget

1. Select the widget.
2. Under Input, enter the name of the lookup screen in the Table Lookup property or choose More to select the screen from the Select Library Member dialog box. Choose OK.
3. (Optional) If the widget is a single line text widget, you can also attach a displayable selection screen by naming it in the Selection Screen property under Help.

This selection screen provides the user with a list from which to choose. Refer to “Using Selection Screens” on [page 12-10](#) for more information.

---

# Using External Help Sources

---

Panther provides support for external help engines (WinHelp, etc.). You provide Panther with the following information:

- A help file that includes the compiled text from the defined help engine.
- Help context identifiers that you define, which are used to determine the help topic to be displayed at runtime.
- An installed help hook function that calls the help engine and passes it the context identifier. Refer to “Help Function” on [page 44-32](#) in *Application Development Guide* for more information.



## Attaching Help from Another Source

### How to Attach an External Help to an Application Object

1. Select the widget or screen.

At runtime, screen-level help is invoked only if the widget that has focus has no help of its own (including internal help).

2. Specify the context identifier in the External Help Tag property under Help.

The identifier indicates to Panther that there is an external help engine. Assuming that the external help processing hook function is installed, Panther passes the identifier to the hook function when the user chooses Help, and the hook function displays the appropriate help topic.

If the hook function fails, either because it is not installed or the external help file is not found, Panther attempts to access Panther help. Refer to “Determining the Level of Help Requests” [on page 12-1](#) for details on the order of precedence Panther uses to display help.

---

## Using Tooltips

---

You can implement tooltips for widgets in Panther Windows and Motif applications. A tooltip is a popup containing a text message that appears next to a widget in response to a mouse hover event. A mouse hover event occurs when the mouse pointer remains over a widget for approximately one-and-a-half seconds.

For each platform, there are widgets for which tooltips cannot be defined. For Windows, the exceptions are lines, boxes, grid frames and tab decks. For Motif, the exceptions are lines, boxes, tab decks, and grid frames (including widgets within a grid frame).

## Adding Tooltips in the Editor

The property `Tooltip Text` is found in the `Help` category in the `Properties` window in the editor. This is a string valued property that simply holds the text to be shown in the tooltip for a widget. If no tooltip text is specified for a widget, it will not display a tooltip at runtime. It is not possible to have a widget display an empty tooltip.

`Tooltip Text` is not a multi-valued property on the editor. All the occurrences of an array will show the same tooltip at runtime. However, starting in Panther 5.10, occurrence tooltips are supported at runtime.

The `Tooltip Text` property can be changed at runtime. Its value in C is `PR_TOOLTIP_TEXT` and its JPL mnemonic is `tooltip_text`.

## Controlling Tooltip Appearance

Tooltip appearance is, in general, determined by platform settings in the Windows desktop or in Motif. In the Windows desktop however, there is a property called `tooltip_style` (`PR_TOOLTIP_STYLE` in C) that can be used to get and set the Windows tooltip style bits.

The `tooltip_style` property is of application scope. It applies to the tooltips associated with all widgets and to the tooltips associated with the toolbar.

The following table lists the bits that can be set with the `tooltip_style` property, the Windows bits they correspond to, and the values so defined.

<b>Panther bit</b>	<b>Windows bit</b>	<b>Value</b>
<code>PV_TOOLTIP_ALWAYS</code>	<code>TTS_ALWAYSSTIP</code>	<code>0x01</code>
<code>PV_TOOLTIP_NOPREFIX</code>	<code>TTS_NOPREFIX</code>	<code>0x02</code>
<code>PV_TOOLTIP_NOANIMATE</code>	<code>TTS_NOANIMATE</code>	<code>0x10</code>
<code>PV_TOOLTIP_NOFADE</code>	<code>TTS_NOFADE</code>	<code>0x20</code>
<code>PV_TOOLTIP_BALLOON</code>	<code>TTS_BALLOON</code>	<code>0x40</code>

Some of these bits are relevant only to the latest versions of the Windows operating system. Other bits are undefined, but may be defined in future releases of Windows.

`PV_TOOLTIP_ALWAYS` is set by default. This means that the tooltip will be displayed even when the Panther application is not active. Clearing this bit will prevent tooltips from being displayed when the Panther application is inactive. However, tooltips will always display on an inactive form within an active Panther application.

`PV_TOOLTIP_NOPREFIX` is off by default and is irrelevant to Panther applications.

`PV_TOOLTIP_NOANIMATE` and `PV_TOOLTIP_NOFADE` are off by default. They are used to override Windows desktop settings. If a user's desktop is set up with the tooltip animation and fading features enabled tooltips will display those behaviors. Setting these bits will override the desktop preferences and turn off animation or fading, respectively, for tooltips in the Panther application.

`PV_TOOLTIP_BALLOON` is off by default. Setting this bit will cause tooltips to be displayed in a "balloon," like the voice balloon used in comic strips, rather than the usual plain rectangle.

## Examples

This JPL code will turn off the `TOOLTIP_ALWAYS` bit and prevent tooltips from being displayed for Panther widgets when the Panther application is not active:

```
@app()->tooltip_style = @app()->tooltip_style & ~ PV_TOOLTIP_ALWAYS
```

This will set both the `BALLOON` bit and the `NOANIMATE` bit:

```
@app()->tooltip_style = @app()->tooltip_style | \  
    PV_TOOLTIP_BALLOON | PV_TOOLTIP_NOANIMATE
```



# 13 Display Widgets

There are several means of displaying data on a screen. You can create:

- **Static labels**—For displaying information that will not be changed at runtime. Static labels are protected from receiving focus. You define the content of a static label at design time; it cannot be changed by the user or programmatically at runtime.
- **Dynamic labels**—For displaying information that might be changed at runtime. Dynamic labels cannot receive focus and the user cannot directly access or change the widget's content; however, the content can be changed programmatically.
- **Graphs**—For displaying data graphically. The data can be dynamic or static. Refer to “Using Graphs to Display Data,” [on page 13-4](#) for details on using graphs in your application.

---

## Labeling Information

---

Use static labels to display text on your screen that the user cannot interact with or modify. Static labels cannot be changed at runtime—either by the user or by application code. They are read-only widgets and are especially suited for:

- Labelling data entry fields.
- Displaying onscreen instructions.

Static labels have a limited set of properties associated with them and therefore, you will find that dynamic labels are a better choice for displaying text.

Static labels are automatically assigned to database columns that are imported into a repository. The label is given a name beginning with the letter L and is followed by the name of the text widget that is associated with it. For example, the database column `cust_id` is imported as a text widget of the same name. The import process creates a corresponding static label with the name `Lcust_id`.

## Defining the Look of a Static Label

You determine the content, size, and appearance of a static label at design time. By default, a static label takes the background color of the container that holds it; usually the background color of the screen is the background color of the label.

## How to Define the Content of a Static Label

1. Select the static label.
2. Under Identity, type the desired text in the Label property.

If the widget has no values set in the Height and Width properties under Geometry, the widget's size adjusts to fit its content (Size to Content property under Geometry is set to Yes).

**Note:** The widget's size will adjust to accommodate a larger label. However, if you increase the widget's size, either by dragging its resize handles or by specifying a size in the Length property, the widget's size remains unchanged.

## How to Resize a Static Label To Fit Its Content

1. Select the static label.
2. Under Geometry, clear the Height and Width properties of any values.

As long as the Size to Contents property is set to Yes, trailing blanks are eliminated.

If you reset both Height and Width properties, the Size to Contents property is ignored.

**Note:** If you change the Length property (either by setting the property value or by resizing the widget itself) or specify a picture to display on the label, the Size to Contents property is automatically reset to No.

## How to Define a Static Label's Color

1. Select the static label.
2. Under Color, set the FG (foreground) and/or BG (back ground) Color properties as desired.

---

# Creating Output Labels

---

Use dynamic labels for displaying data or output that can be generated and changed by the application at runtime. The user cannot directly change the contents of dynamic labels. Typically, dynamic labels are useful for:

- Displaying current date and time.
- Displaying derived data—such as the results of calculations, totals, and database queries.
- Protecting the label's content from being overwritten.
- Hiding and unhiding text programmatically as well as activating/deactivating (graying) at runtime.

## Defining Output Labels

You can control how the output data appears by setting the Format/Display properties. These include, for example, definitions for date/time format, how numeric output should appear, and what characters should appear if the output is null. For details on how to define display formats, refer to [page 10-4](#), “To Display or Not Display.”

When the content of a dynamic label changes, you can also force the widget's size to adjust to fit its new content.

## How to Change the Size Dynamically When Content Changes

- Under Geometry, set the Size to Contents property to Yes.

In the screen editor and at runtime, trailing blanks are eliminated and any previously set Height and Width property values are removed.

If you decrease the number of characters in the Label property, the label's length adjusts appropriately to fit the content.

If you reset both Height and Width properties, the Size to Contents property is ignored.

**Note:** If you change the Length property (either by setting the property value or by resizing the widget itself in the screen editor) or specify a picture to display on the label, the Size to Contents property is automatically reset to No.

In addition, you can display multiple lines of data in an output label by setting the Array Size property.

---

# Using Graphs to Display Data

---

You can present data on your application screens in graph or chart format by using graph widgets. Graph data can be generated at runtime or obtained from static sources and can be displayed in a variety of formats:

- Pie chart—Displays data from a single source as proportional segments of a circle.
- Bar/Line graph—Bar, line, curve, trend, point, or area plots.
- XY-plot—Line, curve, and point plots.



- High/Low chart—High, low, close, and open values plotted together on a single marker that resembles an I-beam. (Used, for example, to track the performance of a single stock over a period of time.)

The information on graph widgets is divided into five main sections. The first describes how to create a graph widget of any type and provides detailed information on the topics that apply to more than one graph type.

The four remaining sections are each dedicated to one particular type of graph: pie chart, bar/line graph, XY-plot, and high/low chart. Each of these sections outlines the procedure for creating a graph of the specified type and presents detailed information on the topics unique to that graph type.

## How to Create a Graph Widget

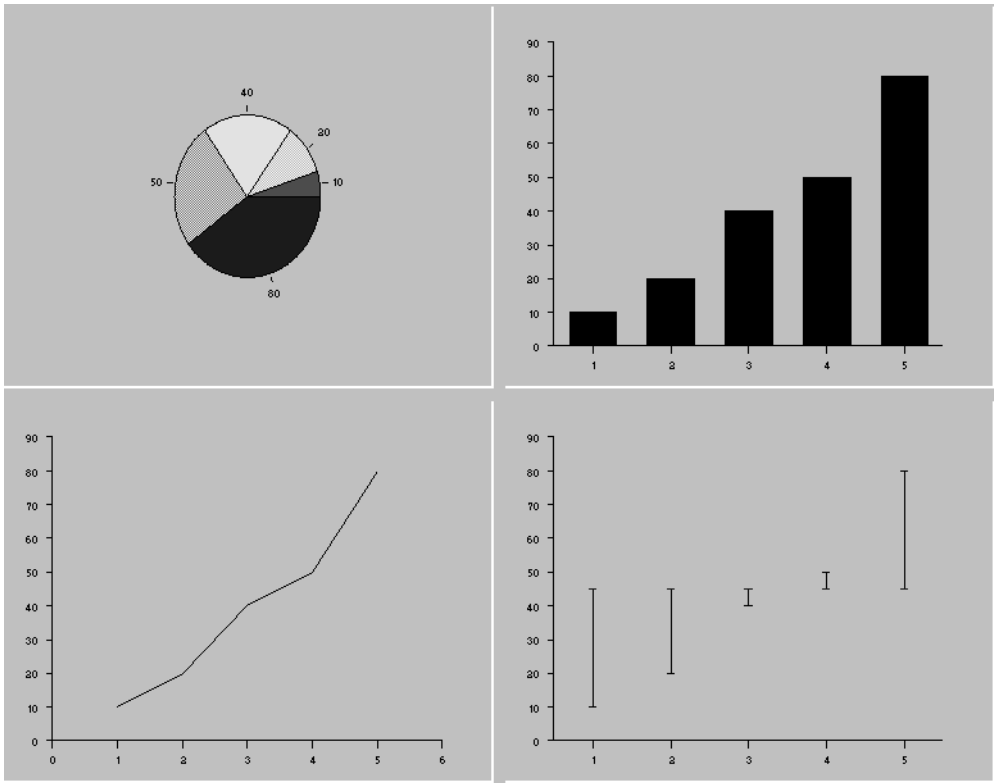
1. Choose Create→Graph or choose the graph icon from the Create toolbar.
2. On your application screen, click once to create a default-sized widget or drag out the graph to the desired dimension (which you can change later).

On GUI platforms, a pie chart containing sample data appears in the widget. Once you supply data to the widget, it will replace the sample data initially displayed.

In character mode Panther, an empty rectangle is displayed. (Graphs are not visible in character mode Panther, although you can set all graph widget properties; the graph will be displayed when the screen is opened in a GUI environment.)

3. (Optional) Under Identity, assign a name to the graph in the Name property. Panther internally assigns the graph widget a field number. Graphs are numbered sequentially from left to right and top to bottom—as `Graph #1`, etc.
4. (Optional) Under Identity, assign a label to the graph in the Label property. This label is displayed in character mode to indicate where the graph is located—useful if you are developing a screen in character mode Panther that will eventually be used on a graphical platform.
5. Under Graph, set the Chart Type property to one of the following: Pie (default), Bar, XY-Plot, or High/Low. The chart displayed in the widget changes to match the type you have specified.

**Note:** If data are not visible on the sample chart, adjust the graph widget's foreground and background colors. Refer to “Changing Color Properties” on page 11-3 for information.



**Figure 13-1** Samples of chart types. Newly created graphs are displayed with sample data

The properties unique to graph widgets are listed under the Graph heading in the Properties window. Some graph properties apply to all types, but many are specific only to particular graph types.

If you later (either in the editor or at runtime) change the type of a graph widget, some of the properties initially specified will not be applicable to the new graph. These properties no longer appear in the Properties window, but they are retained in the widget. If you later change the widget back to the

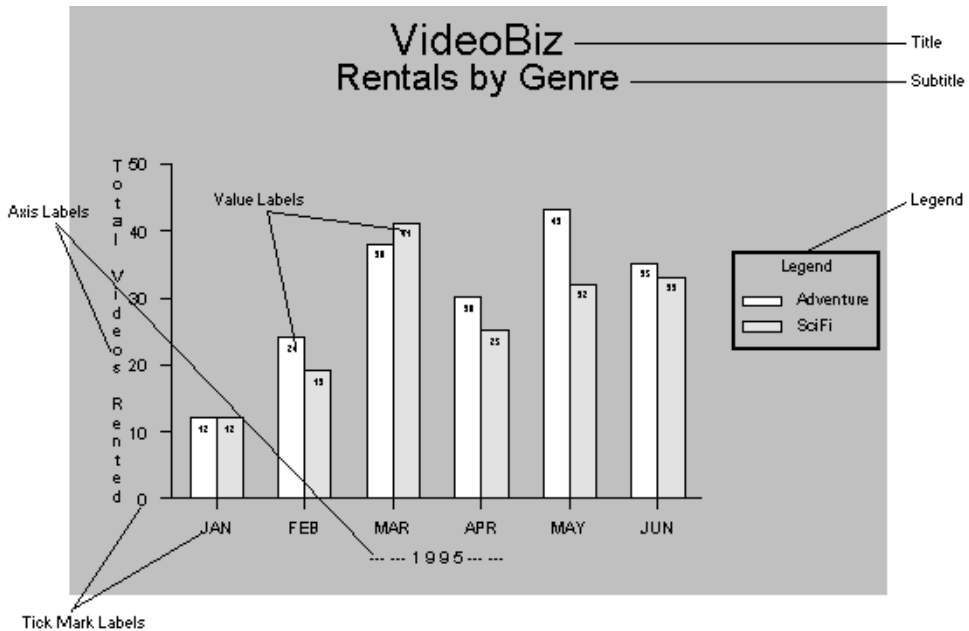
previous type, these properties again appear in the Properties window, their settings the same as before the type was changed.

**Note:** If you enter Test mode before populating the graph with data, the widget is displayed as a graph or chart with no data. The graph or chart that initially appears when you create the widget is populated with sample data and serves only as a visual cue. In Test mode, the pie chart appears to consist of just a single segment representing the entire pie chart; other chart types simply display X and Y axes but no data.

## Controlling Graph Text

Text that can appear on a graph falls into the following categories:

- **Title**—Title and/or subtitle. For information, on adding titles and subtitles, refer to [page 13-9](#), “Adding a Title to a Graph.”
- **Legend**—A key to the data shown on the graph. For information on adding a legend, refer to [page 13-10](#), “Adding a Legend to a Graph.”
- **Label**—Axis and tick mark labels; pie chart segment values, percentages, and labels; bar/line graph values. For information on adding labels, refer to [page 13-15](#), “Describing the X and Y Axes of a Graph,” or [page 13-35](#), “How to Identify the Segments of a Pie Chart.”



**Figure 13-2** A variety of text components is available for all chart types.

## How to Specify a Font for the Graph

The font and style for all text in the graph widget is determined by the property settings under the Font heading. Note that the fonts available on some platforms are specific to graph widgets and might differ from those you can choose for the screen and other widget types.

1. Select the graph widget.
2. Under Font, specify the font name and style properties for the graph.

These settings apply to all text in the graph widget. If the Font Name property is set to Default, text on the graph will be in the Simplex font.

To ensure portability of fonts across different platforms, use graph-specific font specifications (prefixed with the word Graph in the font name). On Windows, graph-specific fonts are listed after the system fonts.

For further information on applying font properties to widgets, refer to [page 10-7](#), “Specifying Fonts.”

## Specifying Sizes for Graph Text

Text size within a graph widget is always specified relative to the graph size. This way, if you change the size of the graph widget, the text size increases or decreases proportionately.

Text size is specified as a value between 0.0 and 100.0, inclusive, representing a percentage of the widget size. It is set independently for the title, subtitle, legend, and labels. Table 13-1 indicates the location of the applicable Text Size property and its default value for each of these text elements.

**Table 13-1 Setting Text Size properties for text elements in a graph widget**

Text element	Location of Text Size property	Default
Title	under Graph: a subproperty of Title	5 . 0
Subtitle	under Graph: a subproperty of Subtitle	4 . 0
Legend	under Legend (a subheading of Graph): a subproperty of Placement	2 . 0
Labels (axis, tick mark, and pie segment labels)	under Graph	2 . 0

## Adding a Title to a Graph

You can add a title and/or subtitle to any graph widget. The title, if present, is centered near the top of the widget. The subtitle, if present, is centered beneath the title. The font and style of the title and subtitle text are determined by the settings of the properties under the Font heading.

### How to Place a Title on the Graph

1. Select the graph widget. Under Graph, enter a title in the Title property. A new property, Text Size appears as a subproperty of Title.

2. Set the Text Size property to a value between 0.0 and 100.0, inclusive. The default text size for the title is 5.0. For more information, refer to [page 13-9](#), “Specifying Sizes for Graph Text.”

## How to Place a Subtitle on the Graph

1. Select the graph widget. Under Graph, enter a subtitle in the Subtitle property. A new property, Text Size appears as a subproperty of Subtitle.
2. Set the Text Size property to a value between 0.0 and 100.0, inclusive. The default text size for the subtitle is 4.0. For more information, refer to [page 13-9](#), “Specifying Sizes for Graph Text.”

## Adding a Legend to a Graph

A legend is an optional element of a graph widget. If present, it serves as a key to the data displayed on the graph:

- For a bar/line graph, XY-plot, or high/low chart, the legend identifies each data set plotted. The identifying label for each data set is specified in the Legend property for the corresponding data series.
- For a pie chart, the legend can be used to identify chart segments—an alternative to placing labels beside the corresponding segments. The identifying text for the segments is specified under Pie Segments, in the Label Source property. Label Location, a subproperty of Label Source, specifies whether the text is to appear beside the segments or in a legend. For more information on displaying segment labels, refer to [page 13-35](#), “How to Identify the Segments of a Pie Chart.”

**Note:** Panther does not display the legend if it is empty. Therefore, you must enter legend text for at least one data series, or specify pie segment labels with Location set to Legend, before the legend will appear in the widget.

## How to Add a Legend to the Graph Widget

1. .Select the graph widget. Under Graph, expand the Legend heading and set the Placement property to one of the following:

- Default—The legend is centered vertically on the right side of the graph widget, outside the data space.
- None—The legend is not displayed.
- Location—Allows you to specify one of nine preset locations for the legend, eight around the perimeter of the graph, and one in the middle. Also allows you to specify whether the perimeter locations are within or outside of the data space. The default, if Location is specified, is centered vertically on the right side of the graph widget, outside of the data space. Refer to [page 13-12](#), “How to Specify the Legend Placement by Location.”
- Position—Allows finer control over legend positioning than the Location setting. If no coordinates are specified, the legend does not appear. Refer to [page 13-13](#), “How to Specify the Legend Placement by Position.”

If you set the Placement property to Location, Position, or Default, additional properties are displayed.

2. (Optional) Enter a title in the Title property.
3. Set the Text Size property to a value between 0.0 and 100.0, inclusive. The default text size for the legend is 2.0. For more information on specifying text size, refer to [page 13-9](#), “Specifying Sizes for Graph Text.”
4. Set the Border Width property to a value between 0 and 100, inclusive. The default is 0, indicating no border. A border of width 1 is a thin line; higher values represent a wider border. The absolute width of the border scales up and down with the size of the widget.
5. Enter the text to identify each data series or pie segment in the legend:
  - For bar/line graphs, XY-plots, and high/low plots—Under the Data Series headings, enter the applicable legend text in the Legend property. If no legend text is specified for a given data series, that series will not be identified in the legend. For more information, refer to [page 13-23](#), “Defining the Data Series.”
  - For pie charts—Under Pie Segments, specify the segment labels in the Label Source property, and set the Label Location property to Legend. For more information on pie segment labels, refer to [page 13-35](#), “Describing Segment Characteristics.”

The legend is displayed only if there is at least one data series with legend text specified or one pie segment label.

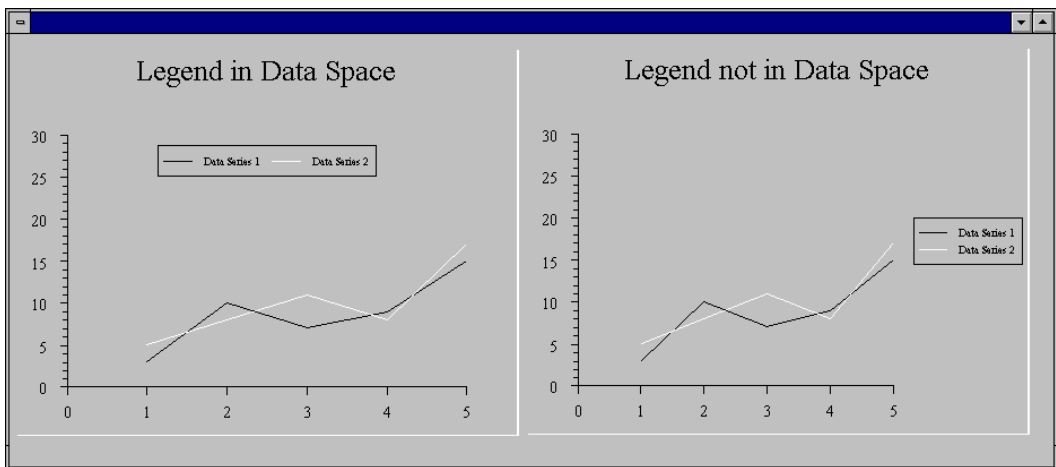
## How to Specify the Legend Placement by Location

If you set the legend's Placement property to Location, you can place the legend in one of nine preset locations in the graph widget, either within or outside of the data area.

1. Select the graph widget. Under Graph, expand the Legend heading and set the Placement property to Location. Additional properties appear as subproperties of Placement.

By default, when Location is specified, the legend is centered vertically on the right side of the graph widget, outside of the data space.

2. Set the In Data Space property to indicate whether or not the legend should appear within the data space:
  - Yes—The entire area of the graph widget, except for space allocated to the title and subtitle, is used for the chart or plot. The legend is placed within the data space.
  - No—(default) The plot is shifted and resized appropriately to create a margin for placement of the legend.



**Figure 13-3** Legends can be displayed in the data space (left graph) or outside the data space (right).

If the legend is centered in the widget (X Location set to Center, Y Location set to Middle), the legend appears within the data space, regardless of the setting of the In Data Space property.



3. Set the X Location and Y Location properties:
  - X Location—Left, Center, or Right (default).
  - Y Location—Top, Middle (default), or Bottom.

If the X location is set to Center, the entries appear in a row across the legend; otherwise, the entries appear in a column.

## How to Specify the Legend Placement by Position

If you set the legend's Placement property to Position, you determine its location precisely by specifying X and Y coordinates and by indicating which part of the legend is anchored to those coordinates.

1. Select the graph widget. Under Graph, expand the Legend heading and set the Placement property to Position. Additional properties appear as subproperties of Placement.
2. Set the X Position and Y Position properties to the desired coordinates for the legend. Values for these properties must be integers between 0.0 and 100.0, inclusive.
  - X Position—Horizontal position within the widget. 0 is the left edge; 100, the right edge; 50, the center; and so forth.
  - Y Position—Vertical position within the widget. 0 is the bottom; 100, the top; 50, the middle; and so forth.
3. Set the X Anchor and Y Anchor properties to indicate what point in the legend should be anchored to the X Position and Y Position coordinates you have set.
  - X Anchor—Left, Center (default), or Right.
  - Y Anchor—Top, Middle (default), or Bottom.

## Specifying Text Size for Graph Labels

### How to Specify Text Size for Graph Labels

1. Select the graph widget.

2. Under Graph, set the Text Size property to the desired size for all labels on the graph.

**Note:** The value must be a number between 0.0 and 100.0, inclusive, representing a percentage of the widget size. The default is 2.0.

This text size setting applies to:

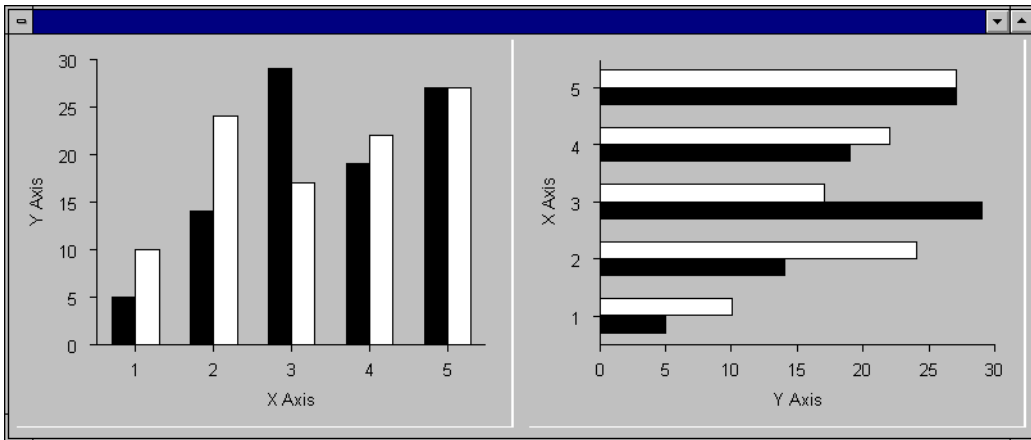
- Axis and tick mark labels on bar/line graphs, XY-plots, and high/low charts.
- Values displayed on bar/line graphs.
- Values and percentages displayed on pie charts.
- Pie chart segment labels (if they are displayed with the segments, rather than in a legend).

The font and style of the text are determined by the settings of the properties under the Font heading.

## **Orienting the Graph Vertically or Horizontally**

Bar/line graphs, XY-plots, and high/low charts can be oriented either vertically or horizontally. By default, they are oriented vertically.

When the graph is oriented vertically, the X axis runs horizontally and the Y axes run vertically. When the graph is oriented horizontally, the X axis runs vertically and the Y axes run horizontally.



**Figure 13-4** Graphs can be oriented vertically (default) or horizontally (right)

## How to Specify the Graph Orientation

1. Select the graph widget.
2. Under Graph, set the Orientation property to the desired orientation:
  - Vertical
  - Horizontal

The default is Vertical.

## Describing the X and Y Axes of a Graph

Bar/line graphs, XY-plots, and high/low charts have one X axis and one or two Y axes (Y1 and Y2). The X axis is described by the properties under the X Axis and X Tick Marks subheadings of the Graph heading; the Y1 and Y2 axes are described by the properties under the Y1 and Y2 Axis and Tick Marks subheadings, respectively.

The Y1 Axis and Tick Mark headings are present whether or not data is plotted against this axis. Headings for the Y2 axis are displayed if at least one data series is plotted against it. (Refer to “How to Specify the Y Axis for a Data Series” on [page 13-29](#) for information.)

The text size for axis labels and tick marks is determined by the setting of the Text Size property under the Graph heading. For information on setting text size, refer to [page 13-9](#), “Specifying Sizes for Graph Text.”

## How to Specify the Axis Locations

In a Bar/line Graph, XY-Plot, or High/Low Chart, Panther allows you to specify where the axes are positioned on the graph.

1. Select the graph widget.
2. Under Graph, expand the applicable Axis heading and set the Location property to one of the following values:
  - Edge (default for Y1 and Y2 axes)
  - Opposite Edge, Zero (default for X axis)
  - None

The axis locations defined by each of these settings depends on the orientation selected for the graph. Refer to “Orienting the Graph Vertically or Horizontally” [on page 13-14](#) for an explanation of graph orientation.

Table 13-2 shows where each Location setting places the axes when the graph orientation is set to Vertical. Table 13-3 shows where each Location setting places the axes when the graph orientation is set to Horizontal.

**Table 13-2 X and Y axes locations in vertical orientation**

	Location property setting			
	Edge	Opposite Edge	Zero	None
X Axis	along bottom of data space	along top of data space	aligned to zero-point of Y1 axis (default)	not displayed
Y1 Axis	along left side of data space (default)	along right side of data space	aligned to zero-point of X axis	not displayed
Y2 Axis	along right side of data space (default)	along left side of data space	aligned to zero-point of X axis	not displayed

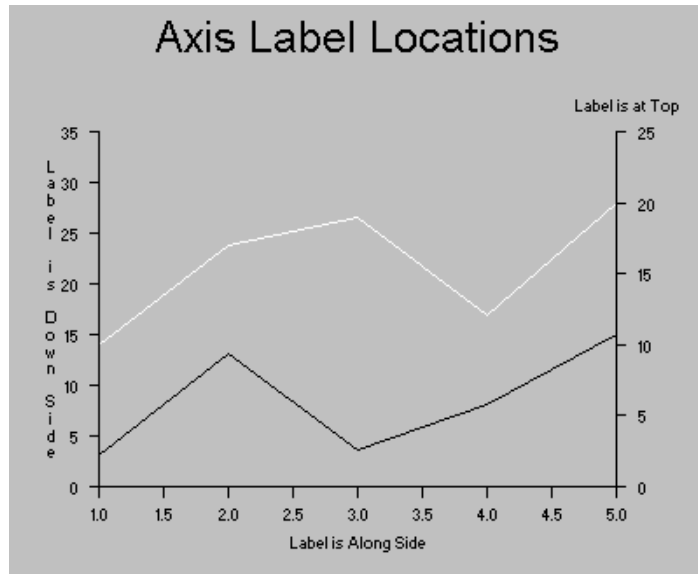
**Table 13-3 X and Y axes locations in horizontal orientation**

Location property setting				
	Edge	Opposite Edge	Zero	None
X Axis	along left side of data space	along right of data space	aligned to zero-point of Y1 axis (default)	not displayed
Y1 Axis	along bottom of data space (default)	along top of data space	aligned to zero-point of X axis	not displayed
Y2 Axis	along top of data space (default)	along bottom of data space	aligned to zero-point of X axis	not displayed

## How to Place Labels on the X/Y Axes

In a Bar/line Graph, XY-Plot, or High/Low Chart, Panther allows you to specify text labels for the axes in the graph widget.

1. Select the graph widget. Under Graph, expand the applicable Axis heading and enter identifying text for the corresponding axis into the Label property.
2. Set the Label Location to one of the following:
  - Along Side—(default) The label runs parallel to the axis. If the axis is vertical, the label is rotated 90 degrees.
  - Down Side—The label reads from top to bottom, with each character right side up relative to the screen. (Applicable only if the axis runs vertically.)
  - Top—The label text is oriented horizontally and is placed near the top of the axis. (Applicable only if the axis runs vertically.)



**Figure 13-5** Axis label and its placement can be specified via the Label and Label Location properties under the applicable axis heading.

## Placing Tick Marks on the Axes

By default, Panther determines appropriate minimum and maximum values and tick mark increments based on the data to be plotted. You can specify the following tick mark characteristics for each axis:

- maximum and minimum values
- scale (linear or common log)
- major and minor increments
- placement inside or outside the axis line
- width
- label (for each tick mark, individually)
- grid lines

## How to Specify Minimum and Maximum Values for the X/Y Axes

In a Bar/line Graph, XY-Plot, or High/Low Chart:

1. Select the graph widget.
2. Under Graph, expand the applicable Axis heading and enter the desired minimum and maximum values for the axis in the Minimum and Maximum properties, respectively.

By default, Panther determines appropriate minimum and maximum values for each axis based on the data to be plotted.

## How to Specify the Type of Scale on the X/Y Axes

In a Bar/line Graph, XY-Plot, or High/Low Chart:

1. Select the graph widget.
2. Under Graph, expand the applicable Axis heading and set the Scale property to one of the following:
  - Linear (default)
  - Common Log

## How to Specify Tick Mark Increments on the X/Y Axes

By default in a Bar/line Graph, XY-Plot, or High/Low Chart, Panther determines appropriate increments for the major tick marks on each axis, based on the data to be plotted. You can specify different values if desired, as well as place minor tick marks on the axes, at increments of your choosing.

1. Select the graph widget.
2. Under Graph, expand the applicable Tick Marks heading and enter the desired increment between major (larger) tick marks in the Major Increment property.

**Note:** The increment must be a positive value. The default is a value determined by Panther based on the data plotted against this axis.

3. Enter the desired increment between minor (smaller) tick marks in the Minor Increment property.

**Note:** The increment must be a positive value. The default is no minor tick marks.

## How to Specify Tick Mark Style on the X/Y Axes

In a Bar/line Graph, XY-Plot, or High/Low Chart:

1. Select the graph widget
2. Under Graph, expand the applicable Tick Marks heading and set the Style property to one of the following:
  - In—Tick marks run inward from the edge of the data space (towards the plot).
  - Out—(default) Tick marks run outward from the edge of the data space (away from the plot area).
  - Both—Tick marks run both inward and outward. (If the axis is on an edge, the tick marks cross it.)
  - None—Tick marks are not displayed.

## How to Specify Tick Mark Width on the X/Y Axes

In a Bar/line Graph, XY-Plot, or High/Low Chart:

1. Select the graph widget.
2. Under Graph, expand the applicable Tick Marks heading and set the Width property to a value between 1 and 100, inclusive.

**Note:** The default tick mark width is 1. Higher values represent broader tick marks. The absolute width scales up and down with the size of the widget.

## How to Label the Tick Marks on the X/Y Axes

In a Bar/line Graph, XY-Plot, or High/Low Chart:

1. Select the graph widget.
2. Under Graph, expand the applicable Tick Marks heading and set the Label Source property to either:



**Note:** The Labels set in the Label Source property are only placed on major (larger) tick marks. Minor (smaller) tick marks are not labelled.

- The name of an array containing the tick mark labels.

To specify an array on the current screen, simply enter the widget name.  
For example:

```
Y1_tick_labels
```

To specify an array on another screen, enter the array name in the `screen_name!field_name` format, such as:

```
graph_format_screen!Y2_labels
```

Any type of array can be used. However, single and multiline text widgets and list boxes are best suited to this purpose. The array can be hidden.

- A list of tick mark labels.

Begin the list with an equal sign (=) and separate all values with a space or other appropriate delimiter. If a non-alphanumeric character follows the equal sign, it is interpreted as the delimiter for the list; if an alphanumeric character follows the equal sign, then space is assumed to be the delimiter.  
For example:

```
=Jan Feb Mar Apr May Jun
```

```
= Jan Feb Mar Apr May Jun
```

```
=/Mar 31/ Jun 30/Sept 30/Dec 31
```

**Note:** In the third example, a slash (/) is used as the delimiter since the values contain spaces.

## How to Show Tick Marks as Grid Lines

**Note:** If desired, you can expand the tick marks on any or all axes into grid lines.

In a Bar/line Graph, XY-Plot, or High/Low Chart:

1. Select the graph widget.
2. Under Graph, expand the applicable Tick Marks heading and set the Grid Style property to one of the following:
  - Solid
  - Dashed

- Dotted
- None (default)

If this property is set to any value other than None, the tick marks are drawn as lines across the entire data space. If grid styles are specified for both the horizontal and vertical axes, a “graph paper” effect is created.

## Creating a 3-Dimensional Effect

### How to Display a Graph in 3D

1. Select the graph widget. Under Graph, set the 3D property to Yes. Several new properties appear as subproperties of 3D. Refer to Table 13-4 for a listing of these properties.
2. Set the 3D subproperties to specify the desired perspective for displaying the graph.

**Table 13-4 3D properties for graph widgets**

Property*	Applicable Chart Types	Description
Horiz Rotation	Bar XY-Plot High/Low	Horizontal rotation (about the vertical axis). Value must be a number of degrees between 0 and 90, inclusive. Default is approximately 20 degrees.
Vert Rotation	all	Vertical rotation (about the horizontal axis). Value must be a number of degrees between 0 and 90, inclusive. Default is 20 degrees for bar/line graphs, XY-plots, and high/low plots and 30 degrees for pie charts.

At 0 degrees, the viewer is looking straight into the graph, as if it were simply shown in two dimensions. At 90 degrees, the viewer is looking directly at the top (for bar/line graphs, XY-plots, and high/low plots) or lower edge (for pie charts) of the graph.

**Table 13-4 3D properties for graph widgets (Continued)**

Property*	Applicable Chart Types	Description
Depth	all	Depth of each object in the graph. Value must be a number between 0.0 and 100.0, inclusive representing a percentage of the widget size. Default depth is 10.0 percent.

\*All properties in this table are subproperties of 3D, under the Graph heading.

For more information on how the 3D property settings affect the appearance of a particular chart type, refer to the following:

- Pie chart: “How to Display a Pie Chart in 3D” ([page 13-33](#))
- Bar/Line graph: “How to Display a Graph in 3D” ([page 13-22](#))
- XY-plot: “How to Display an XY-Plot in 3D” ([page 13-46](#))
- High/Low/Close plot: “How to Display a High/Low Chart in 3D” ([page 13-50](#))

## Defining the Data Series

A data series represents not only the source for one set of data values, but also the plot style, Y axis specification, and labelling information for the data in the series.

For all chart types, there is at least one “Data Series” subheading of the Graph properties heading; under each data series, the data or its source is specified in one or more “Value Source” properties. Table 13-5 shows the names of these headings and properties for each chart type, along with the number of data sources permitted for each.

**Table 13-5 Data source properties for graph widgets**

Chart type	Data series heading	Value source property	Number of data sets permitted
Pie	Data Series	Value Source	One data series with a single value source.

**Table 13-5 Data source properties for graph widgets (Continued)**

<b>Chart type</b>	<b>Data series heading</b>	<b>Value source property</b>	<b>Number of data sets permitted</b>
Bar	Data Series #n	Value Source	Up to 12 data series, each with a single value source.
XY-Plot	Data Series #n	X Value Source Y Value Source	Up to 12 data series, each with two value sources: X and Y.
High/ Low	High Data Series Low Data Series Close Data Series Open Data Series Data Series #n	Value Source	Up to 12 data series, each with a single value source. The first four are plotted as high, low, close, and open data, respectively; any additional data series are treated as if plotted on a bar/line graph.

When you first expand the Data Series heading, only the Value Source property appears. Once you specify the value source, additional properties are displayed. The specific properties you can set for a data series depend on the chart type.

In the Properties window, one additional data series heading is always provided beyond the last one that contains a value source (up to the maximum number of data series permitted for the specified chart type). Thus, there is always a place to add the next data set. For example, if three data series have been entered for a bar/line graph, the heading Data Series #4 is also present in the Properties window.

## How to Specify Data Values

The method for specifying the data value source is the same for all chart types. The names of the properties differ, however, and the number of sources permitted depends on the type.

Data supplied in the value sources must be numeric. If the values have a numeric format that includes non-numeric characters such as a currency symbol, these are stripped out. In pie charts and bar/line graphs, where the value can be displayed with the plot, only the numeric portion appears.

1. Select the graph widget.

2. Under Graph, expand the applicable Data Series heading and set the Value Source property to either:

- The name of an array containing the data.

The array name need not exist or contain data at this time. Data is used in the graph as it becomes available, such as when the array is populated at runtime. To specify an array on the current screen, simply enter the widget name. For example:

```
cust_totals
```

To specify an array on another screen, enter the array name in the `screen_name!field_name` format, such as:

```
admin_screen!num_rentals
```

**Note:** If the value source is an array on another screen, the values do not appear on the graph while you are in the screen editor; sample data is displayed instead. In test or application mode, however, Panther plots the values from the source you have specified.

Any type of array can be used as a value source. However, single and multiline text widgets and list boxes are best suited to this purpose.

Since every keystroke into the source array at runtime causes a re-draw, use hidden arrays as value sources for the data series. Capture the data from user input or from the database into an onscreen array; then copy the contents of that array into the hidden array. The chart is thus re-drawn only once—when the values are copied into the hidden array that you have specified as the value source.

- A list of the data values.

To enter a list of values into Value Source, begin the list with an equal sign (=) and separate all values with a space or other appropriate non-alphanumeric delimiter. If the character following the equal sign is alphanumeric, space is assumed to be the delimiter; if the character following the equal sign is not alphanumeric, then that character is the delimiter for the list. For example:

```
= 1.5 3 4.5
```

```
=1.5 3 4.5
```

```
=/1.5/3/4.5
```

**Note:** Once you have specified the value source for a data series, additional properties are displayed.

## How to Specify the Plot Style for a Data Series

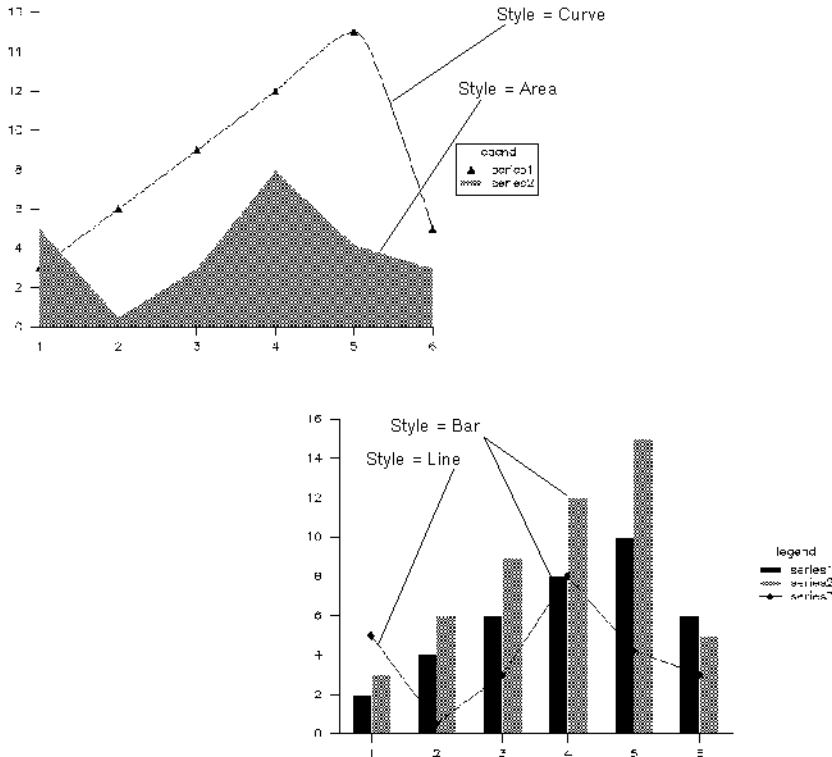
For each data series in a bar/line graph or XY-plot and for data series 5 and up in a high/low chart, you can specify the format for displaying the data. You can use different formats within the same graph.

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series heading and set the Style property.

If the Style property is not displayed, be sure to set the Value Source property first.

**Note:** If the chart type is Bar (all data series) or High/Low (data series 5 and above), the style must be one of:

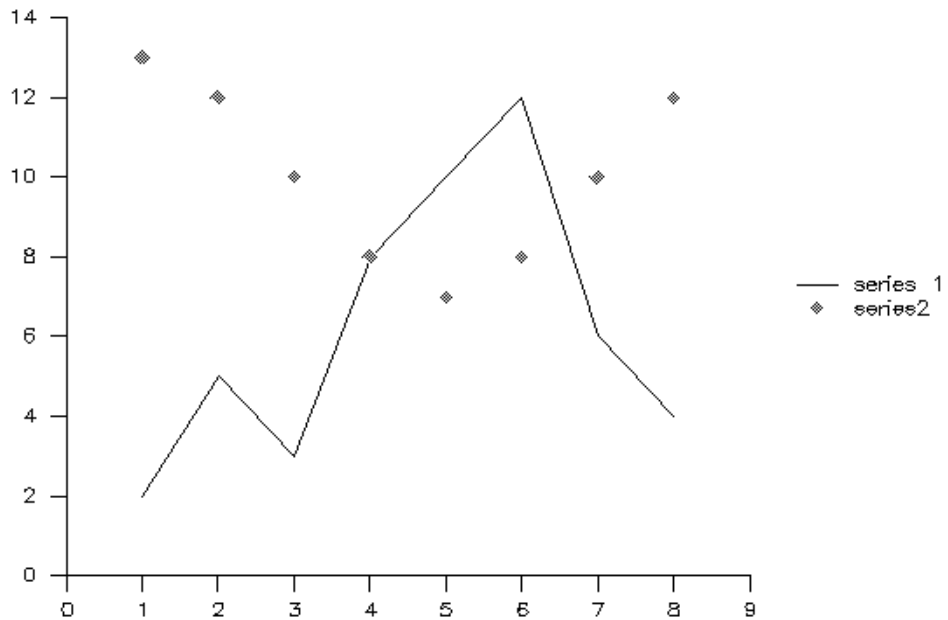
- Bar (default)—Values appear as bars of appropriate length.
- Line—Each data point is connected to the next by a straight line. Line Style and Line Width properties appear as subproperties when Style is set to Line.
- Curve—A curve is fit to the data points. Line Style and Line Width properties appear as subproperties when Style is set to Curve. (Not applicable in 3D.)
- Point—Data points are indicated but have no connecting lines.
- Trend—A trend line is fit to the data points. Line Style and Line Width properties appear as subproperties when Style is set to Trend.
- Area—The entire area between the data points and the X axis is filled in.



**Figure 13-6** Plot styles can be combined on a bar/line graph.

If the chart type is XY-Plot, the style must be one of:

- Line (default)—Each data point is connected to the next by a straight line. Data points are connected by a straight line, point to point. Line Style and Line Width properties appear as subproperties when Style is set to Line.
- Curve—A curve is fit to the data points. Line Style and Line Width properties appear as subproperties when Style is set to Curve. (Not applicable in 3D.)
- Point—Data points are indicated but have no connecting lines.



**Figure 13-7** Plot styles can be combined on an XY-plot.

## How to Specify the Line Style for a Data Series

1. Select the graph widget. Under Graph, expand the applicable Data Series subheading and set the Style property to Line, Curve, or Trend. Line Style is displayed as a subproperty of Style.
2. Set Line Style to one of: Solid (default), Dashed, Dotted, Long Dash, Dash Dot, or None.

## How to Specify the Line Width for a Data Series

1. Select the graph widget. Under Graph, expand the applicable Data Series subheading and set the Style property to Line, Curve, or Trend. Line Width is displayed as a subproperty of Style.



2. Set Line Width to a value between 1 and 100, inclusive. The default line width is 1. The higher the value, the broader the line.

## How to Specify the Point Marker Format for a Data Series

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series subheading, and set the Point Marker property to the desired format.

**Note:** If the Point Marker property is not displayed, be sure to set the Value Source property first.

The point marker format for the data series can be any one of: None (default), Dot, +, \*, o, x, Square, Diamond, Triangle, Circle, Filled Square, Filled Diamond, Filled Triangle, or Filled Circle.

Point markers are displayed if the Style property for the data series is Line, Curve, Point, or Trend.

## How to Specify the Y Axis for a Data Series

If the chart type is Bar, XY-Plot, or High/Low, you can specify the Y axis (Y1 or Y2) against which the data in this series is to be plotted.

By default, the Y1 axis appears on the left and the Y2 axis on the right when the graph is oriented vertically. When the graph is oriented horizontally, the Y1 axis runs along the bottom and the Y2 axis along the top. For more information on positioning the Y axes, refer to [page 13-16](#), “How to Specify the Axis Locations.”

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series subheading and set the Y Axis property to the desired axis (Y1 or Y2) for this data set.

**Note:** If the Y Axis property is not displayed, be sure to set the Value Source property first.

## How to Specify the Color for a Data Series

If the chart type is Bar, XY-Plot, or High/Low, you can specify a Panther basic color for each data set's plot. For information on Panther basic colors, refer to [page 11-2](#), “Using Panther Basic Colors.”

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series subheading and set the Color property to the desired Panther basic color.

The data in this series will be plotted in the specified color.

**Note:** If the Color property is not displayed, be sure to set the Value Source property first.

By default, each data series is plotted in a different color: data series #1 in black, #2 in blue, #3 in green, #4 in cyan, and so forth, through the list of Panther basic colors.

## How to Enter Legend Text for a Data Series

If the chart type is Bar, XY-Plot, or High/Low, you can supply a text string for each data series to identify it in the legend for the graph.

If you do not enter legend text for the series, it will not be listed in the graph's legend. If the graph does not have a legend, any legend text entered here is ignored; it is retained, however, and is used if a legend is later added to the graph.

For information on creating and displaying a legend in a graph widget, refer to [page 13-10](#), “Adding a Legend to a Graph.”

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series subheading and enter identifying text for the data set into the Legend property.

The text you enter in this property is used to identify the data series in the legend for the graph.

**Note:** If the Legend property is not displayed, be sure to set the Value Source property first.

## Converting Data between Chart Types

If you change the chart type (either while in the editor or at runtime) of the widget after data value sources have been specified, Panther retains the information, even if it is not all applicable to the new chart type.

For example, if a bar graph with three data series is changed to a pie chart, only the first data series is used. If this widget is later changed to a high/low plot, all three data series are used; they become the High, Low, and Close Data Series, respectively. If converted back to a bar chart or line graph, the values once again become Data Series #1, 2, and 3.

When an XY-plot is converted to another chart type, the Y value source becomes the value source for the corresponding data series. The X value source is ignored, but it remains intact and is again used if the graph is later converted back to an XY-plot.

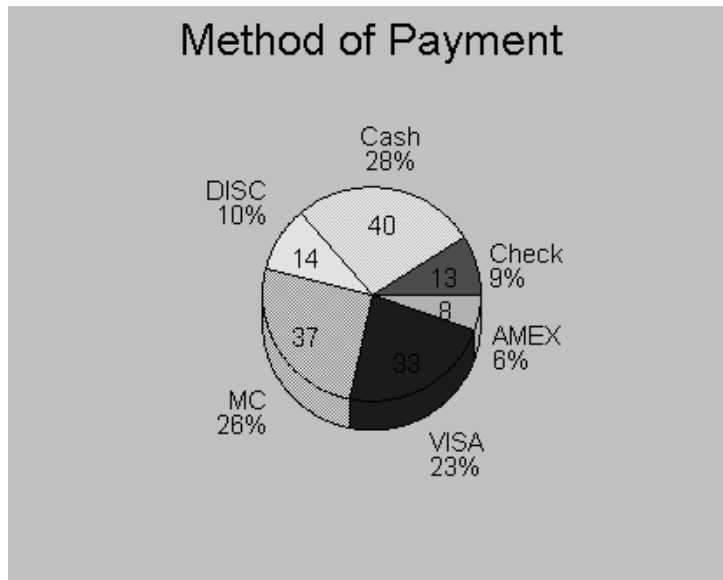
When any other chart type is converted to an XY-plot, the value source from the original chart becomes the Y value source on the XY-plot.

---

## **Pie Charts**

---

A pie chart expresses values from a single data source as proportional segments of a circle. Graph properties for a pie chart allow you to customize its format and to specify the data source.



**Figure 13-8** Pie chart drawn in 3D. Value Location property is set to In; Percent Location is Out; Label Location is Out.

## How to Create a Pie Chart

The following steps summarize the process for creating a pie chart:

1. Create a graph widget. Refer to “How to Create a Graph Widget” [on page 13-5](#) for detailed instructions.
2. Under Graph, set the Chart Type property to Pie.
3. (Optional) Add a title and subtitle to the graph. Refer to “Adding a Title to a Graph” [on page 13-9](#) for instructions.
4. Set the 3D property (for the graph widget, not the screen) to one of the following: Yes or No (default). Refer to “How to Display a Pie Chart in 3D” [on page 13-33](#) for detailed instructions.
5. Provide data for the pie chart. Refer to “How to Specify the Data Source for a Pie Chart” [on page 13-33](#) for instructions.

6. Set properties under the Pie heading to specify the pie chart layout. Refer to “Describing the Chart Layout” [on page 13-34](#) for instructions.
7. Set properties under the Pie Segments heading to specify the format and labelling of pie segments. Refer to “Describing Segment Characteristics” [on page 13-35](#) for instructions.
8. (Optional) Add a legend to the chart. Refer to “Adding a Legend to a Graph” [on page 13-10](#) for instructions.

## How to Display a Pie Chart in 3D

By default, a pie chart is displayed in two dimensions, as a flat circle. You can, however, specify that you want the chart to appear in three dimensions.

1. Select the graph widget. Under Graph, set the 3D property to Yes. Two additional properties, Vert Rotation and Depth, are displayed.
2. Set the Vert Rotation property to the desired rotation about the horizontal axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the pie chart, as if it were simply shown in two dimensions. As the degree of rotation increases, the top of the chart rotates away from the viewer so that at 90 degrees, the chart appears to be lying on a horizontal plane, and only the lower edge is visible.

The default value for Vert Rotation is 30 degrees.

3. Set the Depth property to the desired depth for the chart, expressed as a proportion of the overall size of the widget. This value must be a number between 0.0 and 100.0, inclusive. The default depth is 10.0 percent.

## How to Specify the Data Source for a Pie Chart

Data for the pie chart must come from a single source, which can be either an array on any screen in the application or constant data entered directly into the Properties window.

1. Select the graph widget.

2. Under Graph, expand the Data Series heading and set the Value Source property to either the name of an array containing the data or to a list of data values.

For detailed information on formatting the value source array name or value list, refer to [page 13-24](#), “How to Specify Data Values.”

## Describing the Chart Layout

By default, a pie chart is centered in the widget and has a diameter 30% of the widget size. The segment order is counterclockwise, beginning on the right. You can change the pie chart layout via the properties under the Pie subheading.

### How to Specify the Size of the Pie Chart

1. Select the graph widget.
2. Under Graph, expand the Pie heading and set the Diameter property to the desired diameter of the pie chart, expressed as a percentage of the overall size of the graph widget.

**Note:** This value must be a number between 0.0 and 200.0, inclusive.

At 100.0, the edge of the pie chart just touches the boundary of the widget. At 200.0, the edges of the chart are not visible. The default diameter is 30.0 percent.

### How to Specify the Position of the Pie Chart

1. Select the graph widget.
2. Under Graph, expand the Pie heading and set the X Position and Y Position properties to the desired coordinates for the center of the pie chart.

**Note:** Values for these properties must be values between 0.0 and 100.0, inclusive.

- X Position—Horizontal position within the widget. 0 is the left edge, 100-the right edge, 50-the center, and so forth.
- Y Position—Vertical position within the widget. 0 is the bottom, 100-the top, 50-the middle, and so forth.

The default position for a pie chart is centered (X Position and Y Position both set to 50).

## How to Specify Segment Order and Position

1. Select the graph widget.
2. Under Graph, set the following Pie properties:
  - **Start Angle**—Specify the desired location of the first segment. Start Angle is expressed in degrees and must be a value between 0 and 359, inclusive. The default is 0, a horizontal line to the right (3 o'clock). The values run counterclockwise: 90 points to 12 o'clock, 180 to 9 o'clock, and so forth.
  - **Direction**—Specify the order in which segments are placed around the chart. The value must be one of the following: Clockwise or Counterclockwise. The default is Counterclockwise.

## Describing Segment Characteristics

Panther provides a variety of ways you can identify the segments in a pie chart. Any or all of the following can be displayed:

- **Numeric value**—the data entered for each segment in the value source.
- **Percentage**—percent of the whole “pie” represented by this segment.
- **Label**—identifying labels, entered individually for each segment, that can appear either beside the segments or in a separate legend.

You can also control the appearance of each individual segment: normal, exploded, or hidden.

## How to Identify the Segments of a Pie Chart

1. Select the graph widget. Under Graph, expand the Pie Segments heading.
2. In the **Value Location** property, specify where you want the numeric value of the segments to appear. (These are the values specified Under **Data Series**, in the **Value Source** property.) Value Location must be one of the following:
  - **In**—The value of each segment is displayed within the segment.

- Out—(default) The value of each segment is displayed outside the segment.
  - None—Segment values are not displayed.
3. In the Percent Location property, specify where you want the segment percentages to appear. Percent Location must be one of the following:
- In—The percentage represented by each segment is displayed within the segment.
  - Out—The percentage represented by each segment is displayed just to the outside of the segment.
  - None—(default) Percentages are not displayed.

4. In the Label Source property, indicate where the segment label text is stored. The Label Source can be either:

- The name of an array containing the segment labels.

To specify an array on the current screen, simply enter the widget name. For example:

```
genre_segment_labels
```

To specify an array on another screen, enter the array name in the *screen\_name!field\_name* format, such as:

```
graph_format_screen!segment_labels
```

Any type of array can be used. However, single and multiline text widgets and list boxes are best suited to this purpose. The array can be hidden.

- A list of segment labels.

Begin the list with an equal sign (=) and separate all values with a space or other appropriate non-alphanumeric delimiter. If a non-alphanumeric character follows the equal sign, it is interpreted as the delimiter for the list; if an alphanumeric character follows the equal sign, then space is assumed to be the delimiter. For example:

```
=Comedy SciFi Adventure Drama Musical
```

```
= Comedy SciFi Adventure Drama Musical
```

```
=/Oct 14/Nov 14/Dec 14/Jan 15/Feb 15
```

(In the third example, a slash (/) is used as the delimiter since the values contain spaces.)



Labels in the array or list must be entered in the same order as the corresponding data values are specified in the Value Source property under Data Series.

Once you set the Label Source property, an additional property, Label Location, is displayed.

5. Set the Label Location property to indicate where you want the labels to appear. Label Location must be one of the following:
  - Out—(default) Segment labels are displayed beside the corresponding segments.
  - Legend—Segment labels are displayed in a legend. Refer to “Adding a Legend to a Graph” on page 13-10 for instructions.
6. Under Graph, set the Text Size property to the desired size for the text that appears in or beside the pie chart segments. The value must be a number between 0.0 and 100.0, inclusive, representing a proportion of the widget size. The default is 2.0.

**Note:** If the Label Location property under Pie Segments is set to Legend, the labels are displayed in the text size set for the legend.

## How to Specify the Style for Each Pie Chart Segment

1. Select the graph widget.
2. Under Graph, expand the Pie Segments heading and set the Style Source property to either:

- The name of an array containing the segment styles.

To specify an array on the current screen, simply enter the widget name. For example:

```
my_pie_styles
```

To specify an array on another screen, enter the array name in the `screen_name!field_name` format, such as:

```
graph_format_screen!segment_styles
```

Any type of array can be used. However, single and multiline text widgets and list boxes are best suited to this purpose. The array can be hidden.

- A list of segment styles.

Begin the list with an equal sign (=) and separate all values with a space or other appropriate delimiter. If a non-alphanumeric character follows the equal sign, it is interpreted as the delimiter for the list; if an alphanumeric character follows the equal sign, then space is assumed to be the delimiter. For example:

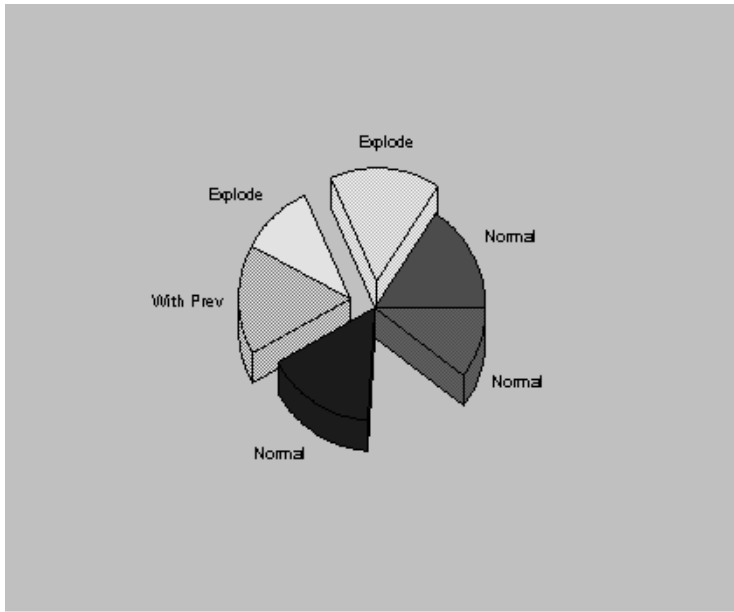
```
=/explode/with prev/normal/normal/hidden/normal
```

**Note:** In this example, a slash (/) is used as the delimiter since the values can contain spaces.

Styles in the array or list must be entered in the same order as the corresponding data values are specified in the Value Source property under Data Series.

Segment styles can be any of the following:

- Normal—(default) Narrow end of the segment is anchored to the center of the pie.
- Explode—The segment appears to be pulled out slightly from the pie and does not touch the previous segment (regardless of whether or not the previous segment is also exploded).
- With Prev—Like explode except that, if the previous segment is also exploded, the segments are pulled out together.
- Hidden—The segment is not shown, nor is its identifying text displayed.



**Figure 13-9** The Style Source property under Pie Segments is set to: `=/normal/explode/explode/with prev/normal/hidden/normal`. Notice that for a hidden segment, the label is also hidden.

---

## Bar/Line Graphs

---

A bar/line graph allows you to display data from up to 12 sources in a variety of bar and line formats. Graph properties for a bar/line graph allow you to customize its format and to specify the data value sources.

### How to Create a Bar/Line Graph

The following steps summarize the process for creating a bar/line graph:

1. Create a graph widget. Refer to “How to Create a Graph Widget” [on page 13-5](#) for detailed instructions.
2. Under Graph, set the Chart Type property to Bar. An additional property, Bar Type, is displayed.
3. Set the Bar Type property to one of the following: Absolute (default), Stack, Step, 100, Overlap. Refer to “How to Format the Bar Type” [on page 13-41](#) for an explanation of bar graph formats.
4. (Optional) Add a title and subtitle to the graph. Refer to “Adding a Title to a Graph” [on page 13-9](#) for instructions.
5. (Optional) Add a legend to the graph. Refer to “Adding a Legend to a Graph” [on page 13-10](#) for instructions.
6. Set the Orientation property to one of the following: Vertical (default) or Horizontal.
7. Describe the X and Y axes. Refer to “Describing the X and Y Axes of a Graph” [on page 13-15](#) for detailed instructions on describing and labelling axes and tick marks.
8. Set the 3D property (for the graph widget, not the screen) to one of the following: Yes or No (default). Refer to “How to Display a Bar/Line Graph in 3D” [on page 13-42](#) for detailed instructions.
9. Under each Data Series heading, set the Value Source property. Refer to “How to Specify Value Sources for a Bar/Line Graph” [on page 13-44](#) for detailed instructions.

Once you have set the Value Source property, you can specify additional characteristics of the data series:

- plot format
- Y axis
- color
- legend text
- location of value labels

## How to Format the Bar Type

Use the Bar Type property, under Graph, to define how each *slice* through the data is formatted.

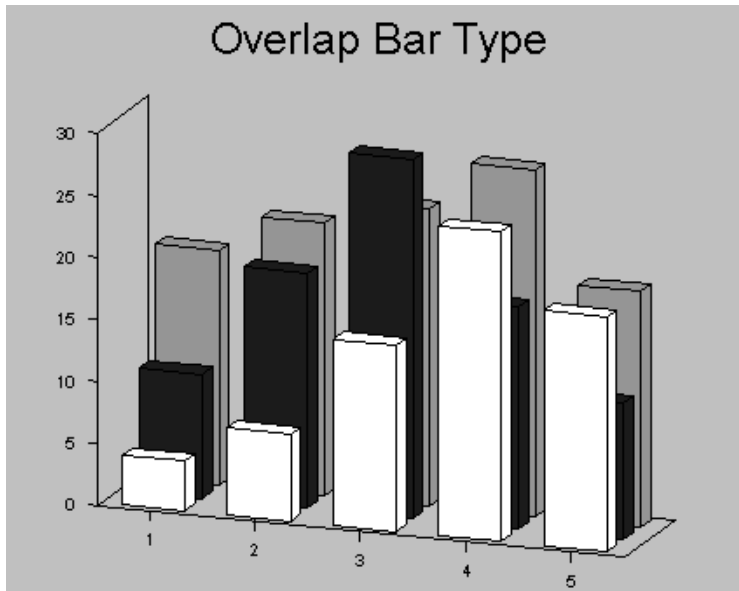
A slice is the  $n^{\text{th}}$  value of each data series in the widget, all of which are plotted at the  $n^{\text{th}}$  tick mark on the X axis. For example, the third value in each data series is plotted at the third tick mark.

Values in a slice can be formatted in a variety of ways, such as side-by-side, stacked (as if they were added together), or overlapped.

## How to Specify the Bar Type of a Bar/Line Graph

1. Select the graph widget. Under Graph, set the Chart Type property to Bar. An additional property, Bar Type, is displayed.
2. Set the Bar Type property to one of the following:
  - Absolute—(default) The values in a slice are plotted side-by-side and touching at the corresponding X axis tick mark.
  - Stack—Bars for a slice are stacked on top of each other, with the first data set starting at zero. The values within a slice are added; the total height of the stacked bar represents the sum of all values in that slice.
  - Step —Like Stack, but the bars are wider so that they touch the slices on either side.
  - 100—Similar to Stack, but the values are scaled so that all bars reach the same height, shown as 100% on the Y axis.
  - Overlap—(3D only) Bars in a slice are placed front to back instead of side-by-side. If the graph is not displayed in 3D, this setting has the same appearance as Absolute.

**Note:** Stack, Step, and 100 are best used with positive values.



**Figure 13-10** Bar/line graph with 3D property set to Yes and Bar Type set to Overlap.

## How to Display a Bar/Line Graph in 3D

By default, a bar/line graph is displayed in two dimensions, as a flat plot. You can, however, specify that you want the graph to appear in three dimensions.

1. Select the graph widget. Under Graph, set the 3D property to Yes. Three additional properties, Horiz Rotation, Vert Rotation, and Depth are displayed.
2. Set the Horiz Rotation property to the desired rotation about the vertical axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the graph. As the degree of rotation increases, the right side rotates toward the viewer so that at 90 degrees, the viewer is looking directly into the right side of the graph. At this setting, bars are visible only if they are taller than any of the bars that are “closer” to the viewer.

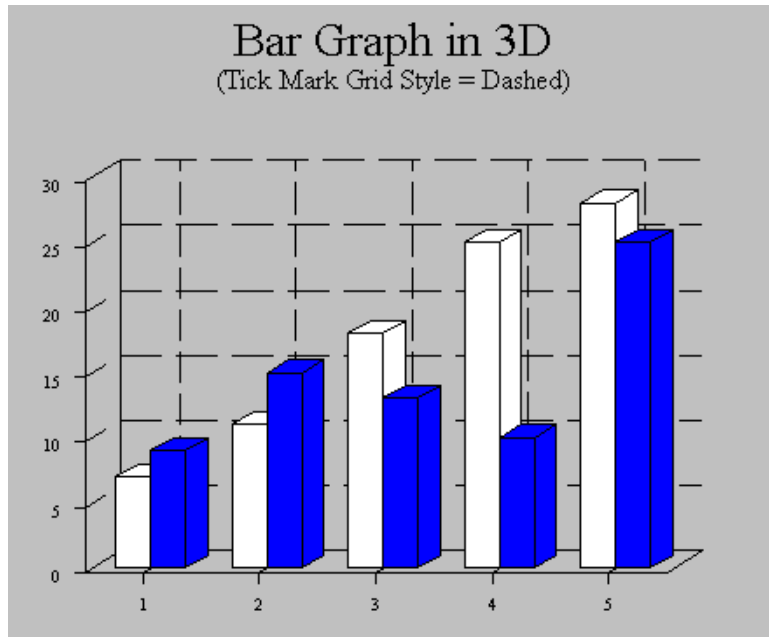
The default value for Horiz Rotation of a bar/line graph is a special value that approximates a 20 degree view angle, but the horizontal axis remains parallel to the bottom edge of the widget. At any other setting, the horizontal axis is angled to match the three-dimensional perspective. To return to the default setting if you have previously entered a number into this property, simply enter a blank space.

3. Set the Vert Rotation property to the desired rotation about the horizontal axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the graph. As the degree of rotation increases, the top of the chart rotates toward the viewer so that at 90 degrees, the graph appears to be lying on a horizontal plane, with only the top edge visible.

The default value for Vert Rotation of a bar/line graph is 20 degrees.

4. Set the Depth property to the desired depth for the graph, expressed as a proportion of the overall size of the widget. This value must be a number between 0.0 and 100.0, inclusive. The default depth is 10.0 percent.



**Figure 13-11** Bar/line graph with Bar Type property set to Absolute; 3D set to Yes; tick mark Grid Style for both X and Y1 axes set to Dashed.

## How to Specify Value Sources for a Bar/Line Graph

Each data series in a bar/line graph takes a single value source. The first value in each data series is plotted at the first tick mark on the X axis, the second at the second tick mark, and so forth.

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series heading and set the Value Source property to either the name of an array containing the data or to a list of data values.

For detailed information on formatting the value source array name or value list, refer to [page 13-24](#), “How to Specify Data Values.”



Once you have specified the value source for a data series, additional properties are displayed.

## How to Display the Values for a Data Series

For each data series on a bar/line graph, you can specify whether or not to display the value of each point or bar.

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series heading, and set the Value Location property to one of the following:
  - None—(default) Values are not displayed for this data series.
  - In—The value is displayed within the bar or area. If the plot style is line, curve, point, or trend, the value is displayed beside the data point.
  - Out—The value is displayed outside the bar or area. If the plot style is line, curve, point, or trend, the value is displayed beside the data point.

**Note:** If the Value Location property is not displayed, be sure to set the Value Source property first.

---

# XY-Plots

---

An XY-plot allows you to display data from up to 12 X and Y data source pairs. Graph properties for an XY-plot allow you to customize its format and to specify the data sources.

## How to Create an XY-Plot

The following steps summarize the process for creating an XY-plot:

1. Create a graph widget. Refer to “How to Create a Graph Widget” [on page 13-5](#) for detailed instructions.
2. Under Graph, set the Chart Type property to XY-Plot.
3. (Optional) Add a title and subtitle to the graph. Refer to “Adding a Title to a Graph” [on page 13-9](#) for instructions.
4. (Optional) Add a legend to the graph. Refer to “Adding a Legend to a Graph” [on page 13-10](#) for instructions.
5. Set the Orientation property to one of the following: Vertical (default) or Horizontal.
6. Describe the X and Y axes. Refer to “Describing the X and Y Axes of a Graph” [on page 13-15](#) for detailed instructions.
7. Set the 3D property (for the graph widget, not the screen) to one of the following: Yes or No (default). Refer to “How to Display an XY-Plot in 3D” [on page 13-46](#) for detailed instructions.
8. Under each Data Series heading, set the Y Value Source property. Refer to “How to Specify the X and Y Value Sources” [on page 13-47](#) for detailed instructions.

Once you have set the Y Value Source property, you can specify additional characteristics of the data series:

- X data source
- plot format
- Y axis
- color
- legend text

## How to Display an XY-Plot in 3D

By default, an XY-plot is displayed in two dimensions, as a flat plot. You can, however, specify that you want it to appear in three dimensions.

When an XY-plot is displayed in 3D, only the axes acquire depth; the plot lines do not. All elements, plot lines as well as axes, however, are rotated together so that the lines appear in the correct position relative to the axes.

1. Select the graph widget. Under Graph, set the 3D property to Yes. Three additional properties, Horiz Rotation, Vert Rotation, and Depth are displayed.
2. Set the Horiz Rotation property to the desired rotation about the vertical axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the graph. As the degree of rotation increases, the right side rotates toward the viewer so that at 90 degrees, the viewer is looking directly into the right side of the graph. At this setting, bars are visible only if they are taller than any of the bars that are “closer” to the viewer.

The default value for Horiz Rotation of an XY-plot is a special value that approximates a 20 degree view angle, but the horizontal axis remains parallel to the bottom edge of the widget. At any other setting, the horizontal axis is angled to match the three-dimensional perspective. To return to the default setting if you have previously entered a number into this property, simply enter a blank space.

3. Set the Vert Rotation property to the desired rotation about the horizontal axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the graph. As the degree of rotation increases, the top of the chart rotates toward the viewer so that at 90 degrees, the graph appears to be lying on a horizontal plane, with only the top edge visible.

The default value for Vert Rotation of an XY-plot is 20 degrees.

4. Set the Depth property to the desired depth for the axes, expressed as a proportion of the overall size of the widget. This value must be a number between 0.0 and 100.0, inclusive. The default depth is 10.0 percent.

On an XY-plot, only the axes acquire depth; the plotted lines do not.

## How to Specify the X and Y Value Sources

Each point on an XY-plot is identified by two coordinates—the X value and the Y value. Thus, each data series in an XY-plot has two value sources. The first entry in the Y value source is plotted against the first entry in the X value source, the second Y value against the second X value, and so forth, until all value pairs have been exhausted. Only value pairs are plotted; excess X or Y values are ignored.

1. Select the graph widget. Under Graph, expand the applicable Data Series heading and set the Y Value Source property to either the name of an array containing the data or to a list of data values.

For detailed information on formatting the value source array name or value list, refer to [page 13-24](#), “How to Specify Data Values.”

After you have entered the Y Value Source, additional properties for the data series are displayed.

2. Set the X Value Source property to either the name of an array containing the data or to a list of data values.

Alternatively, you can omit the X Value Source.

If no X Value Source is given for a data series, Panther plots the Y values against the X values for the previous data series. If the first data series in an XY-plot has no X values, Panther assumes integer X values of 1, 2, 3, etc.

---

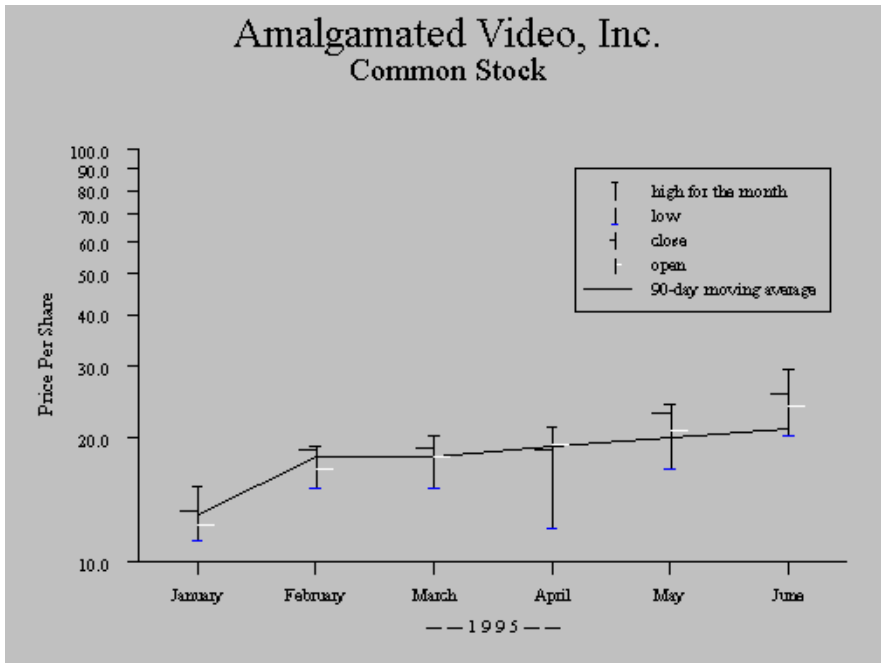
## High/Low Charts

---

In a high/low chart, the first four data series are labelled High, Low, Close, and Open. Corresponding values from these data series are plotted together on a single marker that resembles an I-beam. This type of chart is typically used to track the values of stocks, bonds, or commodities.

High/low charts can also accommodate up to eight additional data series, which are labelled numerically, beginning with Data Series #5, and are formatted in the same manner as data on a bar/line graph.

Graph properties for a high/low chart allow you to customize it and specify the data sources.



**Figure 13-12** A high/low chart. One related data series is also plotted on this graph; its Style property, under the Data Series #5 heading, is set to Line. The Y1 axis Scale property is set to Common Log, and the Y1 axis Label Location is Along Side. The legend is located in the data space.

## How to Create a High/Low Chart

The following steps summarize the process for creating a high/low chart:

1. Create a graph widget. Refer to “How to Create a Graph Widget” on page 13-5 for detailed instructions.
2. Under Graph, set the Chart Type property to High/Low.
3. (Optional) Add a title and subtitle to the graph. Refer to “Adding a Title to a Graph” on page 13-9 for instructions.

4. (Optional) Add a legend to the graph. Refer to “Adding a Legend to a Graph” [on page 13-10](#) for instructions.
5. Set the Orientation property to one of the following: Vertical (default) or Horizontal.
6. Describe the X and Y axes. Refer to “Describing the X and Y Axes of a Graph” [on page 13-15](#) for detailed instructions on describing and labelling axes and tick marks.
7. Set the 3D property (for the graph widget, not the screen) to one of the following: Yes or No (default). Refer to “How to Display a High/Low Chart in 3D” [on page 13-50](#) for detailed instructions.
8. Under each Data Series heading, set the Value Source property. Refer to “How to Specify Value Sources for a High/Low Chart” [on page 13-52](#) for detailed instructions.

Once you have set the Value Source property, you can specify additional characteristics for the data series. For the high, low, close, and open data series, you can specify the following:

- Y axis—“How to Specify the Y Axis for a Data Series” [\(page 13-29\)](#)
- color—“How to Specify the Color for a Data Series” [\(page 13-29\)](#)
- legend text—“How to Enter Legend Text for a Data Series” [\(page 13-30\)](#)

For data series 5 and above (if any are present), you can specify:

- plot format—“How to Specify the Plot Style for a Data Series” [\(page 13-26\)](#)
- Y axis—“How to Specify the Y Axis for a Data Series” [\(page 13-29\)](#)
- color—“How to Specify the Color for a Data Series” [\(page 13-29\)](#)
- legend text—“How to Enter Legend Text for a Data Series” [\(page 13-30\)](#)
- location of value labels—“How to Display the Values for a Data Series” [\(page 13-45\)](#)

## How to Display a High/Low Chart in 3D

By default, a high/low chart is displayed in two dimensions, as a flat plot. You can, however, specify that you want it to appear in three dimensions.

When a high/low chart is displayed in 3D, the high/low bars do not acquire depth. The axes, as well as the bars, lines, and/or areas from additional data series, however, do take on a three-dimensional appearance.

1. Select the graph widget. Under Graph, set the 3D property to Yes. Three additional properties, Horiz Rotation, Vert Rotation, and Depth are displayed.
2. Set the Horiz Rotation property to the desired rotation about the vertical axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the graph. As the degree of rotation increases, the right side rotates toward the viewer so that at 90 degrees, the viewer is looking directly into the right side of the graph. At this setting, bars are visible only if they are taller than any of the bars that are “closer” to the viewer.

The default value for Horiz Rotation of a high/low chart is a special value that approximates a 20 degree view angle, but the horizontal axis remains parallel to the bottom edge of the widget. At any other setting, the horizontal axis is angled to match the three-dimensional perspective. To return to the default setting if you have previously entered a number into this property, simply enter a blank space.

3. Set the Vert Rotation property to the desired rotation about the horizontal axis. This value must be a number of degrees between 0 and 90, inclusive.

At 0 degrees, the viewer is looking straight into the graph. As the degree of rotation increases, the top of the chart rotates toward the viewer so that at 90 degrees, the graph appears to be lying on a horizontal plane, with only the top edge visible.

The default value for Vert Rotation of a bar/line graph is 20 degrees.

4. Set the Depth property to the desired depth for the graph, expressed as a proportion of the overall size of the widget. This value must be a number between 0.0 and 100.0, inclusive. The default depth is 10.0 percent.

The high/low bars do not acquire depth, but all other plot elements, including the axes, do.

## How to Specify Value Sources for a High/Low Chart

Each data series in a high/low chart takes a single value source. The first value in each data series is plotted at the first tick mark on the X axis, the second at the second tick mark, and so forth. Corresponding points from the High, Low, Close, and Open data series are plotted together on a single marker. Values from additional data series are formatted as if they were on a bar/line graph.

1. Select the graph widget.
2. Under Graph, expand the applicable Data Series heading and set the Value Source property to either the name of an array containing the data or to a list of data values.

For detailed information on formatting the value source array name or value list, refer to [page 13-24](#), “How to Specify Data Values.”



# 14 Data Entry Widgets

Data entry widgets are used to accept and display text or data that the user enters. They also accept data from external sources, such as a database or an LDB screen.

- Single line text widget, by nature, accepts a single line of data.
- Multiline text widget accepts one or more lines and is enclosed within a single border.
- Combo box allows data entry and also provides the user with a drop-down list from which to choose a value.

In addition, the following widget types function as input devices. These widgets accept data that a user specifies either by selecting an option or by setting a value on the device. They do not allow the user to actually type the data.

- Option menu
- Scale

This chapter describes:

- Each of the data entry widget types and how to use them.
- How you control what keystrokes are acceptable at runtime for a widget. For example, defining if the field accepts numbers or letters and if it should be in a particular format.
- How you control the contents of the field as a whole. For example, defining what occurs when the user first enters the field, and whether data is required, converted, validated, and protected.

---

# Defining the Input with Keystroke Filters

---

The Keystroke Filter property, a property for single line and multiline text widgets and combo boxes, provides a mechanism for validating input, character-by-character, as a user types. In addition, you can define a regular expression that also validates the entire field when the field is exited at runtime. In general, if a user attempts to enter a character that does not match the Keystroke Filter specification for the field, a warning beep sounds, and the character is rejected.

By default, the Keystroke Filter property is set to Unfiltered, meaning that entry of any character is accepted when typed.

## How to Define an Input Data Filter for a Selected Widget

Under Input, in the Keystroke Filter property, select the desired filter from the Setting option menu. Possible filters are:

- Digits only
- Logical yes/no values
- Alphabetic, numeric, and alphanumeric characters
- Regular expression—A regular expression subproperty is displayed where you can enter the string expression.
- Edit mask—An edit mask subproperty is displayed where you can enter the data mask string.

### Digits Only Filter

Specifying Digits-only as a filter type:

- Allows entry of the digits 0 through 9 only.

- Allows no blanks to the left of any digit if the widget has a left-justified (default) format.
- Allows no blanks to the right of any digit if the Justification property is set to Right.
- Does not allow leading or trailing blanks.

In general, use this type of data filter for fields requiring no special punctuation or format, or if the input can be variable in length. For example, specify a digits-only filter for a field where a quantity is entered. On the other hand, if you have a field used for entering a number of fixed length and format, like a Social Security number or identification number, consider using an edit mask filter.

## How to Embed Special Characters in a Digits-Only Field

You can embed alphabetic characters, punctuation, or special characters in a digits-only field.

1. Select the data entry widget.
2. Under Format/Display, enter the format in the Initial Text property.

For example, an initial text entry of # - - displays in the field at runtime. The user can type numbers without overwriting the punctuation.

Refer to Chapter 10, “Controlling the Way Things Look,” for more information on defining the display format.

## Yes/No Entries

Specifying Yes-No as a Keystroke Filter setting allows entry of the initial letters of “yes” and “no.” This edit uses the initial letters, in upper and lower case, defined in the `SM_YES` and `SM_NO` entries in the Panther message file. You can change the message file to support non-English equivalents of `Y` and `N`. Refer to “Setting Yes/No Values” on page 45-24 in *Application Development Guide* for details on changing the default values.

By default, all of the following entries are acceptable at runtime: `y`, `Y`, `n`, `N`, or space, which is converted to `n`.

## Character and Numeric Edits

The alphabetic and numeric keystroke filters let you enforce entry of alphabetic characters, numbers, or a combination of both. In general, these filters are useful for a field that accepts data that can vary in length, like names or titles, or that requires no special punctuation or format.

### How to Embed Punctuation or Special Characters, Such as Hyphens, Periods, and Pound Signs (#)

1. Select the data entry widget.
2. Under Format/Display, specify the format in the Initial Text property. Refer to “Giving Widgets Initial Data” [on page 10-2](#) for details on defining display format.

Panther uses library functions to evaluate alphabetic and numeric input.

For information on how Panther interprets keystroke filters for multilingual applications, refer to “Keystroke Filter Translation” [on page 48-3](#) in *Application Development Guide*.

### Alphabetic Edits

Allows only a-z, A-Z, and the space character. Alphabetic filters allow the entry of a variable number of characters. For example, in a Name field, the user can enter any size name, up to the length specified for the field.

### Numeric Edits

Allows entry of digits, a plus or minus sign, and one decimal point. If a plus or minus sign is entered, it must be leftmost in the field. When a valid character is entered into the middle of an otherwise empty field, it is rejected (due to the leading/trailing blanks).

- Left-justified numbers—No blanks are allowed to the left of any character entered.
- Right-justified numbers—No blanks are allowed to the right of any character.

Numeric filters are useful when input data must be specified to be positive or negative, or when the data length is not fixed or formatted.

## Alphanumeric Edits

Allows entry of any digits, the letters a-z and A-Z, and the space character.

## Edit Masks

Edit masks let you:

- Impose a pattern of symbols or characters that limits the kind and number of characters that the user can type into a field—and are useful for data of fixed length and format, such as social security numbers and telephone numbers.
- Enforce digits-only, alphabetic, or alphanumeric input as well as the display of embedded punctuation and special characters.
- Include data characters and display characters: *display characters* are protected and serve as place savers for static text/numbers, and *data characters* allow data entry and are filtered on a character-by-character basis.

Panther strips out the display characters, or mask, from the data. Library functions like `sm_getfield` return data characters only.

## How to Define an Edit Mask

1. Select the data entry widget.
2. Under Input, set the Keystroke Filter to Edit Mask. The Edit Mask subproperty is displayed.
3. In the Edit Mask property, type the display and data characters (refer to Table 14-1) required to define the mask.

**Note:** If your edit mask includes embedded punctuation, omit the punctuation from other format properties, substring operations, and JPL procedures when validating, comparing, or manipulating the data.

**Table 14-1 Edit mask characters**

Character	Name	Function
9		Digit specification
x		Letter specification
A		Alphanumeric specification
*	asterisk	Any character
\	backslash*	Causes next character to be treated as a data character

\* To specify a backslash as display text, use two backslashes.

## How to Specify a Range or Character/Numeric Restriction with an Edit Mask

1. Select the data entry widget.
2. Under Input, set the Keystroke Filter to Edit Mask. The Edit Mask subproperty is displayed.
3. In the Edit Mask property, enter the display and data characters (refer to Table 14-1) required to define the mask. This filters the data as the user types it.
4. Under Input, in the Regular Exp property, define an expression (refer to “Regular Expressions” on page 14-7 for details) to check the pattern (for example, numbers between 900 and 999) when the field is exited.

## Examples of Edit Masks

### Edit Mask Using Data and Display Characters

The following mask represents an 11-character field; seven of the characters are data characters; the letters C, N, pound sign, and hyphen are display characters:

```
CN#\9\9\9-\9\9\9\9
```

When the user enters the field, the cursor is positioned on the first data character, in this example, on the first digit position, bypassing the display characters (CN#).

## Edit Mask on Widgets with Null Values

If a widget has an edit mask and also has the Null Field property under Format/ Display set to Yes, only the data characters are replaced by the null edit. Consider defining a null text string indicator in the Null Text property (a subproperty of Null Field), such as an asterisk, and set the Repeating property to Yes. This ensures that a field with the edit mask `CN#\9\9\9-\9\9\9\9` appears as `CN#***-****` when the field is null.

## Edit Mask on a Widget with a Data Formatting Property Set to Date/Time

Include the display characters in the edit mask, for instance, as `\9\9-\9\9-\9\9`. Specify a custom date format in the Format Type property that does not include the special characters—for example, `MON2DATE2YR2` as opposed to `MON2-DATE2-YR2`. This latter specification would result in an invalid date/time format error when the user enters a date value.

# Regular Expressions

Regular expressions let you:

- Enforce or exclude a specific pattern of letters and/or numbers, as well as a specified length.
- Restrict the field to a range or set of characters (for example, `[0-5]` allows only a number between zero and five).
- Validate data on a character-by-character basis (as a keystroke filter) or as a field validation filter (as a Regular Expression property).

You might use character-level expressions, for example, to restrict a field to a set of characters (e.g., `[0-9]`), and use the field-level expression to establish a pattern (that is, `9...` to match any number between 900 and 999).

The rules for defining a regular expression are the same for both character-level and field-level expressions, except they are stored as different properties.

## How to Define a Character-Level Regular Expression

1. Select the data entry widget.
2. Under Input, set the Keystroke Filter property to Regular Exp. The Regular Exp subproperty is displayed.
3. In the Regular Exp property, enter the expression string.

At runtime, each character is checked against the regular expression as the user types. Invalid input is rejected immediately, and a warning beep is issued. The data is also verified when the field is validated, in the event that the user deleted a character, causing the string to be invalid.

## How to Define a Field-Level Regular Expression

Select the data entry widget. Under Input, type the expression string in the Regular Expression property.

At runtime, the content of the field is matched against the regular expression when the field is validated. If the data doesn't match the specified pattern, Panther displays a message (for example, `Entry out of range`) and the cursor is automatically positioned at the invalid character if the entry does not match the specified pattern.

## Examples of Regular Expressions

`[0-9]\{3\}-[a-zA-Z0-9]\{3,6\}` defines an identification number of three-digits followed by a hyphen, followed by a minimum of three letters or numbers, but no more than six.

`[iI][cC][eE]` matches `ice`, `icE`, `iCe`, `Ice`, `IcE`, `ICe`, or `ICE`.

`212-[0-9]\{3\}-[0-9]\{4\}` matches any phone number having a 212 area code.

`[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}` matches any social security number.

`[a-zA-Z_][0-9a-zA-Z_]*` matches an identifier in the C programming language.



## Constructing Expressions

There are two kinds of rules for constructing a regular expression: one for forming simple expressions, and one for combining expressions into more complex expressions.

### Rules for Defining a Simple Expression

The simplest regular expression is a single character, which matches itself. The regular expression `z` matches the string `z`. A blank in a regular expression matches a blank in a field. Table 14-2 identifies special characters you can use in constructing a regular expression.

**Table 14-2 Special characters used in regular expressions**

Character	Name	Function
<code>\</code>	backslash	Makes any special character, including itself, ordinary. Makes <code>{}</code> , <code>()</code> and digits special in certain contexts.
<code>.</code>	dot	Matches any single character, including (but not limited to) itself.
<code>[ ]</code>	square brackets	Surround a character class expression
<code>*</code>	asterisk	Causes the preceding character class or single character expression to match zero or more occurrences
<code>\( \)</code>	quoted parentheses	Surround an arbitrary subexpression for the purpose of rematching
<code>\1</code>	quoted digits	Rematch a previous expression enclosed in quoted parentheses <code>\( \)</code>
<code>\{ \}</code>	quoted curly braces	Surround a repeat count to the preceding character class or single character expression

The backslash (`\`):

- Followed by a character forces the character to match itself even if the character is special. For instance, the sequence `\.` matches the dot, and the sequence `\\` matches a single backslash.
- Makes certain ordinary characters special; for example, backslashes that precede curly braces are quoted curly braces and are used to define a repeat count expression (refer to “Rules for Defining an Expression with a Repeated Pattern Match” on page 14-11 for information on the type of expression).

## Rules for Character Classes

Character classes are a group of characters between brackets (`[ ]`). A valid entry matches any of the specified characters within the brackets. For example, the expression `[13579]` matches any single, odd-digit; the expression `[aA]` matches an `a` of either case.

Table 14-3 lists special characters that you can include within a character class when constructing a regular expression.

- Characters in the character class are sorted by ASCII value when the expression is saved.
- Abbreviate long lists of consecutive characters by using a hyphen (`-`). For instance, `[a-z]` matches any lowercase letter, and `[A-Za-z]` matches any letter at all. Because of the ASCII collating sequence, `[A-z]` matches all letters *plus* some punctuation characters that fall between `z` and `a`.
- You can use any number and combination of characters and ranges within one set of brackets.
- To negate or restrict character classes, place a caret (`^`) immediately after the opening bracket. This causes the class to match any character *not* in the class. The expression `[^0-9+.-]` matches any non-numeric character.
- The backslash is not a special character and cannot be used as the quote character in a character class. It stands for itself.

**Table 14-3 Special characters within a character class**

Character	Name	Function
<code>^</code>	caret	Negates the character class when it immediately follows <code>[</code> (square bracket)

**Table 14-3 Special characters within a character class**

Character	Name	Function
-	hyphen	Denotes a character class range unless it is the first or last character in the class
]	square brackets	Closes the character class unless it is the first character, or the first character after an opening ^ (caret)

The following exceptions indicate when a caret, hyphen or bracket stands for itself:

- The caret stands for itself if it is not the first character.
- The closing bracket stands for itself if it is the first character, or if it is the first character after an opening caret.
- The hyphen stands for itself if it is the first or last character, or if it is the first character after an opening caret.

For example, to allow a field to accept any character *except* ^ (caret), - (hyphen), [ (opening square bracket), ] (closing square bracket), . (dot or period), or \ (backslash), use the following expression:

```
[^][.\-]
```

## Rules for Combining Two or More Expressions

Put one expression after the other. The entered data must match whatever matches the first expression, followed by whatever matches the next, and so on. Data is matched sequentially with each subexpression.

For example, an expression that comprises the following subexpressions:

```
[^xXyYzZ][a-z][a-z][a-z][a-z]
```

accepts any five-letter word that doesn't begin with upper or lower case X, Y, or Z and whose second, third, and fourth letters are lower case characters.

## Rules for Defining an Expression with a Repeated Pattern Match

You can define an expression that includes the number of times that the expression, or subexpression, is to be repeated.

- An asterisk `*` that follows an expression or subexpression means that the pattern defined in the preceding expression is to be repeated zero or more times. For example, the expression `[0-9]*` matches any number of digits or none at all, as opposed to `[0-9]` which matches only one digit in the range of zero to nine.

To force the entry of at least one digit, use the expression `[0-9][0-9]*`.

- To enforce a more definite repeat count for an expression, specify the count within quoted curly braces `\{` and `\}` immediately after the expression or subexpression. The count or number you define specifies the number of times that the expression is to be repeated. There are three forms you can use to specify a repeat number:
  - `\{n\}`— Specifies exactly  $n$  repetitions.
  - `\{n,\}`— Specifies  $n$  or more repetitions.
  - `\{n,m\}`— Specifies at least  $n$  repetitions but no more than  $m$ .

For example, `[0-9]\{5\}` matches a number of exactly five digits. While a five letter word, beginning with the letter J, could be expressed as:

```
[J][a-z]\{4\}
```

## Rules for Rematching Subexpressions

You can construct a complex expression using abbreviated syntax. These types of expressions are useful when specific repeated patterns are required as input in a field.

- Use quoted parentheses `\(` and `\)` around the subexpression. Then place a quoted digit later in your expression; for example, `\n` will match whatever the  $n$ th subexpression surrounded by `\(\)` matched.

For example, to enforce a string construction that expects the first and third characters to be the same, your expression might be:

```
\([A-Z]\)\([0-9]\)\1
```

Acceptable input would be `B5B`, `J9J`, and so on; while `A8B` is not accepted. To rematch the second subexpression as well as the first, change the expression to

```
\([A-Z]\)\([0-9]\)\1\2
```

in which case, acceptable input would be `B5B5`, `A1A1`, and so on, while input like `U2U5` and `A7B7` are not.

## Converting Case

### How to Ensure that Data is Entered in the Desired Case

1. Select the data entry widget.
2. Under Input, in the Convert Case property, specify the case style:
  - Lower
  - Upper
  - Mixed (combination of both upper and lower case characters)

As the data is entered at runtime, the characters are converted.

---

## Controlling What Occurs on Input

---

For the most part, Input properties control the behavior of the field and not the actual content of the field. These properties apply to those widgets that accept data—single line and multiline text widgets, combo boxes, and, in some cases, scales.

## Specifying a Range of Values

A range specification can ensure that a user stays within a specified range when entering data in the selected widget. Any widget that can accept data by user input, including a scale widget, can have a Minimum Value and Maximum Value property specification.

### How to Set a Range

1. Select the data entry widget or scale widget.  
**Note:** If the data entry widget has a Keystroke Filter property setting of Numeric, or if the widget's Keystroke Filter property setting is one of

Unfiltered, Regular Exp or Edit Mask and if the Data Formatting property setting is Numeric, the values entered are compared numerically.

Otherwise, the values are compared as character strings. For example, a Minimum Value of `Martin` and a Maximum Value of `Mary` accept any character string that falls within the specified range—where `Marty` is a valid entry, but `Marilyn` is not.

2. Under Input, enter a value in the Minimum Value property. At runtime, empty fields are considered within the range specification. Scale widgets require both minimum and maximum values; by default, scale widgets have a minimum setting of 0 (zero).

If you only specify a Minimum Value, then data greater than or equal to the specification are accepted at runtime and empty fields are considered to be within the range.

3. Under Input, enter a value in the Maximum Value property. At runtime, empty fields are considered within the range specification. Scale widgets require both minimum and maximum values; by default, scale widgets have a maximum setting of 100.

If you only specify a Maximum Value, then data less than or equal to the specification are accepted at runtime.

## Defining Prerequisites or Restrictions on Input Data

The following properties let you define the restrictions or behavior of a field when the user brings focus to that field at runtime.

### Required

#### How to Force Data Entry

1. Select the data entry widget.
2. Under Input, set the Required property appropriately:
  - Yes—At runtime, the user must enter at least one non-blank character before exiting the field. If the user leaves the field blank, an error message is displayed and the cursor returns to the field's first enterable position.

- No—(default) The field can be empty.

## Rules for Required Data and Null Edits

If the Null Field under Format/Display is set to Yes and the Required property is set to Yes, then the field must be non-null at runtime in order to pass validation. The null indicator string (set in the Null Text property) does not satisfy the requirement for data.

## Rules for Data Required in Arrays

For widgets that are arrays, the Required property specifies that data is required only for the number of onscreen occurrences.

## Must Fill

### How to Ensure that a Field is Filled to its Maximum Length

1. Select the data entry widget.
2. Under Input, set the Must Fill property to Yes.  
At runtime, the field is considered valid in either of the following cases:
  - The field is completely empty.
  - The field has no blanks. This includes leading, trailing, and embedded blanks.
3. To ensure that the field is completely filled and not left empty, set both the Must Fill property and the Required property to Yes.

## Select on Entry

The Select on Entry property can be useful for:

- Right-justified fields.
- Currency fields, in which editing previous data might be confusing for the user.
- Fields that have data that would tend to change completely.

## How to Control if Data is Selected when Tabbing into a Field

1. Select the data entry widget.
2. Under Input, set the Select on Entry property appropriately:
  - Yes—At runtime, the contents of the field is selected when the user first enters the field; the selected text is deleted when the user types new data. Moving the cursor from the first position deselects the text.
  - No—(default) The cursor moves to the first position in the field. New data is inserted and existing data is pushed to the right and not overwritten.

## Input Protection

By default, text widgets and combo boxes are not protected from data entry; however, you can change the property setting. At runtime, library functions can remove the input protection, while other functions can put data into fields that are protected.

## How to Control Whether Data Can be Overwritten

1. Select the data entry widget.
2. Under Input, set the Input Protection property:
  - Yes—At runtime, the user cannot type into the field; all characters are rejected and a warning beep sounds. However, the user can make selections, scroll the data in a scrolling array, or clear data.
  - No—(default) Data can be overwritten by the user.

## Protect from Clearing

All data entry and input device widgets (except for option menus which do not have this property) have the Clearing Protect property. You can protect the data in such widgets by setting the property accordingly.

## How to Control Whether Existing Data in a Field Can be Cleared

1. Select the widget.
2. Under Input, set the Clearing Protect property:



- Yes—The data in the field cannot be cleared when the user presses the CLR (clear all), FERA (clear field), DELL (delete line), or INSL (insert line) logical keys.
- No—(default) The field can be cleared of data by using the logical keys that delete or clear data.

---

## Entering Data on Multiple Lines

---

Both a single line text widget having an array specification greater than one and a multiline text widget can accept and display multiple lines of text. However, they look different. An arrayed single line text widget displays as a collection of separate widgets, while a multiline text widget's onscreen elements are enclosed within a single border.

For character mode Panther applications, you can define different border styles, a title, and horizontal scroll bars on a multiline text widget.

Refer to “Arrays” [on page 14-5](#) in *Application Development Guide* for more information about using arrays.

Both single line and multiline text widgets can have a greater number of occurrences than can display on the screen. This is referred to as a *scrolling array*. On a multiline text widget you can include a vertical scroll bar to allow users to view offscreen occurrences.

If the maximum length of the widget is greater than the display length, the widget is referred to as a *shifting field*. When the Word Wrap property under Format/Display is set to No for a multiline text widget, you can include a horizontal scroll bar on the widget to allow users to view offscreen data. Refer to “Creating Shifting/Scrolling Fields” [on page 10-13](#) for more information on defining scrolling/shifting properties and behavior.

In addition, users can view offscreen data by using vertical scroll bars, the right/left arrow keys, or the SHIFT, SCROLL, or ZOOM keys.

## Defining a Word Wrap Field

A multiline text widget, by default, accepts and displays data in a word wrap fashion. It is nonshifting and scrollable. As the user types into a multiline field, words shift between occurrences to avoid being split, and large unused portions of occurrences are eliminated. The following properties define a multiline text widget:

- Under Format/Display, Word Wrap is set to Yes.
- Under Geometry, the Length and Max Data Length are the same; therefore, the field is nonshifting and the Horizontal Scroll Bar property has no effect (except in character mode).

In word wrap fields, the following logical keys are available to aid in editing the content:

Mnemonic	Long Name	Description
WMODE	Wordwrap Mode	toggle control code display
WNL	Hard New Line	hard new line a word wrap field
WTAB	Hard Tab	hard tab in a word wrap field

**Note:** The ZOOM logical key is not supported if the Word Wrap property is set to Yes for a multiline text widget.

## Word Wrap Function

For the most part, word wrap is handled by the native GUI (Motif and Windows specifically). Therefore, word wrapped fields behave as follows:

- Arrow keys move the user up, down, right, and left within the array.
- To exit a word wrapped array, the user can click out of the field or press TAB.
- If you want each line to be validated when the user moves from occurrence to occurrence, you can set the WW\_COMPATIBLE setup variable (refer to [page 2-30](#) in *Configuration Guide*).

---

# Using Input Devices

---

Scales and option menus provide users of your application with the ability to enter data without actually typing the information. Combo boxes provide both a list from which to choose as well as a means of typing data. These input device widgets are particularly useful when you want to provide users with a range or list of choices, but with some restrictions. These devices also occupy a minimal amount of space on a screen, and therefore are useful when conservation of screen real estate is an issue.

## Using Scales



A scale widget allows a user to specify a numeric value within a predefined range of values. The scale widget can be horizontal or vertical in position and consists of:

- A slider that moves between two values.
- A label that dynamically changes to reflect the current value.

Use the library functions `sm_getfield` and `sm_putfield` to retrieve and set a value at runtime.

## How to Implement a Scale Widget

1. Select a scale widget—either horizontal or vertical.
2. Under Input, set the Minimum and Maximum Value properties to the desired values (refer to “Specifying a Range of Values” [on page 14-13](#) for more information).

By default, scales have a minimum value of 0 and a maximum value of 100. The length of the scale is determined by the maximum and minimum values and the number of decimal places you specify.

**Note:** Vertical widgets always have the lower end of the range at the lower end of the scale.

## How to Set the Scale to an Initial Value

1. Select the scale widget.
2. Under Input, set the Initial Value property to a value that falls within the minimum and maximum range specifications.

On screen entry, the scale's slider is positioned to this value and can act as a default.

**Note:** If the user chooses the CLR (clear all) or FERA (clear field) keys, the scale resets to its minimum value, as opposed to the initial value specification.

3. (Optional) To protect the initial value specification from being cleared, under Input, set the Clearing Protect property to Yes.

## How to Allow a Decimal Value Specification on the Scale

- Under Format/Display, specify a number in the Decimal Places property.
  - The default 0 forces a whole number specification on the scale
  - 2 sets the decimals to two places

## Using Option Menus and Combo Boxes

Option menus and combo boxes are similar in appearance—each has a text portion and a list portion; however, option menus are protected from data entry, while combo boxes allow users to type into the text portion of the widget.

At runtime, the user can select from these widget types by:

1. Clicking on the widget's indicator to expand the list of options or, for an option menu, entering the initial character of an item on the list.

2. With the list expanded, a selection can be made by doing either of the following:
  - Entering the first character of an item on the list.
  - Clicking on the desired item.

The selected item is displayed in the text portion of the widget.

The list of options for these widgets can be populated in two ways:

- By entering data directly in the Properties window.
- By identifying a screen that contains the data (which can be populated from a database or from an LDB).

To determine what selection is made or entered (if a combo box) by the user, use the library function `sm_e_getfield`.

## Populating the List with Constant Data

### To Populate an Option Menu or Combo Box List with Static Data

1. Select or create an option menu or combo box.
2. Under Identity, specify Constant Data (default) in the Drop-Down Source property.
3. Select the Drop-Down Data property. The Drop-Down Data dialog box opens.
4. Enter or edit data by doing any of the following:
  - Type the data directly in the dialog box window.
  - Choose Editor to invoke your local editor and enter the desired data.
  - Choose Read File to import an external file.

This data is displayed when the user clicks on the widget's indicator.

In Motif, the drop-down list displays all occurrences to their full length. In character mode and Windows, the list width is determined by the length/width of the option menu widget itself, and therefore, some occurrences may appear truncated.

## Defining the Number of Occurrences

For applications running in character mode and Windows, you can define the number of entries that are immediately visible when the user expands an option menu or combo box. This property setting is particularly useful if the data list is extensive or screen space is limited.

**Note:** In Motif, option menus and combo boxes do not scroll; therefore, the Drop-Down Size property has no effect.

### How to Define the Number of Occurrences that Display on an Option Menu or Combo Box (for Windows and Character Mode Applications)

- Under Identity, specify the number of occurrences to be displayed in the Drop-Down Size property.

A blank value causes all occurrences to be displayed. If the number is less than the total number of occurrences, the drop-down list will have a vertical scroll bar so the user can scroll to the offscreen occurrences.

## Using Data from an External Source

You can populate an option menu or combo box from an external source, for example, a screen that is dynamically populated from a database. You must also create the external source screen.

### How to Populate an Option Menu or Combo Box List from an External Source

1. Select an option menu or combo box.
2. Under Identity, specify External Screen in the Drop-Down Source property.
3. In the Drop-Down Screen property, type the screen name or choose More and select it from the Select Library Member dialog box.

The screen you specify should consist of a list box (or any other array widget) that can have either a specific or infinite number of occurrences. It can be populated dynamically at runtime from still another source, such as another screen (LDB, local data block) or a database.

4. Under Identity, in the Initialization subproperty, choose either of the following settings:

- **Fill at PopUp**—(default) Invokes the external screen only when the user activates the combo box or option menu. If the external screen is populated as the result of a database query, you can ensure that the list contains the most up-to-date information. This specification does not force the opening and closing of the external screen until user-activated.
- **Fill at Init**—Opens the external screen when the selected combo box or option menu is initialized—that is, the external screen populates the widget when the screen opens. If the external screen, for example, is populated from a database, the data is retrieved any time the widget is initialized. Opening and closing the external screen can take time, particularly if a database query is involved. On the other hand, if the data on the external screen are not likely to change while the current screen is displayed or if other widgets on the screen use the same external screen, use this specification.

## How to Create an External Source from a Database Column

Create a screen that holds a database-derived widget that you copied from a repository entry. The screen will contain a simple screen entry function that invokes a `VIEW` command, and fetches the required data.

1. Select the widget in the untitled screen.
2. Under Geometry, set the Array Size property to a number greater than 1.
3. Under Geometry, set the Scrolling property to Yes.
4. Set the Max Occurrences subproperty to the number of occurrences to be read in. A blank value allows for an unspecified number of occurrences (the actual maximum number is platform-specific).
5. Select the screen (click in an empty area of the screen to deselect the widget).
6. Under Focus, select JPL Procedures. The JPL Program Text dialog box opens.
7. Type a simple procedure to call a `VIEW` transaction command, for example:

```
proc procedureName
{
call sm_tm_command("VIEW")
}
```

Choose File→Close→JPL/Javascript to save the procedure and to return to the Properties window.

8. Under Focus, type the *procedureName* in the Entry Function property.
9. Save the screen (this is the name you provide in your option menu's Drop-Down Screen property).

## How to Create an External Source from an LDB (Local Data Block)

Create a screen that holds one widget that has more than one occurrence (list box, multiline text, or single line text with an Array Size property value greater than 1). Refer to “Using Local Data Blocks” on [page 25-7](#) in *Application Development Guide* for more details on using LDBs.

When a widget gets its data from an LDB, the widget on the external screen should be initialized with no data.

## Defining Initial Setting

Only the top portion of a combo box and option menu displays when the screen opens at runtime.

## How to Define What Displays in the Text Portion of an Option Menu or Combo Box

1. Select the option menu or combo box.
2. Under Format/Display, enter the desired text string in the Initial Text property.

The data can be one of the items in the drop-down portion of the widget, and can serve as a default. If you clear the Initial Text property of any text, on entry to the screen, the option menu/combo box will be blank.



# 15 Grid Widgets

A grid widget is a two-dimensional array, displayed as columns and rows, that allows users to scroll data both vertically and horizontally. It can be particularly useful for displaying data in a spreadsheet fashion and for displaying database query results in a tabular arrangement.

With a grid widget, you can display an unlimited number of rows associated with a specified set of columns.

---

## Creating and Editing a Grid Widget

---

A default grid widget consists of a frame, numeric row designators, alphabetic column titles (visible once a widget is added to the grid), and horizontal and vertical scroll bars. By default, all members of the grid are synchronized to scroll together. Grid widget properties define the grid as a whole and, to some extent, the widgets that are contained within the frame.

### How to Create and Populate a Grid Widget

1. Choose Create→Grid Frame or choose the grid frame icon from the Create toolbar.
2. On your application screen, click once to create a default-sized widget or drag out the grid widget to the desired dimension (which you can change later).

The default grid allows for the display of five rows (Onscreen Rows property) and up to 999 occurrences (Max Occurrences property).

3. (Optional) Under Identity, assign a name to the grid widget in the Name property.

Panther numbers grid widgets sequentially from left to right, and top to bottom—as Grid #1, Grid #2, and so on. These numbers are for the Editor's use only and cannot be used to access the grid widget itself at runtime.

4. Add widgets (dynamic labels, single line text, or list boxes) to the grid widget by doing any of the following:
  - Drag an existing widget or widgets to fall within the grid widget border. Widgets can be copied from within the current screen, from another screen, or from an open repository entry.
  - Choose the widget from the Create menu or select the icon from the Create toolbar and click within the grid widget border.
  - Select a widget on the screen and under Identity, specify the name of the grid in the Grid property (this property is available only when a grid widget is on the same screen as the selected widget).

Each widget is added as the last (rightmost) array, or column, in the grid. The column acquires its number of onscreen rows and number of occurrences from the grid widget itself. Panther internally numbers each column with a field number. Default titles are assigned to the columns, alphabetic identifiers starting with the letter A and going from left to right.

---

## **Manipulating a Grid Widget and Grid Members**

---

### **How to Select a Grid Widget**

Rubberband a portion of the grid widget, or

- In Motif, click on an area of the grid widget, not on a member of the grid.
- In Windows, click anywhere on the grid, except on a row or column title.
- Select it using the Widget List.

When you move, copy, or delete a grid widget, the members of the grid are moved, copied, or deleted as well.

## How to Resize a Grid Widget

Select the grid widget. Do either of the following:

- Drag it by a resize handle to the desired size.
- Under Geometry, set the Onscreen Rows and Onscreen Columns properties to the desired number of rows and columns.

## How to Select One or More Grid Members

Do any of the following:

- Under Motif, click on the desired grid member; Shift+click to select more than one grid member.
- Under Windows, click on the desired grid member's column title; Shift+click to select more than one grid member.
- Select the desired grid member (Shift+click to select more than one widget) from the Widget List.
- Select the grid widget, and then choose Edit→Group→Select Members to select all members of the grid.

## How to Remove a Grid Member from the Grid

Do any of the following:

- Select the grid member. Under Identity, set the Grid Property to None.

- In Windows, click on the grid member's title to select the column and drag the selected widget beyond the border of the grid widget.
- In Motif, select the grid member by clicking on one of its row separators (the mouse pointer appears as a right-left arrow indicator), and drag the widget beyond the border of the grid widget.

## How to Delete the Grid Widget, but not the Widgets within the Grid

1. Do either of the following:
  - Select the grid members. Under Identity, set their Grid property to None.
  - Select a grid member, and drag it outside of the grid border.
2. When all grid members have been moved from the grid widget, select the grid widget and choose Edit→Delete or Cut (or the Cut button from the toolbar).

---

# Grid Widget Features

---

Grid widgets consist of columns and rows. The columns are created by dragging one or more of any combination of the following widget types into the grid widget border:

- Dynamic label
- Single line text
- List box

You determine how many columns are visible on the screen. However, this does not define how much data the grid widget as a whole can contain. Therefore, you can define both onscreen dimensions and offscreen specifications. The background color of the stripe row is defined by the Color Control Panel and corresponds to the *highlight color*.

## Specifying Grid Fonts

You can assign a font for the grid widget by setting the Font Name property under the Font heading. This specification applies to the column titles and row titles if they are used. The font also affects all members of the grid, unless the individual grid member has its own font specification.

**Note:** Under Motif, the grid members always acquire the font specification from the grid widget itself. In other words, the font specification on individual grid members is ignored.

## Displaying Columns

### Defining the Grid's Width

If the grid as a whole contains more columns than can be displayed at one time, you can use a horizontal scroll bar to allow users to shift the columns from left to right and vice versa. In addition, you can include columns that are wider than the grid widget itself. At runtime, the user can expand such fields by using the ZOOM logical key.

### How to Control the Onscreen Width of the Grid Widget

1. Create a grid widget and add the widgets that will make up the grid.
2. Select the grid widget.
3. Under Geometry, in the Onscreen Columns property, enter the number of columns that you want to display. This number defines how many columns are fully visible in the grid widget when it is fully shifted to the left.

In the screen editor, the grid widget adjusts to the correct size to show the specified number of columns exactly (assuming the screen can accommodate the total width of the columns).

If the Onscreen Columns property is blank, partial columns or blank space can display on the right side of the grid widget when it is fully shifted to the left.

### How to Change the Column Order

When you add a widget to a grid widget, it is automatically positioned to the right of all other columns. Select the grid member that you want repositioned, and do either of the following:

- Under Identity, enter the desired position by number in the Grid Column subproperty.

**Note:** If the grid's Rows Title property is set to First Column, entering 1 in the Grid Columns property defines the selected column as the rows' titles.

- Drag the column right or left beyond the adjacent column.

**Note:** If you drag the column beyond the grid widget border, the widget is removed from the grid and becomes a screen widget.

### How to Control How the Grid Can Be Shifted from Left to Right to Display Offscreen Columns

1. Select the grid widget.
2. Under Geometry, set the Horizontal Scroll Bar to the desired setting:
  - Yes—(default) Provides a horizontal scroll bar whereby the user can click on the scroll bar to shift the display of columns from left to right and vice versa. You can use the scroll bar while you are in Edit mode.

Shifting by column shifts in such a way to allow for whole columns to display.
  - No—No scroll bar is displayed. The column display can be shifted by using the right and left arrow keys or Tab.

### Displaying Column Titles

Each column of the grid, by default, is displayed with an alphabetic column title, starting with the letter A. The column title displays at the top of the grid and remains fixed above the rows of data that scroll below them.

### How to Define Column Titles

1. Select the grid widget.

2. Under Format/Display, specify the source/type of column heading in the Column Titles property:
  - Auto Number—Each column is automatically numbered sequentially from left to right beginning with 1.
  - Auto Letter—(default) Each column is automatically assigned an alphabetic heading, beginning with A...Z, AA...AZ, BA...BZ, ..., ZA...ZZ.
  - Per Column—Uses the name designated in the individual widget's Column Title property (under Identity). Widgets derived from database columns automatically derive their Column Title property setting from the Name property.
  - None—No titles appear over the columns.

## Defining Stationary Columns

When a grid widget contains a large amount of data, sometimes shifting the columns from left to right causes vital information to be shifted outside the grid's viewport. You can define how many columns should be stationary when the user tabs or shifts the columns horizontally.

### How to Define Stationary Columns

1. Select the grid widget.
2. Under Format/Display, enter the number of columns that should be stationary in the Frozen Columns property.

The default is 0.

Columns are frozen from the left. So, if you enter 2, the first two columns in the grid are stationary. The remaining columns can be shifted.

The user can enter data into columns that are frozen.

If only frozen columns are visible, the other columns will not be accessible. Therefore, you will need to resize the grid to display at least one nonfrozen column.

## Defining Column Separators

Grid widgets display with vertical and horizontal lines to provide definition to each grid cell. You can display columns of data with or without these separators.

## How to Define the Display of Column Separators

1. Select the grid widget.
2. Under Format/Display, set the Column Separators property accordingly:
  - Yes (default)—Displays vertical lines between each grid member column.
  - No—Causes the grid to display with no vertical lines.

## Displaying Rows

### Defining the Number of Rows

The number of onscreen rows determines the height of the grid widget and how many rows, or records, are displayed at a time. If there are more rows than can be displayed within the grid widget, use a vertical scroll bar to allow users to scroll the data from top to bottom and vice versa.

### How to Define the Number of Rows that Should Be Displayed at Runtime

1. Create a default-size grid widget or drag it out to an approximate size that will allow the desired number of rows to display. Doing this in Windows may cause problems since that grid widget works best when no partial rows are displayed.
2. Add the widgets that will make up the grid.
3. (Optional) Under Geometry, enter the desired number of rows that you want to display in the Onscreen Rows property. This value overrides any array-specific properties you may have previously set on individual grid members.
4. Under Geometry, enter the total number of occurrences in the Max Occurrences property.

This value overrides any value that you have previously set for individual widgets that are now members of the grid. A blank value returns an unlimited number of rows (default is 1000 as set by `max_fetches`).

5. Under Geometry, set the Vertical Scroll Bar property to the desired setting:
  - Yes—(default) Allows the user to use the scroll bar to display data in offscreen rows.



- No—The user can scroll data by using arrow keys or Page Up/Page Down keys to display offscreen rows.

This value overrides any scrolling-specific property settings that you have previously set for individual widgets that are now members of the grid.

6. (Optional) Under Geometry in the Scroll Increment property, enter the number of occurrences by which the rows should scroll when the user presses the Page Down key and Page Up key.
7. (Optional) Under Geometry in the Circular property, specify the desired behavior:
  - Yes—Causes the first row to redisplay after the last element in the grid is reached, and vice versa when the user arrows or pages up/down to scroll the rows.
  - No—(default) If the user uses the arrow keys to move the cursor to the last/first row, the cursor moves to the next or previous field. If the user presses Page Down/Up to scroll the rows, Panther displays the message "No more data" when the first/last row is reached.

## Displaying Row Titles

Each row of the grid, by default, is displayed with a numeric row designator that corresponds to the setting of the Onscreen Rows property under Geometry. You can display a more useful row indicator by defining the grid's row titles.

### How to Display Titles for Rows of the Grid

1. Select the grid widget.
2. Under Format/Display in the Row Titles property, specify the title source/type:
  - Auto Number—(default) Displays the number corresponding to the row number beginning with 1 (one). As the data is scrolled, the numbers increment appropriately until the last row is reached.
  - First Column—The first data column is used as the row titles. The data is used from this column, but all other display attributes are ignored. In addition, the column can no longer receive focus in Edit mode (use the Widget List to gain access to the widget's properties) or at runtime. All other columns in the grid adjust accordingly; the leftmost column in the grid is considered to be the second column in the grid.

- None—No row titles are displayed.

## Selecting Rows of Data

By default, all members of the grid widgets scroll synchronously. The assumption is that a row in the grid corresponds to a record in the database, or at least, each element of the row has related information. You can allow a user to click on a single item to select just the one cell or click on a single item to select the entire row. When an item is selected, it is displayed in reverse video or is highlighted.

### How to Control What a User Selects in a Grid

1. Select the grid widget.
2. Under Format/Display, set the Stripe Current Row property to the desired value:
  - Yes—(default for grid created with the screen wizard) At runtime, the user can select a row by clicking on any one item in any column and, as a result, the entire row is highlighted. The Up and Down arrow keys will also move the stripe up and down the arrays.
  - No—(default) Individual occurrences, or cells, can be selected. There is no highlight bar.

## Defining Row Separators

Grid widgets display with vertical and horizontal lines to provide definition to each grid cell. You can display rows of data with or without these separators.

### How to Define the Display of Row Separators

1. Select the grid widget.
2. Under Format/Display, set the Row Separators property accordingly:
  - Yes (default)—Displays horizontal lines between each grid member row.
  - No—Causes the grid to display with no horizontal lines.

## Changing Row Height

The height of a grid row is scaled in relation to the font size. You can change the height of a row by adjusting the space, or the margin, between the text and the row dividers.

The default row margin is dependent on the grid widget's Row Separator property:

- With row separators, the default is 2 pixels.
- Without row separators, the default is 0 pixels.

### How to Adjust The Row Margin

- For a single grid widget, under Geometry, enter a value for the Row Margin property. The default unit of measurement is character units. Refer to [Table 9-1 on page 9-7](#) for a list of valid units of measurement.
- For all grid widgets, set the application property `default_row_margin`. The default units for this application property are pixels.

**Note:** Applications written in previous versions of JAM, Panther, and Panther can preserve the original row height behavior by setting the application property `default_row_margin` to 6 in Windows or 5 in Motif.

## Defining Column Click Behavior

You can define the behavior that occurs when a user clicks on the column header. The Column Click Action property can be defined for any widget in a grid.

**Note:** Change the Column Click Action property on widgets in the repository to allow the application's widgets to inherit the new value.

## Sorting Items

To specify the column sort order:

1. Select the column. Under Format/Display, set the Column Click Action property to Sort.
2. Set the Sort Order property as desired:

- Lexicographic —An alphanumeric sort order which allows for better sorting of international font sets.
- Numeric—The Data Formatting property, under Format/Display, must also be set to Numeric.
- Date/Time—The Data Formatting property, under Format/Display, must also be set to Date/Time.
- Custom—If the Sort Order property is set to Custom, a Sort Order Function property subheading will appear. Enter the name of your custom function.

In Windows and in character mode applications, the field used to sort the grid rows will be indicated by a small arrow in that field's column heading. The direction of the arrow indicates the direction of the sort—an up arrow for an ascending sort, a down arrow for a descending sort.

**Note:** The sort function was not compatible with the transaction manager before Panther 4.60.

## Writing a Custom Sort Function

The Sort Order Function property, under Format/Display, is the name of a JPL procedure or installed C function that you have written to compare two elements in the sort. After evaluating the two string arguments according to their positions, the function must return one of the following:

- COMPARE\_LESS—the first argument is less than the second one.
- COMPARE\_GREATER—the first argument is greater than the second one.
- COMPARE\_EQUAL—the two arguments are equal.
- COMPARE\_ERROR

Since the arguments to this function will be strings, to implement numeric sorts, you will have to convert the strings to appropriate numeric values.

## Specifying a Function

Alternatively, you can specify your own function to define the behavior that occurs when the user clicks on the column heading.

## How to Define a Custom Click Function

1. Select the column.
2. Under Format/Display, set the Column Click Action property to Custom. The Column Click Function property will appear.
3. Enter the name of your custom function.

The Custom Click Function is passed a single parameter, which is the object ID of the field whose column heading was clicked. The value it returns is ignored. The function can be written in JPL or can be an installed C function.

As an example, to do a normal columns sort and then perform other operations, `sm_obj_sort_auto` can be called to do the sort and the other operations can then be performed.

---

# Moving and Resizing Grid Columns at Runtime

---

The grid widget's dimensions determine the overall size of the frame and how many rows and columns can be displayed. You can make grid members very small or very large; however, the data within the cells of the column might not be completely visible. You can allow users to both move and resize individual columns horizontally in the grid as well as horizontally expand individual cells to see the contents.

## How to Allow Users To Move and Resize Grid Members

1. Select the grid widget.
2. Under Geometry, set the Column Move Resize property accordingly:
  - Yes (default)

The user can move a column:

- In Windows, by clicking on the column's title and dragging it either left or right in the grid.
- In Motif, by clicking on the column's row separator (the mouse pointer appears as a right-left arrow indicator) and dragging the column either left or right in the grid.

**Note:** To ensure that Windows users can, in fact, move grid members, the grid's Column Titles property must be set to Auto Number, Auto Letter, or Per Column.

The user can resize a column horizontally by dragging on the column's right column separator (the mouse pointer appears as a right-left arrow indicator).

- No—The grid columns cannot be move or resized at runtime.

## How to Expand a Single Grid Cell to Its Maximum Length

1. Click in the desired cell
2. Choose the ZOOM logical key.

A zoom window opens where the user can view, edit, or enter data (if applicable).

---

## Grid Focus Properties

---

The following Focus properties can be set for grid widgets:

## **Entry and Exit Function Properties**

The grid's entry function is executed when the cursor first enters a field the grid. The order of function calls is: grid entry, group entry, grid row entry, and field (widget) entry.

The grid's exit function is executed in the reverse order. The grid exit function is called when the cursor leaves the grid.

Grid entry and exit functions are passed an occurrence number, which is always zero.

### **Focus Protection Property**

Focus protection set on a grid prevents the cursor from moving into the grid, regardless of how individual members have focus protection set. However, if the grid widget is not protected (Focus Protection is set to No), then the focus protection setting for individual columns is honored.

### **Row Entry Function Property**

The row entry function is executed after the grid entry function and any group entry function, but before the field entry function.

### **Row Exit Function Property**

The row exit function is executed in reverse order of the row entry function. The row exit function is called after field exit functions, and before any group exit function or the grid exit function.





# 16 Tab Controls

A tab control widget contains one or more cards which are accessed by clicking on a graphic index tab associated with each card. This is particularly useful for providing a multi-layered control panel.

**Notes:**

- This widget is only available for Windows and Motif applications.
- Tab decks (and tab cards) are not interchangeable between the two platforms. Screens using Motif tab decks must run on the Motif platform; screens using Windows tab decks must run on the Windows platform.

---

## Creating and Editing a Tab Control

---

A tab control consists of two kinds of elements:

- The *tab deck* widget: a container for a series of tab cards. When you initially create a tab deck, two tab cards are created within it.
- The *tab card* widget: a container on which additional widgets (such as buttons, text fields, grids) can be placed. Associated with each tab card is a field that is rendered as the card's *index tab*.

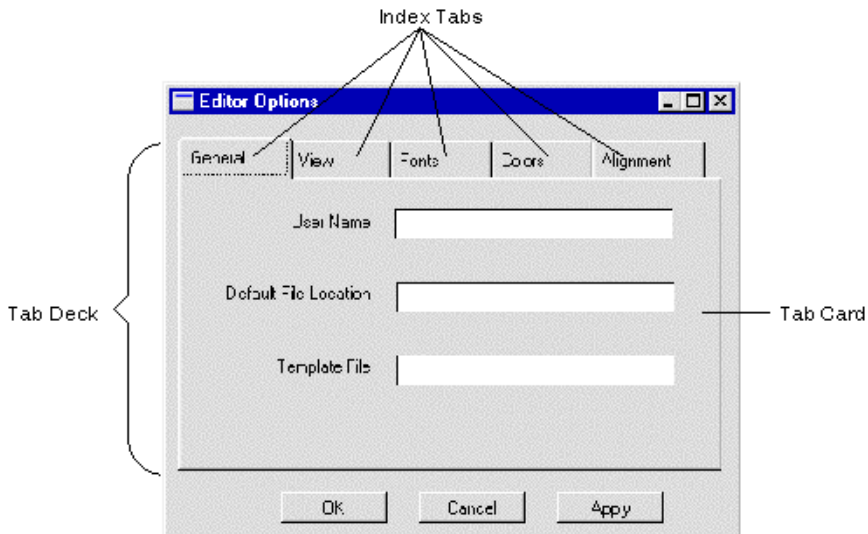




Figure 16-1 A sample tab dialog screen illustrating the parts of a tab control.

## How to Create and Populate a Tab Control

1. Choose Create→Extended Widgets→Tab Deck or .
2. On your application screen, click once to create a default-sized tab control or drag out the frame to the desired dimension. Initially the control consists of a tab deck and two tab cards.
3. Add additional tab cards by choosing Create→Extended Widgets→Tab Card and then clicking on the deck; the tab card icon is .

Where the mouse click occurs in relation to the existing index tabs determines the placement of the new tab card. Clicking to the far right adds it to the end of the deck, clicking to the far left adds it to the beginning of the deck, and clicking in the middle of the tab deck adds it between two index tabs of existing tab cards.

**Note:** A new tab card cannot be placed outside the borders of a tab deck.

4. Conversely, you can delete one or both of the initial tab cards to create a deck of one card or no cards, respectively.
5. Add widgets (such as dynamic labels, single line text, check boxes, and so on) to the topmost tab card by doing either of the following:
  - Drag existing widgets to fall within the tab deck border. Widgets can be moved and copied from within the current screen, from another screen, or from an open repository entry.
  - Choose the widget from the Create menu or select the icon from the Create toolbar and click within the tab deck border.

---

## Manipulating a Tab Deck and Tab Cards

---

### How to Select a Tab Deck

- Click on the topmost card anywhere *below* the index tab.

### How to Make a Tab Card the Topmost Card

- Click once on the card's index tab to bring the card to the top of the deck.

### How to Select a Tab Card for Editing

- With a tab card already positioned as topmost card, click on the card's index tab to bring focus to the card.

**Note:** Two clicks are required to bring a card into focus in order to allow multiple selection of items appearing on different cards. (For instance, you can select an item from one card, then click a tab on a different card to make it the topmost, then shift+click to select a second item on the new topmost card.)

## How to Change the Tab Card Order

You can change the order (left to right) in which the tab cards appear.

- Change the card's number property in the Properties window (refer to the description of the Card Number property [on page 16-8](#) for more information).

## How to Move, Copy, or Delete a Tab Deck

When you move, copy, or delete a tab deck, the members of the tab deck are moved, copied, or deleted as well.

## General Notes About Tab Controls

While working with tab controls, be aware that:

- Only widgets that lie within the geometrical boundaries of a tab deck can belong to a particular card in that tab deck.
- A widget cannot belong to more than one tab card.
- A tab card cannot exist *outside* of a tab deck.
- An entire tab deck—with its related tab cards—can be nested within a single tab card.
- If you select a group of widgets on multiple cards, you can simultaneously align them with the alignment commands (such as Edit→Align).
- Widgets cannot be “rubberbanded” with the mouse in order to select multiple widgets on a tab card.
- The logical keys `NCARD` and `PCARD` move to the next and previous tab cards, respectively.
- The index tabs operate differently in Windows and Motif applications. Refer to the “Label” description [on page 16-7](#).
- Only Motif 2.2 supports having both text *and* an image on a tab. Earlier versions can only have text *or* an image.

---

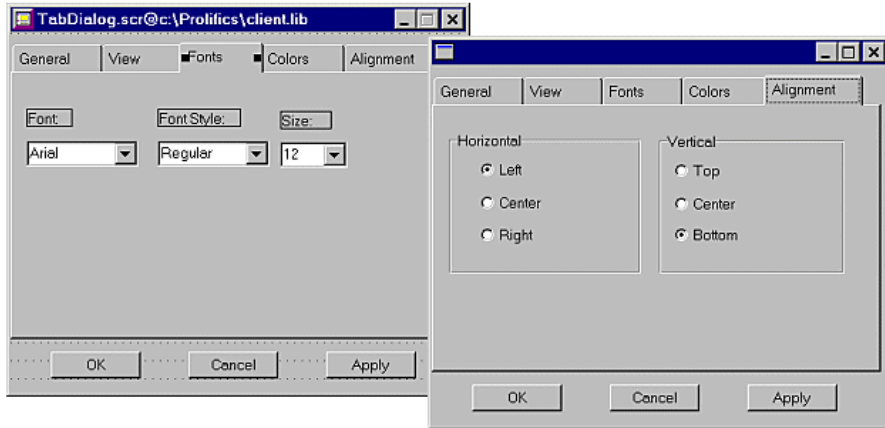
# Creating a Tab Dialog Screen

---

A typical use for tab controls is in dialog windows. Such a dialog window would consist of a single tab control. Each card in the tab deck would provide a series of user-selectable choices. Outside of the tab control, there would typically be a series of push buttons to indicate user acceptance of the choices made.

## How to Build a Tab Dialog Screen

1. Create a tab deck on a blank screen.
2. Click on the index tab of a card to bring it to the top.
3. Add widgets (such as text widgets, check boxes and option menus) for that particular card's function.
4. Create selection groups for check boxes and radio buttons in order to limit the number of selections within a group (refer to “Grouping Selection Widgets” [on page 20-10](#)).
5. Visually group widgets within a tab card by creating a box widget or line widget (refer to “Using Boxes and Lines” [on page 21-1](#)).
6. Create push buttons for the screen below the tab deck. With the desired push button selected, set the Identity→Default/Cancel property to specify the push button's behavior (refer to “Assigning Default and Cancel” [on page 19-6](#)).
7. With the screen selected, set the Identity→Dialog property to Yes (refer to “Creating a Dialog Box” [on page 6-5](#)).



**Figure 16-2** Two views of a tab deck, showing two of the cards in the deck. Notice the push button controls which are placed on the screen outside the tab deck.

---

## Tab Control Properties

---

This section describes special properties appearing in the Properties window for tab cards and tab decks. In addition, there are runtime properties for tab controls. For more information, refer to “Accessing Tab Controls” on page 23-12 in *Application Development Guide*.

### Tab Deck Properties

The following properties of tab decks require special mention:

## Identity

### Hidden

If set to yes, this hides the entire tab deck and overrides any Hidden property settings you have made for individual cards.

### Conceal Tabs

Hide the index tabs at runtime (decks will still have visible tabs in the editor). By choosing Yes, the topmost card in the deck is displayed at runtime without an index tab.

If set to Yes, users will be unable to change the tab card and the logical keys `NCARD` and `PCARD` will not advance the card; but the top card can be changed by setting the deck's `topmost_card` property.

## Color

You cannot specify the color of the tab deck itself. It is derived from your Windows color scheme. Setting `BG Color` affects only the background color of label widgets placed on the tab deck.

## Font

Render the labels on the index tabs. A single setting affects all the index tabs in a particular deck.

## Tab Card Properties

For tab cards, the following categories in the Properties window should be noted:

### Identity

#### Label

Enter the text to appear on a card's index tab.

**Note:** For Motif applications, all the index tabs in a tab deck have the same length. Also, the index tabs will not be resized when the labels are shortened. Only Motif 2.2 supports having both text *and* an image on a tab.

### Hidden

As with other widgets, the Hidden property can be set to Yes/No/Always, allowing individual cards to be either displayed or not at runtime. The Always value, however, has no relevance to the positioning of the tab cards (which it does with other widgets). A tab card which has its Hidden value set to Always cannot have its value set to No at runtime.

### Card Number

Indicate where in the sequence of cards a particular tab card is located. Cards appear left to right based on a the Card Number property. If you change a card number, the new value must be between 1 and the total number of cards on the deck. The card numbers for all the cards in the deck will be readjusted to reflect the new ordering.

## Geometry

### Length

Determine the width of the card's index tab.

## Color

Setting the Color properties on the tab card has no effect in Windows. It is derived from your Windows color scheme.

## Focus

Tab cards have three sets of entry and exit events. To understand the distinction between them, keep in mind that a tab card consists of two distinct components: a field that is displayed as the card's index tab, and a set of widgets that the tab card contains. At runtime, the index tab of the topmost card in a deck is one of the fields on the screen and can gain focus. Consequently, the index tab has its own entry and exit events.

For more information about entry and exit functions specific to tab cards, refer to “Tab Control Events” [on page 17-18](#) in *Application Development Guide*.

## Card Entry and Exit

For the Card Entry and Card Exit Functions, the index tab field is considered to be a widget “contained on the tab card.”



#### Card Entry Function

Enter the name of the function to be called when one of the widgets contained in the tab card is about to gain focus.

#### Card Exit Function

Enter the name of the function to be called when focus leaves one of the widgets contained in the tab card *and* a widget not contained in the tab card is about to gain focus.

## Expose and Hide Functions

In addition to the events that occur when a tab card is exposed or hidden, the Expose Function is called on screen entry for the card that will be topmost, and the Hide Function is called on screen exit for the current topmost card. These events occur even if the tab deck is hidden. However, the Expose and Hide Functions do not execute when the tab deck's Hidden property is used to hide or unhide a tab deck.

#### Expose Function

Enter the name of the function to be called when the card becomes the topmost card.

#### Hide Function

Enter the name of the function to be called when another card is about to become the topmost card.

## Index Tab Functions

The index tab has the following two functions:

#### Tab Entry Function

Enter the name of the function to be called when the index tab gains focus.

#### Tab Exit Function

Enter the name of the function to be called when the index tab loses focus.

## Order of Events

The order of execution for the above entry functions is as follows:

1. Expose Event
2. Card Entry Event

### 3. Tab Entry Event

Not all events may happen at the same time. For example, focus can go directly onto a widget on the tab card—bypassing the tab index—if a widget on the card receives focus. In that case, the tab entry event does not occur. Similarly, if the card was exposed as part of an earlier event (at screen entry or when it was made topmost), the expose event does not occur when focus enters the tab card.

A situation when all three entry events occur is when the screen is being opened and the tab deck is the first widget to get focus.

Exit events occur in the corresponding opposite order.

## Validation

### Validation Function

Specify the name of a function to be called when the card is validated. This validation function will be triggered when the tab card is validated via a call to `sm_validate`, or when the entire screen is validated via a call to `sm_s_val`. In either case, the tab card validation will be triggered after all the individual widgets on the respective tab card are validated.

To validate all the tab cards in a deck, call `sm_validate` on the tab deck and this will iteratively call validation functions on all the tab cards in the deck, whether they are hidden or not.

**Note:** Do not call `sm_validate` on the tab card as part of the tab card's validation function, since this will produce an infinite loop.

## Format/Display

### Active Pixmap

Attach a pixmap file to the index tab of a tab card.

**Note:** Only Motif 2.2 supports having both text *and* an image on a tab. Earlier versions can only have text *or* an image.

# 17 Framesets and Splitters

A splitter divides a window into a number of subareas, each of which is populated separately. Each subarea is called a "pane". The movable dividing line between two panes is called a "mullion".

In Panther Windows and Motif applications, support for splitters is implemented by means of a special kind of screen called a frameset. The number and configuration of the panes into which each frameset is divided is determined in the Panther editor by placing special widgets called splitters on the frameset. Each pane is itself a Panther widget that has properties associated with it. The most important property of a pane is the name of a screen that will be rendered inside the pane at runtime. The screens associated with a frameset's panes are not displayed along with the splitters in the Panther editor. They must be created and edited separately.

---

## Creating And Editing Framesets

---

In the Panther editor a new frameset is created by choosing File→New→Frameset on the editor's main menu. This will bring up a blank frameset. Such a frameset is much like an ordinary Panther screen, in that it supports a similar set of properties, is stored in libraries, and is generally worked with in much the same way. But a frameset can hold only two kinds of widgets: splitters and panes.

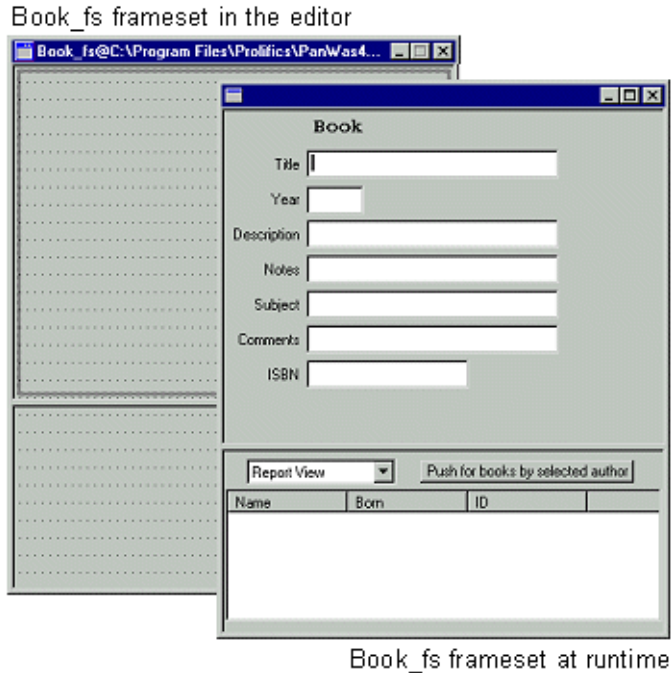




Figure 17-1 Book\_fs from the frameset sample application

## Splitter Widgets

A frameset is divided into multiple panes by placing splitter widgets on it. There are three kinds of splitter widgets – vertical splitters, horizontal splitters and two-way splitters.

Vertical splitter 

Divides the frameset into two or columns, and is represented in the frameset screen in the editor by one or more vertical mullions.

Horizontal splitter 

Divides the frameset into two or more rows, and will be represented in the editor by one or more horizontal mullions.

Two-way splitter 

Divides the frameset into four panes, and will be represented in the editor by one vertical and one horizontal mullion.

A splitter can be configured to divide the area it was placed in into further rows or columns by changing the splitter's Number of Rows or Number of Columns property.

Placing a splitter in an empty frameset divides the whole frameset into panes. Subsequent splitter placements in that frameset will necessarily have to be placed within previously defined panes. Such splitter placements will divide the panes the splitter was placed in into smaller panes. That is to say, splitters can be nested.

Splitters are selected in the editor by clicking on one of the mullions that represents the splitter. A splitter can also be selected from the [Widget List](#).

When a splitter widget is selected in the editor, the properties window is updated to display the properties settable for splitter widgets.

The following are the properties for splitter widgets:

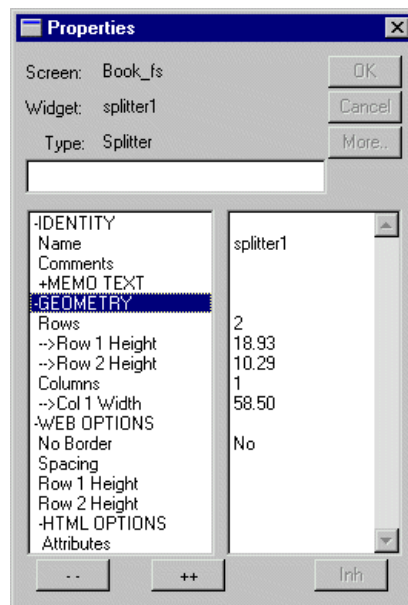


Figure 17-2 Properties Window for Splitter Widgets

The Rows and Columns properties determine how many panes each splitter defines. These can have integer values from 2 to 16. Changing this property will create or destroy panes associated with the splitter. Newly created panes will be added to the right or below the existing panes. Likewise, panes destroyed by changes to this property will be removed from the right hand side and from the bottom of the screen.

The height and width properties, for rows and columns respectively, will be used at runtime to render the frameset. These properties can be set in the editor by dragging one of the mullions associated with a splitter, or it can be entered in the properties window. The size properties are array-valued, with one occurrence for each row or column. Note, however, that the height of the bottommost row, and the width of the rightmost column, cannot be changed. These are computed by the sizes of the other rows or columns, and the height of the container.

Like the other size-related properties in Panther, the height and width properties for splitter rows and columns can be specified using a variety of units. The default is grid units. In many cases pixels will be the best unit to use.

The Web Options and HTML Options are relevant to web deployment of framesets. Refer to [page 17-14](#), “Web Deployment of Framesets.”

## Pane Widgets

Unlike other widgets, pane widgets are not explicitly created by Panther users. Pane widgets are implicitly created when splitter widgets are placed on a Screen.

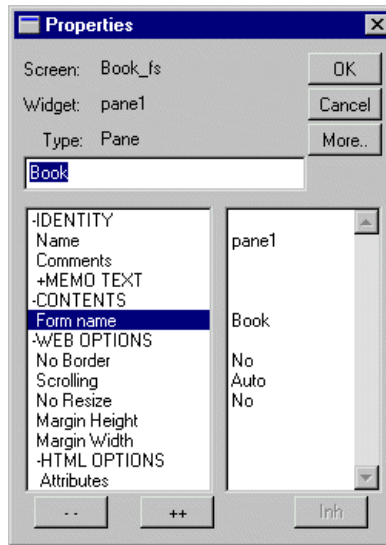
When a vertical or horizontal splitter is placed on an empty frameset, two panes are created. When another splitter is placed in a previously defined pane, new panes are added. Previously defined panes are not destroyed. Panes are added to the top and left, shifting existing panes to the right and down.

A pane widget is selected in the editor by clicking on a frameset screen that contains at least one splitter widget. You can also use the [Widget List](#) to select a pane.

Note that if a frameset contains at least one splitter, it might not be clear how to select the frameset itself. Clicking on one of the mullions between two panes will select a splitter. Clicking on the background of the frameset will select one of the panes. The frameset will be selected when it has focus, and none of the widgets on the frameset screen is selected. A quick way to do that is to click on a pane, and then shift-click on

the same pane again. The first click selects the pane, and deselects everything else. The shift-click deselects the pane, leaving the frameset itself selected. You can also select the frameset using the [Widget List](#).

Pane widgets display the following properties in the properties window when selected:



**Figure 17-3 Properties Window for Pane Widgets**

The Form Name property is used to designate which screen to render inside the pane at runtime. This can be the name of any Panther screen, or the name of another frameset. It is possible to nest framesets within one another.

The Web Options and HTML Options are relevant to web deployment of framesets. Refer to [page 17-14](#), “Web Deployment of Framesets.”

## Frameset Properties

A frameset supports a set of properties much like that of an ordinary screen, and these properties are used in the same way as the corresponding properties of screens. The properties under the Web Options heading and the HTML Options and Browser Options headings are relevant to web deployment of framesets.

Many of the properties of screens are not relevant to framesets. The corresponding properties of the screens displayed in the frameset's panes would be the ones relevant at runtime.

The Wallpaper Pixmap and Color properties deserves special mention. The specified wallpaper or color will only be seen in an empty pane, that is, a pane that has no screen associated with it. A pane that has a screen associated with it will display the background color or wallpaper of that screen.

After the frameset contains splitters, you can view the frameset's properties by selecting the frameset from the [Widget List](#) or by deselecting the current pane/splitter using Ctrl+Click.

---

## Programming With Framesets

---

At runtime, the frameset and the screens displayed in its panes all act as sibling windows. Focus can move from window to window within a frameset, and whatever other windows might be open, as among sibling windows in a Panther application that doesn't use framesets. The frameset itself cannot normally get focus after it has been opened.

### Opening a Frameset

A frameset is opened like any other screen. When a frameset is opened, it goes through the usual set of screen entry operations (unnamed JPL, screen entry function, etc.) Note, however, that all of these operations take place before the screens designated to be opened in the frameset's panes have been opened. This means that processing associated with the frameset's entry procedures cannot reference fields on the screens that will be opened in the frameset's panes.

After the frameset's entry procedures, the screens designated to be opened in the frameset's panes will be opened. They will be opened from bottom to top, and from right to left. So the pane in the lowest and rightmost position will be populated first, and the pane in the topmost and leftmost position will be populated last. Each screen



opened in a pane will itself go through the normal screen entry events. The screens are opened in the reverse or Panther's usual left-to-right, top-to-bottom order so that the screen in the upper left corner will be the last screen opened and will be the one to retain focus. In this manner none of the screens placed in the panes undergo two entry events while opening the frameset.

Once all of the frameset's panes have been populated by their designated screens, the screen in the topmost and leftmost pane will gain focus. Focus will go to the first available field in that screen. Only at this point will the keyboard stack be opened. Any logical keys ungoten by any of the screen entry procedures, including that of the frameset itself will be read by the first screen to get focus.

## Cursor Movement and Window Management using Framesets

If a frameset is the only window open, cursor movement will behave much as though the screens in each of the panes were in separate sibling windows. Tabbing order in each pane will be governed by the usual rules. If you want a TAB from a field in one pane to lead to a field in another pane, you will have to write a routine that calls `sm_wselect`, and then perhaps `sm_gofield` to put focus where you want it.

When focus leaves one pane, the screen in that pane undergoes a screen exit event, with the `K_HIDE` flag set. Likewise when focus enters a pane, the screen in that pane undergoes a screen entry event with the `K_EXPOSE` flag set.

If a frameset is not the only screen open, things get more complicated. To understand how things work, it's best to think in terms of the window stack, and to remember that the frameset itself is a window on the stack, as are the screens in the frameset's panes.

When a frameset is first opened it is the active window while it opens the screens that are to populate its panes. Imagining a simple scenario of opening a frameset with three panes, the sequence of entry and exit events would be as follows:

- frameset enter (`K_ENTRY`)
- frameset exit (`K_EXIT` | `K_EXPOSE`)
- pane3 enter (`K_ENTRY`)
- pane3 exit (`K_EXIT` | `K_EXPOSE`)
- pane2 enter (`K_ENTRY`)

- pane2 exit (K\_EXIT | K\_EXPOSE)
- pane1 (K\_ENTRY)

at this point the window stack would look like:

```
pane1 (active)
pane2
pane3
frameset
```

After its initial entry the frameset itself will not, under normal circumstances, get focus, or undergo entry or exit events. Thus the frameset will tend to linger at the bottom of the window stack.

If the user clicks a button on the first pane that opens another window, a sibling to the frameset itself, the following entry and exit events would occur:

- pane1 exit (K\_EXIT | K\_EXPOSE)
- otherwindow enter (K\_ENTRY)

at which point the window stack would then look like this:

```
otherwindow (active)
pane1
pane2
pane3
frameset
```

If a user now clicks on pane3 in the frameset, the following entry and exit events would occur:

- otherwindow exit (K\_EXIT | K\_EXPOSE)
- pane3 entry (K\_ENTRY)

and the window stack would end up looking like:

```
pane3 (active)
otherwindow
pane1
pane2
frameset
```

Other much more complicated scenarios could be envisioned. Things would get very complicated in the case where framesets are nested. In general, most people will probably not want to do anything on screen hide events (`K_EXIT` | `K_EXPOSE`) for screens that are in framesets.

## Closing Framesets

A screen in Panther is typically closed in one of three ways:

- By the user clicking on the 'x' control on the screen's title bar
- Programmatically, with the function `sm_jclose`
- By means of the logical key EXIT

A frameset is closed in the same manner.

Clicking on the 'x' control on the title bar will cause the logical key EXIT to be issued, and so the first and third of these methods will be same.

When a screen gets the logical key EXIT from the key stack, it calls `sm_jclose`. So really all three of these methods are fundamentally the same.

The function `sm_jclose` is one of those that implicitly operates on the screen that currently has focus. If the screen that currently has focus is in a frameset, then a call to `sm_jclose` will close the whole frameset.

Closing a screen that is in a frameset will cause the whole frameset to be closed. The screens in the frameset will all be closed, in the order in which they are found in the window stack. This list of screens includes the screens in the panes of any other framesets that are nested in the frameset being closed. After all the screens in all of the panes of the framesets have been closed, the framesets themselves will be closed. The outermost frameset will always be the last to close.

As an example, consider the following scenario. The window stack looks like this:

```
panel (active)
pane2
otherwindow
pane3.1
pane3.2
pane3.3
pane3 (nested frameset)
frameset
```

(This means to indicate that pane3 in the frameset is itself a frameset, containing three panes. The panes in that nested frameset are the ones designated 3.1, 3.2, and 3.3.)

If a call to `sm_jclose` is now issued, the window stack is reordered to place all the panes contained in the frameset (including those in the panes of the nested frameset) on the top, followed by all the framesets.

```
pane1 (active)
pane2
pane3.1
pane3.2
pane3.3
pane3 (nested frameset)
frameset
otherwindow
```

And the screens will be closed in order, from the top down. Until the outermost frameset that the screen that had been active when `sm_jclose` was called is closed.

When closing the screens contained in the frameset, normal screen events would occur, just as though a series of sibling windows were being closed one after another. In this case, the event sequence would be:

- pane1 exit (K\_EXIT)
- pane2 enter (K\_ENTRY | K\_EXPOSE)
- pane2 exit (K\_EXIT)
- pane3.1 enter (K\_ENTRY | K\_EXPOSE)
- pane3.1 exit (K\_EXIT)
- pane3.2 enter (K\_ENTRY | K\_EXPOSE)
- pane3.2 exit (K\_EXIT)
- pane3.3 enter (K\_ENTRY | K\_EXPOSE)
- pane3 exit (K\_EXIT)
- pane3 enter (K\_ENTER | K\_EXPOSE)
- pane3 exit (K\_EXIT)
- frameset enter (K\_ENTRY | K\_EXPOSE)
- frameset exit (K\_EXIT)

- `otherwindow enter (K_ENTRY | K_EXPOSE)`

Which would leave the window stack as:

```
otherwindow (active)
```

The reordering of the window stack prior to closing the screens is not accompanied by screen events, except in the unusual circumstance that `sm_jclose` is issued when a frameset has focus. Consider a scenario where the window stack looks like:

```
frameset
panel
otherwindow
pane2
pane3
```

And `sm_jclose` is called, the window stack will be reordered to place all the screens in the frameset at the top, followed by all the framesets contained in the outermost frameset. In this case, the following events would occur:

- `frameset exit (K_EXIT | K_EXPOSE)`
- `panel enter (K_ENTRY | K_EXPOSE)`

The rest of the reordering, to place the frameset after all the screens in its panes, occurs without any screen events taking place, and will leave the window stack as:

```
panel (active)
pane2
pane3
frameset
otherwindow
```

And the screens will close, with `CLOSE` and `EXPOSE` events occurring as described in the previous scenario.

Note that this latter scenario is very unusual. Under normal circumstances a frameset will only gain focus when it is first opened, and will not be in focus when user-written code is being executed. The only way to force a frameset to gain and retain focus is by explicitly giving it focus with a call to `sm_wselect` or `sm_n_wselect`.

---

# Runtime Properties

---

## Screen and Frameset Properties

Screens and framesets have a property called `PR_FRAMESET` that returns a handle to the frameset the screen or frameset is inside, just as the `PR_GRID` property returns a handle to the grid a particular field is in. If a screen or frameset is not in a frameset `PR_FRAMESET` returns "".

Additionally, screens and forms have a `PR_SPLITTER` property that returns a handle to the splitter that defines the pane in which the form is contained. If the screen is not in a frameset, this property will return "".

Screens and forms also have a `PR_PANE` property that returns a handle to the pane that contains the form. If the screen is not in a frameset, this property will return "".

## Splitter Properties

A splitter will return `PV_SPLITTER` when `PR_WIDGET_TYPE` is queried.

The `PR_SPLITTER_ROWS` and `PR_SPLITTER_COLS` properties of splitter widgets are read-only at runtime. Since these properties are read-only, you cannot change a frameset's pane configuration on the fly. To change the number of subwindows in a frameset, you will have to nest framesets.

`PR_SPLITTER_ROWS` returns the number of rows in the splitter. This will be an integer from 1 to 16, but it cannot be 1 if the number of columns is 1.

`PR_SPLITTER_COLS` returns the number of columns in the splitter. This will be an integer from 1 to 16, but it cannot be 1 if the number of rows is 1.

The `PR_SPLITTER_ROW_HEIGHT` property gets or sets the height of each row. This is a multi-occurrence property, indexed from 1 to `PR_SPLITTER_ROWS`. The height of the bottommost row cannot be set. It is computed based on the sum of the other row heights and the size of the container.

The `PR_SPLITTER_COL_WIDTH` property gets or sets the width of each column. This is a multi-occurrence property, indexed from 1 to `PR_SPLITTER_COLS`. The width of the rightmost column cannot be set. It is computed based on the sum of the other column widths and the size of the container.

The `PR_MEMBER` property is an indexed property that will return handles for the panes in the splitter. The pane in the first row, first column will have the index 1. The pane in the first row, second column will have index 2. The pane in the second row, first column will have index `PR_SPLITTER_COLS + 1`. In general the formula to determine a pane's index is:

$$\text{index} = (\text{row} - 1) + \text{PR\_SPLITTER\_COLS} + \text{column}$$

`PR_FRAMESET` will return a handle to the frameset that the splitter is contained in.

If a splitter is contained in a pane defined by another splitter, the property `PR_SPLITTER` will return a handle to that other splitter and the property `PR_PANE` will return a handle to the pane in which the splitter is contained.

## Pane Properties

The `PR_PANE_FORM` property of pane widgets can be changed at runtime. Changing the screen associated with a pane at runtime should close the screen currently in that pane, and then open the newly specified screen in its place. Clearing the `PR_PANE_FORM` property of a pane will leave the pane empty. Since issuing `sm_jclose` will close the whole frameset, clearing the `PR_PANE_FORM` property of a pane is the only way to close a single screen in a frameset.

The `PR_PANE_ROW` property gets the row number of the pane.

The `PR_PANE_COL` property gets the column number of the pane.

The `PR_PANE_INDEX` property gets the index of this pane. The indices are the same as those used by the `PR_MEMBER` property of framesets. The JPL code

```
index = mypane->pane_index
```

is equivalent to:

```
index = (mypane->pane_row - 1) + \
        (@widget(mypane->splitter)->splitter_cols + \
        mypane->pane_col
```

The `PR_MEMBER` property of a pane will return an object id for the pane's contents. This will either be the id of a form (a screen or another frameset), or a splitter, or nothing.

The `PR_FRAMESET` property returns an object id for the frameset in which the pane is contained.

The `PR_SPLITTER` property returns an object id for the splitter that contains the pane.

---

## Web Deployment of Framesets

---

If a frameset is brought up in a jserver, HTML to represent the frameset will be generated and sent to the browser.

The properties in the Web Properties sections of the Properties window control various options relevant to HTML generation of framesets.

### Frameset Web Properties

Framesets support the `NOFRAMES` Markup property. The text entered here will be returned to the browser in the event that the frameset is accessed by a browser that does not support frames. The value of this property is indexed (like the JavaScript and VBScript properties). If no value is provided for the `NOFRAMES` Markup property, the following text will be returned to browsers that do not support frames:

`This Page Requires Frames.`

The `NOFRAMES` Markup property is accessible at runtime, its C constant name is `PR_FRAME_NOFRAMES` and its JPL mnemonic is `frame_noframes`.

### Splitter Web Properties

Splitter Widgets display the following properties under the Web Options heading in the Properties window:



```
No Border
Spacing
Row 1 Height
Row 2 Height
...
Col 1 Width
Col 2 Width
...
```

The No Border property defaults to No. Hence by defaults borders will be shown. If No Border is set to Yes the HTML output is `FRAMEBORDER="0"`. The No Border property is referred to at runtime as `PR_FRAME_NOBORDER` in C and as `frame_noborder` in JPL.

Spacing is the width of the mullions between panes. Setting Spacing = 0 has the same effect as setting No Border = Yes. The Spacing property is referred to at runtime as `PR_FRAME_SPACING` in C and as `frame_spacing` in JPL.

The height and width properties in the Web Options section can be used to override the similar properties in the Geometry section. The latter are used for GUI display, and are used for HTML generation as well, unless height and width properties are explicitly specified in the Web Options section.

There will be one height shown for each row, and one width for each column. If there is only one row, then no heights will be shown, and if there is only one column, then no widths will be shown.

The Web heights and widths can be either given as percentages or in pixels. They also accept the special value \*. Omitted entries default to \*, which tells the browser to choose whatever size it wants. Typically it will divide the available space equally among the regions specified with stars.

For example:

```
Row 1 Height = 25%
Row 2 Height =
Row 3 Height = 25%
Row 4 Height = *
```

will be output as `ROWS="25% , * , 25% , *"`. The browser will allocate half the available space to the two specified rows, and divide the remaining space equally among the rows specified with stars. In this case, that would result in four rows of equal size.

Heights and widths should either be given in percentages or in pixels. Do not mix units.

## Pane Web Properties

Like splitters, panes each have the No Border property. As noted above, it defaults to No. If set to Yes the 3-D border around the pane may be removed (or it may not be, this is browser-dependent behavior). Setting No Border to Yes for some panes and No for other panes can yield strange and unsightly results. It is recommended that you use the splitter-level No Border property if possible.

The Scrolling property supports three values: Auto, No and Yes. It defaults to Auto. If set to Auto, scroll bars will appear only when needed. If set to No, scroll bars will never appear and if set to Yes scroll bars will always appear. The Scrolling property is referred to at runtime as `PR_FRAME_SCROLLING` in C and as `frame_scrolling` in JPL.

The Margin Height and Margin Width properties control the space between the splitter lines and the HTML inside the pane. It defaults to 13 or 14 pixels. However, if you set one and not the other, the unspecified one will default to 0. These are referred to at runtime as `PR_FRAME_MARGIN_HEIGHT` and `PR_FRAME_MARGIN_WIDTH` in C and as `frame_margin_height` and `frame_margin_width` in JPL.

---

# The Frameset Sample Application

---

In the directory `$SMBASE\samples\frameset` there are three files: a `.lib` file, a database file and a `README` file. These constitute the sample application provided to illustrate the use of framesets. This application uses ODBC to access its database. To run the frameset sample application you must have the ODBC database driver installed.

This application also provides examples of code to manipulate a few common ActiveX controls: the `TreeView` control, the `ListView` control and the `ImageList` control. The code for all the application's functionality is in the screens that are found in the library `Biblio.lib`. The JPL that manipulates the ActiveX controls has been commented and written in such a way that it can easily be re-used.

See the `$SMBASE\samples\frameset\README` file for more information, regarding installing and running the frameset sample.

# 18 ActiveX Controls

In Panther, you can create ActiveX control containers in your application screens to be deployed in your Web or Windows applications. ActiveX is part of Microsoft's COM (Component Object Model) technologies, which were derived from the OLE and OLE 2 specifications. COM defines a standard for application object interaction by specifying a set of common interfaces. An interface, which is a set of related functions or methods, is not allowed to change after it is published, thereby ensuring that any components referencing the interface will always find the same functionality. Since COM is a binary standard, components written in different programming languages can work together in the same application.

ActiveX controls are software components using COM technologies to interact with other COM objects and services. Being only a software component, not an application, an ActiveX control must be placed in a control container in order to operate.

The author of an ActiveX control typically uses a programming language, such as Visual C++, to write the control and define the following:

- **Properties**—Each control has a series of properties controlling the data and appearance of the control. In Panther, you can set the initial values of the properties in the editor and change those values at runtime.
- **Interfaces**—ActiveX controls, like all COM objects, can support one or more interfaces. An ActiveX control must support the `IUnknown` interface whose `QueryInterface` method can be used to test support for other interfaces.
- **Methods**—In addition to the interface's methods, custom methods (or functions) can be written for the control. These methods define the control's actions.
- **Events**—The control author defines the events applicable to the control, such as mouse clicks or key presses. At runtime, the control notifies its container when an event occurs.

- CLSID (Class Identifier)—Each ActiveX control is assigned a globally unique identifier when it is created.

To qualify as an ActiveX control, a control needs support for the IUnknown interface and the ability to self-register—to create an entry in the system registry when requested to do so.

Since ActiveX controls, unlike Java applets, have the ability to write and modify data on your system, Microsoft developed the Authenticode technology to digitally sign ActiveX controls. This digital signature identifies the software company that manufactured the control and checks to see if the control was tampered with since it was signed. However, a digital signature does not guarantee that an ActiveX control is safe; it merely identifies the manufacturer. If you are developing Internet applications, you should digitally sign any ActiveX controls that you write and distribute.

---

## **Embedding ActiveX Controls in Application Screens**

---

The ActiveX option is available on the toolbar and on the editor Create menu, in the Extended Widgets section. This option creates an ActiveX control container on your application screen; you must then specify the ActiveX control that will occupy the container.

First, you must obtain the ActiveX control you want to place in your application. You can write the control yourself, use a distributable control, or obtain a development license from a control vendor.

### **How to Embed an ActiveX Control**

1. From the Create menu or toolbar, choose ActiveX and place the container on the screen.

2. If the control is registered on your system, under ActiveX in the Properties window, select the Control Name from the option menu. This automatically fills the CLSID property.
3. If the control is not registered on your system, under ActiveX in the Properties window, enter the CLSID for the control in the CLSID property.
4. Expand the Control Properties category and specify values for any of the control's properties.
5. If the control is not registered on your system, under Geometry, set the Height and Width of the control container.
6. (Web applications only) Under Web Options in the Codebase property, enter the path to the control's file on the Web application server.
7. Specify additional container properties, such as Background Color.
8. (optional) Write any JPL procedures, C functions, JavaScript or VBScript functions to access the control's properties, methods, or events. JavaScript and VBScript code only work in Web applications.

**Note:** You cannot use the transaction manager to access an ActiveX control.

## Setting ActiveX Properties

Under ActiveX, there are properties identifying the ActiveX control in addition to properties of the ActiveX control itself. Once you select an ActiveX control that is registered on your system, the properties belonging to the ActiveX control will be displayed in the Properties window.

Even though the Control Name property is used to select an ActiveX control that is registered on your system, the CLSID, which stands for class identifier, is globally unique and is used to identify the ActiveX control.

## How to Select an ActiveX Control Registered on your Workstation

Under ActiveX, select the Control Name from the option menu. The CLSID property is automatically filled.

## How to Use an ActiveX Control Unavailable on your Workstation

Under ActiveX, enter the CLSID for the ActiveX control. The CLSID is in the following format, with # being any hexadecimal digit (0-9, A-F):

```
#####-####-####-####-#####
```

After you specify the ActiveX control, you can specify the properties belonging to the ActiveX control.

## How to Set Properties of the ActiveX Control

Expand the Control Properties category. If the ActiveX control is registered on your system, all settable properties and their initial values are displayed.

## How to Set Unlisted ActiveX Control Properties

Under Control Properties, select Other Properties which opens a window where you can enter properties and their values. Each property is on a separate line, in a *name=value* format. For example:

```
BlankIfZero = 1
```

The author's documentation of the ActiveX control should inform you of all properties, events, and methods. Panther also provides the ActiveX Viewer which displays all the ActiveX controls that are registered on your system and their properties, events, and methods, including the read-only properties. For more information, refer to page 18-13, "Using AxView, the COM Control Viewer."

For information on properties, refer to [page 18-6](#), "Setting Properties at Runtime."

## Setting Runtime Licensing

Some ActiveX controls require valid runtime licenses. When these controls are created, they check for a runtime license to verify that they are licensed to run on the current computer. If a runtime license is not supplied, these controls check the registry for a license.

Panther supports runtime licensing for ActiveX controls. At runtime, if the Runtime License property exists and the control supports runtime licensing, the control will be created using the license.

Some ActiveX controls return their licenses when the Control Name or the CLSID is provided. If the runtime license is not returned automatically, and the control requires runtime licensing, you can enter the value.

## How to Enter the Runtime License

Under ActiveX properties, in Runtime License, enter the value of the license.

## Setting Color Properties

Colors are implemented differently for each ActiveX control. Some controls use the color of the control container; some controls have their own color properties, such as `BackColor`.

In Panther, you can set the color of the control container in the Color properties. If a control is set to use ambient properties, which are the properties of the control container, the color will be reflected in the ActiveX control.

For any of the ActiveX color properties needing an `OLE_COLOR` value, you can enter a hex or decimal value or select a color from the color selection box, available with the More button. The color selection is converted to an RGB format, with slashes separating the amount of each color. The following formats can be entered in the Properties window:

- RGB values, `rr/gg/bb`
- Web hex values, `#rrggbb`
- OLE hex values, `&H00bbggrr` or `&Hbbggrr`
- OLE decimal values, which are derived from the OLE hex values
- System color values, `&H800000ii`

**Note:** System colors do not get converted to RGB values in the Properties window.

---

# Interacting with ActiveX Controls

---

With ActiveX controls, you have the abilities to set property values, to call methods specified for the control, and to program an event handler for the control.

## Setting Properties at Runtime

Depending on the deployed platform, properties of an ActiveX control can be viewed or changed at runtime using JPL, C, or Java.

For Windows applications, use the JPL property syntax or the C functions `sm_obj_get_property` and `sm_obj_set_property`. For Web applications, use the JPL property syntax, call the C functions `sm_obj_get_property` and `sm_obj_set_property`, or set the properties on the browser-side using JavaScript or VBScript.

To use the JPL property syntax with ActiveX controls, you must prepend `ax_` to the property name. This ensures that there are no conflicts between Panther properties and ActiveX control properties. For example, the following JPL statement uses the JPL property syntax to set the value property of the `PrlSpinner` Control:

```
spinner->ax_Value = 55
```

You could also use `sm_obj_set_property` in a C function or JPL procedure to set the property value, as the following JPL statement illustrates:

```
call sm_obj_set_property (spinner->id, "Value", "55")
```

Properties can specify the appearance as well as the data in an ActiveX control. The sample calendar control has a `ShowDateSelectors` property to display and hide the month and year option menus at the top of the control, which is set in the following JPL procedure:

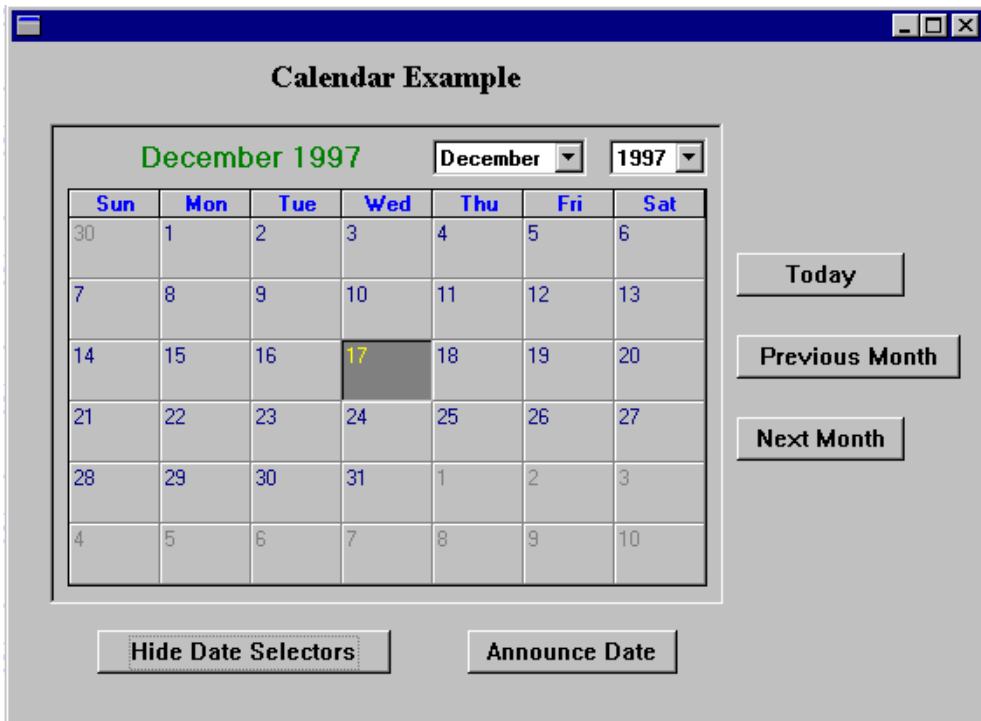
```
proc show_date_sel
{
  if calendar->ax_ShowDateSelectors == 0
  {
    calendar->ax_ShowDateSelectors = 1
  }
}
```



```

        toggler = "Hide Date Selectors"
    }
    else
    {
        calendar->ax_ShowDateSelectors = 0
        toggler = "Show Date Selectors"
    }
    return
}

```



**Figure 18-1** In the calendar control, the month and year option menus and the label on the push button are controlled at runtime with a JPL procedure.

## Transferring Data to the ActiveX Control

In Web applications, data can be transferred to the ActiveX control using JPL procedures, C functions, JavaScript, or VBScript to set the appropriate properties. For information on setting property values using JPL or C functions, refer to [page 18-6](#), “Setting Properties at Runtime.”

**Note:** In this release, the transaction manager cannot be used to access an ActiveX control.

For JavaScript and VBScript, the ActiveX control is accessed as `document`.

- `form_name.control_name.form_name` is specified at screen level by setting the `NAME` attribute in the Web Options→Custom HTML→Form Attributes property.
- `control_name` is specified for the ActiveX control container in the Identity→Name property.

In the following examples, the `NAME` attribute for `form_name` is specified as `main`.

## Data Transfer Using JavaScript

The following JavaScript `onLoad` function, called when the screen is loaded into the Web browser, sends data from the hidden Panther variables to each instance of the PrlSpinner Control. This function is included in the screen's JavaScript property and specified for the `OnLoad` browser event.

```
function onLoad()  
{  
    document.main.PrlSpinner1.value =  
        parseInt(document.main.i_1_len1.value);  
    document.main.PrlSpinner2.value =  
        parseInt(document.main.i_1_len2.value);  
    document.main.PrlSpinner3.value =  
        parseInt(document.main.i_1_len3.value);  
}
```

## Data Transfer Using VBScript

Here is the same function in VBScript. In VBScript, the function named `window_unLoad` is automatically called on screen entry so you do not have to specify the function name in the `OnLoad` property.

```
Sub window_onLoad
    document.main.i_1_Pr1Spinner1.value =
        document.main.i_1_len1.value
    document.main.i_1_Pr1Spinner2.value =
        document.main.i_1_len2.value
    document.main.i_1_Pr1Spinner3.value =
        document.main.i_1_len3.value
End Sub
```

## Submitting Data to the Web Application Server

In Web applications, any data submitted to the Web application server must be in an input field. Since ActiveX controls have no input fields, data must be transferred from the ActiveX control to Panther input variables using JavaScript or VBScript before the screen is submitted back to the Web application server. For more information, refer to “Submitting Data to the Web Application Server” [on page 8-26](#) in *Web Development Guide*.

## Calling ActiveX Methods

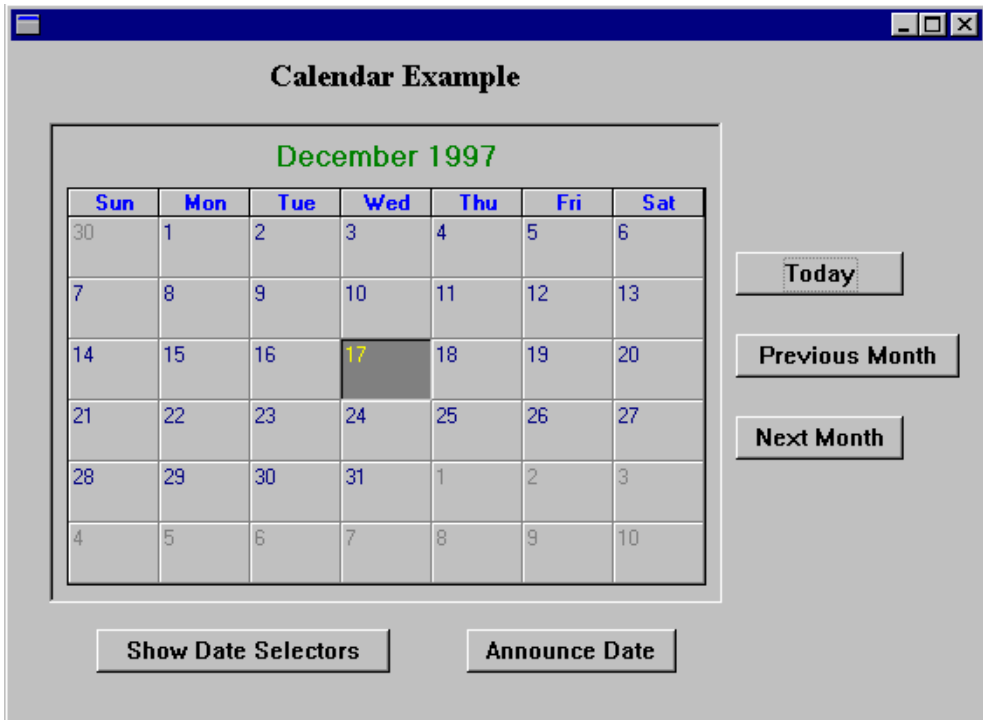
The author of an ActiveX control specifies what methods can be used to access the control and the arguments needed for those methods. Those methods can then be called in Windows applications using `sm_obj_call` and in Web applications using VBScript or JavaScript.

In the Windows application containing the calendar control, the following JPL procedures use `sm_obj_call` to call the control's methods:

```
proc goto_today
call sm_obj_call(calendar->id, "Today")
return

proc goto_prev_month
call sm_obj_call(calendar->id, "PreviousMonth")
return
```

```
proc goto_next_month
call sm_obj_call(calendar->id, "NextMonth")
return
```



**Figure 18-2** The methods for the ActiveX control can be called through a JPL, C, and Java in Windows applications and through JavaScript or VBScript in Web applications.

## Calling Methods Using VBScript

The following example specifies the first level of a treeview control by using VBScript to call the control's methods. Again, the control is accessed as document.

`form_name.control_name`, and the NAME attribute in the Form Attributes property is specified as `main`.

```
Sub BuildTree
Dim parent
Dim child
```

```
Rem Set up attributes for the TreeView control.

Call document.main.i_1_TreeView.SetStyle
    (false, true, true, true)

Rem Associate a pixmap with the treeview. This is a .bmp
Rem file containing two images: one for root nodes and one
Rem for child nodes.

Call document.main.i_1_TreeView.BitmapURL = "pixmap_URL"

Rem Add first root node.

parent = document.main.i_1_TreeView.InsertNode
    (0 "Parent1", "Parent1", 1,1)

Rem Add children for the first root.
Rem Note the use of the return value from the previous
Rem statement to supply the object ID needed for this
Rem method invocation.

child = document.main.i_1_TreeView.InsertNode
    (parent, "Child1", "Parent1.Child1", 2,2)
child = document.main.i_1_TreeView.InsertNode
    (parent, "Child2", "Parent1.Child2", 2,2)

End Sub
```

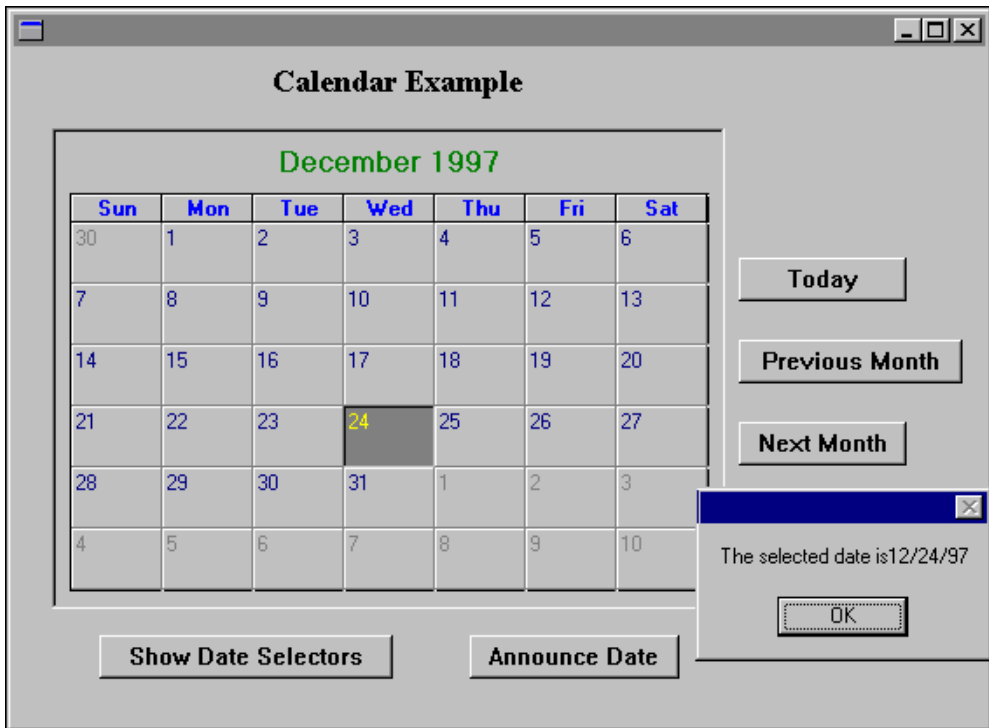
## Specifying an ActiveX Event Handler

The author of an ActiveX control specifies what events are applicable for the control. An event handler can then be written for those events using [sm\\_com\\_set\\_handler](#) in Windows applications and VBScript functions in Web applications.

In the Windows calendar application, the screen entry JPL procedure specifies an event handler for the Db1Click event. When a user double clicks on any part of the ActiveX control, a message listing the selected date is displayed. The screen entry procedure and the JPL procedure called on the Db1Click event are as follows:

```
proc onentry
call sm_com_set_handler \
    (calendar->id, "Db1Click", "get_value")
return
```

```
proc get_value  
vars date = sm_obj_get_property(calendar->id, "Value")  
msg msg "The selected date is" date  
return
```



**Figure 18-3** An event handler for the ActiveX control can be called through a C function in Windows applications and through JavaScript or VBScript in Web applications.

## Using VBScript for Event Handling

The following VBScript function is hooked to the `URLSelected` event for the treeview control. It keeps the Panther input field `i_1_selection` (which would typically be hidden) updated with a value corresponding to the node currently selected in the treeview control. When this screen is posted back to the Web application server, Panther can determine which node the user selected.

```

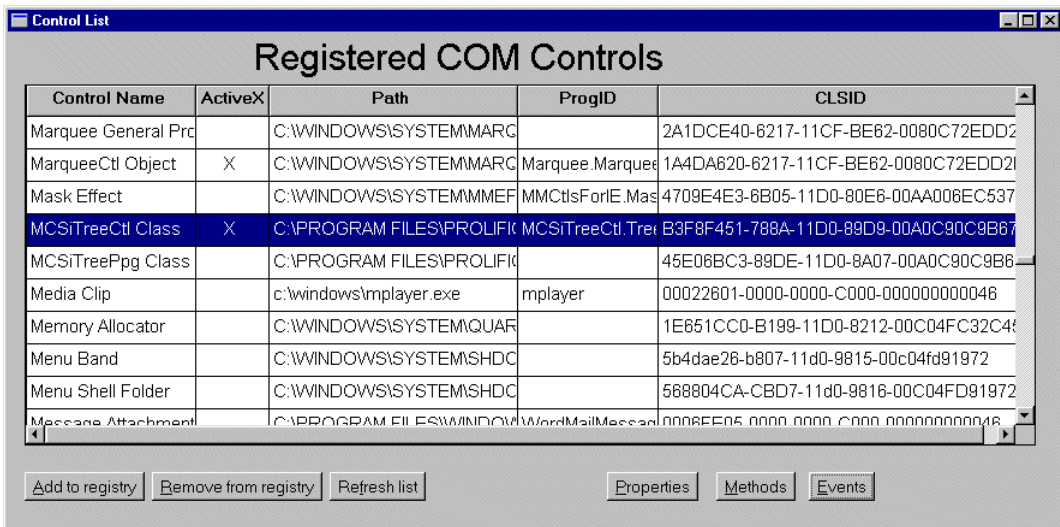
Sub i_1_TreeView_URLSelected (node)
    document.main.i_1_selection.value = node
End Sub

```

Refer to the *Panther Gallery* for additional JavaScript and VBScript samples.

## Using AxView, the COM Control Viewer

AxView, which is available for Windows development, displays the COM controls (which includes ActiveX controls) that are registered on your system. The controls are listed alphabetically by Control Name along with the path location, ProgID (programmatic identifier), and CLSID (class identifier) for each control. ActiveX controls are also identified.

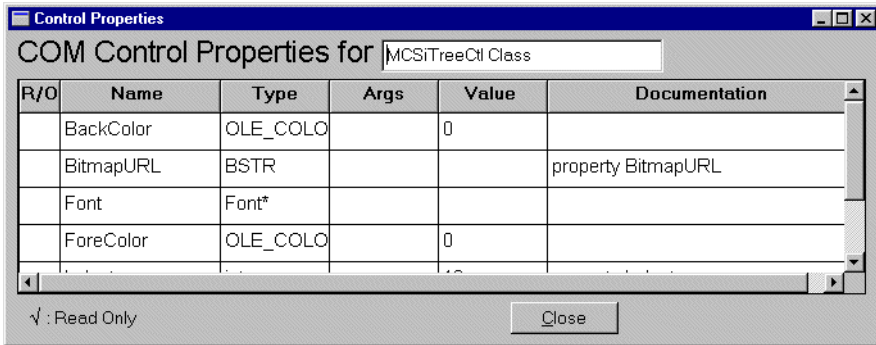


**Figure 18-4** The main window lists the COM controls in the system registry as well as the path location, ProgID—if available, and CLSID for each control.

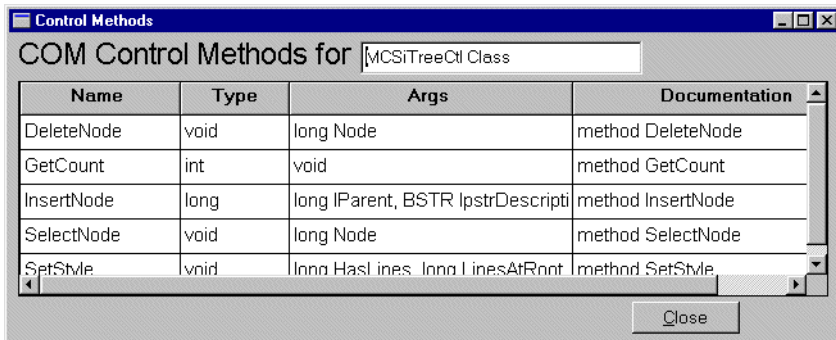
From the main window, you have the following options:

- Add to registry—Add a COM control to the system registry.
- Remove from registry—Delete a COM control from the system registry.
- Properties—View the control's properties and the type of value required for that property.
- Methods—View the methods specified for the control and the arguments needed to call the method.
- Events—View the events specified for the control.

For additional information about a control, refer to the documentation provided by the author of the COM control.

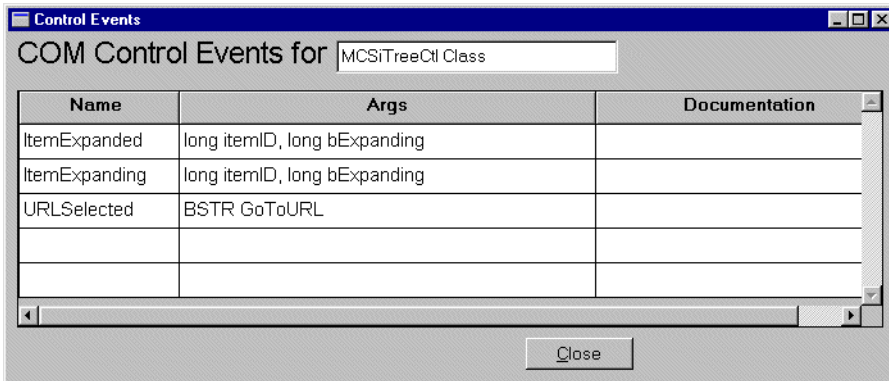


**Figure 18-5** The Properties button lists the properties specified for the selected control.



**Figure 18-6** The Methods button lists the methods for the selected control.





**Figure 18-7** The Events button lists the events specified for the selected control.



# 19 Push Button Widgets

Typically, one of the easiest ways to let a user carry out an action or procedure is to provide a push button. When a user clicks on a push button with the mouse or presses a mnemonic on the keyboard, the push button looks as though it is being pushed in and released, and an action is performed. This action is specified as a control string that is attached to the push button (refer to Chapter 18, “Programming Control Strings,” in *Application Development Guide* for specific information).

This chapter describes

- How to identify a push button—using a descriptive label and mnemonic identifier or, on a GUI platform, using an icon to illustrate the button's action.
- How to specify a cancel and default push button on your screen.
- How to make a push button inactive on screen entry.

At runtime, a user can choose a push button:

- By clicking on it with a mouse, or tabbing to it and pressing Enter.
- By pressing the push button's mnemonic—the underlined character (or alternate colored push button in character mode).
- By pressing Enter—if focus is on the push button, or when focus is on a widget that is not a push button, if the push button is the screen's default push button.
- By pressing Esc—if the push button is the screen's cancel push button.

---

# Identifying a Push Button

---

Push buttons can have:

- A literal label that allows a user to identify what action the push button performs—such as OK, Cancel, Add, etc.
- A mnemonic identifier that allows a user to activate the button by using the keyboard.
- An image or icon that illustrates that the push button is armed, and other icons to indicate that the button is active or inactive (on GUI platforms only).

## How to Define the Push Button's Text

- Under Identity, enter the text in the Label property. By default, the text is centered on the push button.

For Motif, you can change the default alignment of label text on push buttons by changing the value in the Panther-distributed resource file. Refer to “Aligning Button Labels in Motif” on page 10-6 for information.

Store standard push buttons, such as OK and Cancel, in a repository entry. Then you can copy the push buttons from the repository to each application screen where they are needed. This ensures that all push buttons of the same type inherit identical behavior and appearance from a single source. It also saves development time by allowing you to create each push button once, no matter how often it appears in your application.

## How to Adjust the Size to Fit its Textual Content

Select the push button (and any widgets that take a label). Under Geometry, set the Size to Contents property to Yes.

All trailing spaces are eliminated and the widget is reduced in size. Leading spaces are also eliminated from push buttons and toggle buttons so that the label is centered on the widget.

**Note:** If the push button is blank (has no label), the length of the widget is set to 1.

## How to Identify the Keyboard Mnemonic

Identify a mnemonic for a push button to provide users with an alternative way of choosing a push button. The mnemonic is defined as one of the characters in the push buttons label.

1. Select a push button.
2. Under Identity, type the label in the Label property.  
For example, type the word `Cancel`.
3. Under Identity, enter the letter (or its position) that will serve as the access key in the Mnemonic Position property.

You can enter any of the letters that appear in the label as the mnemonic—uppercase `C` defines the first letter, lowercase `c` defines the fourth letter; the numbers 1 and 4 would designate the same letters, respectively.

4. Choose OK.

The property setting appears as character position and character. 1'C' denotes the first character in the string, which happens to be the letter C.

At runtime, the mnemonic is underlined or displayed in a contrasting color in character mode.

---

## Using Pictures on Push Buttons

---

In addition to specifying a textual label on a push button, you can specify that an image be displayed when the application is operating on a GUI platform. If the specified picture cannot be found at runtime, then the label text is displayed instead.

Panther supports BMP (.bmp), GIF (.gif), JPEG (.jpg) and PNG (.png) files under Windows and Motif as well as XPM (.xpm, .xpm1, .xpm3, and .xbm) files on Motif.

You can display three different images, depending on the button's state:

- Active Pixmap—To indicate an active push button (that has not been pressed).
- Armed Pixmap—To indicate an armed button (one that has been pressed). This property is available only if you have specified an Active Pixmap.
- Inactive Pixmap—To indicate an inactive or unavailable (grayed) button. This property is available only if you have specified an Active Pixmap.

## How to Assign a Bitmap or Pixmap to a Push Button

1. Select the push button.
2. Under Format/Display, enter the filename of the picture in the Active Pixmap property. Choose OK.  
Optionally, choose More on the Properties window to display the Select Library Member dialog box where you can select the desired file.
3. (Optional) Enter the filename for the Armed Pixmap and Inactive Pixmap properties. Choose More to display the Select Library Member dialog box where you can select the desired file for each respective property.

**Note:** In addition, the Select Library Member dialog box provides the capability to add image files on the system to a library with the Add button. If you choose the Add button, the Select File dialog box opens and you can choose the file that you want to include in a library.

In Windows, bitmapped images automatically scale to fit within the specified geometry of the push button. In Motif, the push button resizes to accommodate the image. However, once the image is on the push button, the image scales to fit within the specified geometry of the push button.

To ensure portability, consider the size of the image as well as the size of the push button; determine whether the picture can be accommodated.

## How to Create Images in your Application

Creating pictures to use in your application is GUI-dependent. In Windows, you can use the image editor or paintbrush utility to create bitmaps. Motif provides a bitmap utility. Use this or create a pixmap file in the standard pixmap format, either as a text file or via a utility provided with your GUI.

In addition, there are commercially available software packages that provide bitmapped images that can be incorporated into your application.

Refer to “Displaying a Picture on Widgets” [on page 21-10](#) for more information on installing and using images in your application.

---

# Establishing Push Button Behavior

---

There are several ways to control how push buttons are initialized and behave on your application screens. You can:

- Identify a default/cancel push button.
- Define the button's status, that is, whether it is active or inactive.
- Attach a control string to be executed when the button is selected.

## Assigning Default and Cancel

A default push button is activated when the user presses Enter. The control string attached to the push button is executed. Only one push button on a screen can have the Default/Cancel property set to Default. Specifying a default push button automatically resets the Default/Cancel property to Neither on any other default push button on the screen.

A cancel push button is activated when the user presses Esc (or the logical EXIT key). The control string attached to the push button is executed. Only one push button on a screen can have the Default/Cancel property set to Cancel. Specifying a cancel push button automatically resets Default/Cancel property to Neither on any other cancel push button on the screen.

In addition, a push button can be identified as both the cancel and default button on a screen.

## How to Specify a Default and/or Cancel Push Button

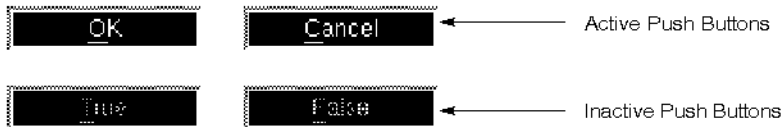
1. Create or select the push button on your screen.
2. Under Identity, choose the desired setting in the Default/Cancel property:
  - **Default**—Identifies the push button as the button to execute when the Enter or NL (New Line logical key) key is pressed.
  - **Cancel**—Identifies the push button as the button to execute when Esc or the EXIT key is pressed.
  - **Both**—Identifies the push button as the button to execute when Enter or Esc is pressed.
  - **Neither**—The default specification for a push button. The push button is executed only when it is chosen.

## Setting the Initial State

When you first add a push button to a screen, it is active; that is, the push button can receive focus and respond to user-generated actions, such as a key press and mouse activity.



When a push button is inactive, it appears grayed and cannot receive focus. You can make a push button active or inactive at runtime by changing the Active property programmatically. For information on changing properties at runtime, refer to “Properties” on page 19-40 in *Application Development Guide*.



**Figure 19-1** The label on an inactive push button appears to be grayed and, therefore, cannot receive focus.

## How to Initialize the Push Button's Status

1. Create or select a push button.
2. Under Identity, set the Active property to the desired setting; specify whether the button will be active or inactive on screen entry:
  - No—Initializes the button in an inactive state; it will appear grayed and cannot receive focus at runtime. If an Inactive Pixmap is assigned to this button, that will be displayed on the button.
  - Yes (default)—The button is active. If an Active Pixmap is assigned to this button, that will be displayed on the button.

## Attaching an Action to the Button

To cause an action to take place when a push button is selected, you attach a control string. The type of action performed by a control string is determined by the leading characters in the control string. Table 19-1 summarizes the syntax you use in the Control String property under Validation.

Refer to Chapter 18, “Programming Control Strings,” in *Application Development Guide* for more information on control strings.

**Table 19-1 Control string specifications and examples**

<b>Leading character</b>	<b>Action</b>	<b>Example</b>
None	Display screen	loginscreen
&	Display stacked window	&(5,4)status
&&	Display sibling window	&&(5,4)status
^	Invoke a function	^funcname
!	Invoke OS program	!dir

## Displaying a Screen

To display another screen when the push button is pressed, specify the name of a screen, along with the appropriate leading character, in the Control String property under Validation for the push button. You can also specify the geometry of the screen and its location. The examples in Table 19-1 specify that the `status` screen displays at the fifth row and fourth column of the physical display (refer to “Screen Display Defaults” on page 13-4 in *Application Development Guide* for details on specifying a viewport).

You can display a screen as a:

- **Form**—When the screen is displayed, all open windows, including the calling screen, are first closed. No special syntax is required.
- **Stacked window**—When the screen opens as a stacked window, all open screens/windows (including the calling screen) remain open with the new window on top. Although the user can bring a lower screen to the top of the display stack by simply clicking on it, the focus remains on the called stacked window until the user takes some action on that window.
- **Sibling window**—When the screen opens as a sibling window, all open screens/windows (including the calling screen) remain open with the new window on top. The new window becomes a sibling of the previously active window, and of all windows that are siblings to the previously active window. The user can move between sibling windows, changing the focus, by clicking on a lower screen or by pressing the VWPT key.

## Invoking a Function

To invoke a function, specify the name of the function preceded by a caret (^) in the Control String property under Validation for the push button, for example, `^myproc`. Panther first checks for installed C functions and then JPL code. For more information on the order of precedence for calling JPL, refer to “Precedence of Called Objects” on page 19-24 in *Application Development Guide*.

You can invoke:

- Your own C programs—Must be installed (refer to Chapter 20, “Writing C Functions,” in *Application Development Guide* for information on adding your own C functions).
- Panther library functions—Are pre-installed in Panther executables.
- Panther built-in functions—These functions are preceded with a `jm` prefix. They need no prototyping or installation. Refer to Chapter 3, “Built-in Control Functions,” in *Programming Guide* for a description of all Panther built-in functions.

**Note:** The control string `jm_keys XMIT` when attached to a push button causes an infinite loop. This occurs because the act of pressing `XMIT` actually activates a push button.

## Invoking a Program

To invoke a system program from a push button, specify the program and its arguments, preceded by an exclamation point (!) in the Control String property under Validation for the push button. At runtime, the string (program name and arguments) are expanded and passed to the operating system for execution. Depending on the operating system and the program, after execution is complete, the user is prompted to press the Spacebar before returning.



# 20 Selection Widgets

This chapter describes each of the selection widgets and how, when they are grouped together, they provide users with choices. Selection widgets include:

- Check boxes—Allow users to make one or more choices.
- List boxes—Can function as a selection list box, allowing the user to select any number of items from the list, or as an action list box, where each item in the list has a corresponding action or event.
- Radio buttons—Allow users to make a single selection.
- Toggle buttons—Allow users to select and deselect a behavior or action.

Selection widgets provide a graphical way of displaying valid choices to your users. Each of these widget types offers a unique presentation and facility for selecting and indicating which option or choice the user has made.

To determine what choice a user has made, refer to “Selection Group Data” [on page 19-44](#) in *Application Development Guide* for information on querying selection groups.

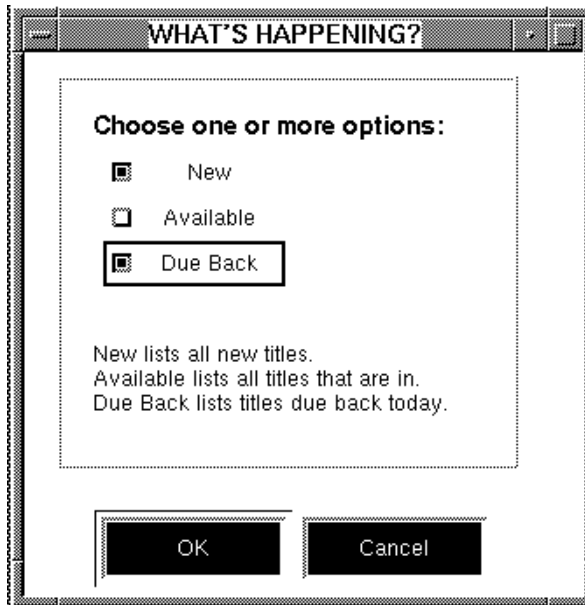
---

# Using Check Boxes

---

Check boxes provide a method for turning conditions off or on. When you group check boxes (refer to “Grouping Selection Widgets” on page 20-10), you provide a set of choices from which a user can select one or more options, or none. At runtime, when a check box item is selected, a check mark or an x (in Windows) appears next to the selected item; a three-dimensional, shaded box changes color in Motif.

In the following illustration, the grouped check boxes allow for one or more selections to be made. You can use the library function `sm_getfield` to determine what choices a user has made.



**Figure 20-1** One or more options can be checked.

Use the Label property to define the literal text that displays next to the check box. If your application runs on a GUI platform, you can assign a pixmap to a check box so that an image appears next to the check box instead of text; assign one to display when

the check box is selected, another to indicate that it is not armed, and still another to indicate that the check box is inactive. Refer to “Displaying a Picture on Widgets” on [page 21-10](#) for more information.

---

## Using List Boxes

---

A list box presents the user with a noneditable, vertical column of choices. The list box can be one of two list box types:

- **Select Any**—Allows the user to select any number of items from the list.
- **Action**—Has an action or event associated with each item in the list.

When you group a list box (choose Edit→Group→Create→Selection Group), you can control the number of allowable selections and assign group entry and exit functions to the widget.

## How to Define a List Box's Behavior

Select a list box. Under Identity, specify the how you want to use the list box in the Listbox Type property:

- **Select Any**—The user can choose zero to any number of items (to change the number of allowable selections, you must make the list box a selection group; refer to “Grouping Selection Widgets” on [page 20-10](#) for more information).

To make a selection, the user can:

- Use arrow keys to move the cursor to the desired item in the list, and then press the Space bar to select.
- Enter the first character of a listed item to move the cursor to that occurrence.
- Mouse click to select and deselect items. Shift+click to select contiguous items; Ctrl+click to select noncontiguous items.

- Ctrl+click to deselect an already selected item.
  - Mouse click to select the first item and drag the mouse to select additional contiguous items.
  - Double-click on the list box to execute the associated control string set in the Double Click property.
- Action—Allows the list box to act like a menu. You can associate an action or event with each item in the list. When the user clicks on one of the items, the corresponding control string is executed (a double-click event is not executed on an action-type list box).

Selected items are displayed in reverse video.

You can determine the user's selection by querying the `selected` property of the list box's occurrences.

Refer to “Making Selections in List Boxes” [on page 23-11](#) in *Application Development Guide* for more information about extended selections in list boxes.

By default, a list box has a border around it. For character mode, you can assign a border style and title as well. You can also set the `Border` property to `No` to display no border at all. These settings have no effect on the GUI platforms. However, to include a title on a list box, you can draw a box (use the box widget) and assign a label to the box. Refer to “Adding Borders to Widgets” [on page 21-9](#) for details on creating borders and boxes around widgets.

Figure 20-2 illustrates three list boxes that are synchronized arrays. In this example, the data from three different, but related, columns scroll together and allow the user to select one or more records from the group. This is achieved by using a validation function on each list box that ensures that the other list boxes have the same occurrence selected as the one the user selected.



The image shows three vertical list boxes side-by-side, each with a header and eight items. The first list box is titled 'Last Name' and contains: Barton, Bassett, Bastille, Bastin, Batterman, Batterson, and Bath. The second list box is titled 'First Name' and contains: Eric, Loretta, Amy, Bryant, Phillip, Frank, and Fred. The third list box is titled 'Phone' and contains: 555-3101, 555-4343, 555-1033, 555-9874, 555-1234, 555-5300, and 555-8936. The list boxes are synchronized, meaning they all show the same row of data at the same time.

Last Name	First Name	Phone
Barton	Eric	555-3101
Bassett	Loretta	555-4343
Bastille	Amy	555-1033
Bastin	Bryant	555-9874
Batterman	Phillip	555-1234
Batterson	Frank	555-5300
Bath	Fred	555-8936

**Figure 20-2** Synchronized list boxes can provide a means of selecting multiple records.

To restrict the number of selections to only one, define a list box as a group (refer to [page 20-10](#), “Grouping Selection Widgets”) and set the # of Selections property appropriately.

## Scrolling the Data

You define how many occurrences display on the screen. When the number of items, or occurrences, exceeds what can be displayed at one time, the user can scroll the list.

### How to Make a Scrolling List Box

1. Select a list box.
2. Define the number or rows that display on the screen by doing either of the following:
  - Under Geometry, enter the number in the Onscreen Rows property.
  - Drag the widget by one of its resize handles to the desired height.

If the number of elements in the array exceeds what can be displayed onscreen in the list box:

3. Under Geometry, set the Scrolling property to Yes. Additional scrolling-related properties are displayed. By default, a vertical scroll bar appears on the widget.

4. Under the Scrolling property in the Max Occurrences subproperty, enter the total number of occurrences required to accommodate the data. A blank (empty) value will return the maximum number of elements.
  5. (Optional) Under the Scrolling property, set the Vertical Scroll Bar property accordingly.
    - Yes—(default) Allows the user to display offscreen occurrences by clicking on the vertical scroll bar.
    - No—The scroll bar is not displayed on the widget. The user can display offscreen occurrences by using the arrow keys or the SHIFT or SCROLL logical keys.
- Note:** The ZOOM logical key displays offscreen occurrences only for an action-type list box.

## Populating List Boxes

### How to Populate a List Box with Data

Do any of the following to populate a list box:

- The list box's Initial Text property.

Select the list box. Under Format/Display in the Initial Text property, enter the data (one item per line). Press Enter after each line. You can also choose the Read File button to read data from an external file. Choose OK to save your additions and to exit the Initial Text dialog box.
- At runtime, from a database or from an LDB screen (refer to “Using Local Data Blocks” [on page 25-7](#) in *Application Development Guide*).
- The screen's Entry Function.

Select the screen that contains the list box. Under Focus in the Entry Function property, enter the name of the function that will populate the list. At runtime, when the screen opens, its screen entry function will populate the list.

## Performing Actions

You can assign a control string to each occurrence in a list box to cause an action or event to occur when the user chooses items on the list. In this way, the list box functions as a menu might. However, if you define the list box to function as a selection group (by choosing Edit→Group→Create→Selection Group), the behavior is controlled by the group validation via a control string attached to the entire list box (for more information on group validation hook functions, refer to “Group Functions” on [page 44-25](#) in *Application Development Guide*).

### How to Make a List Box Function Like a Menu

1. Create or select a list box.
2. Under Identity, specify Action in the Listbox Type property.
3. Under Validation in the Control String property, enter a control string for each list item by either:
  - Typing the string in the Setting field of the Properties window. Press Enter between each to define one control for each occurrence.
  - Choosing More. The Zoom window opens where you can enter or edit the control strings. Assign one control string per occurrence. Choose Close to exit.

Refer to Chapter 18, “Programming Control Strings,” in *Application Development Guide* for details on control string syntax and invoking a control string from a widget.

---

## Using Radio Buttons

---

Radio buttons allow the user to make exactly one selection (or none) in a group. When the user selects a radio button at runtime, its button appears darkened, and any other radio buttons in the group are immediately deselected.

By default, a radio button specifies that zero or one selection can be made, and on screen entry, none of the radio buttons are selected.

## How to Define the Number of Selections

1. Select a radio button group (refer to “Grouping Selection Widgets” on page 20-10 for details on creating/selecting selection groups).

The Type field at the top of the Properties window should indicate that a Group is selected.

2. Under Identity, specify either of the following specifications in the # of Selections property:
  - 0 or 1—(default) Allows the user to select one button or none.
  - 1—The user must select one of the radio buttons in the group.

## How to Establish an Initial Selection

1. Select a radio button group (refer to “Grouping Selection Widgets” on page 20-10 for details on creating/selecting selection groups).

The Type field at the top of the Properties window should indicate that a Group is selected.

2. Under Identity in the Init Selections property, enter the number of the radio button that should appear selected on screen entry. The members of a group are numbered sequentially from left to right and top to bottom.

For example, in Figure 20-3, the radio button group consists of two buttons. The Init Selections property is set to 1. So when the screen opens, the first radio button (Customer) is selected. This group of radio buttons also specifies that one of the two radio buttons must be chosen before the user can proceed.

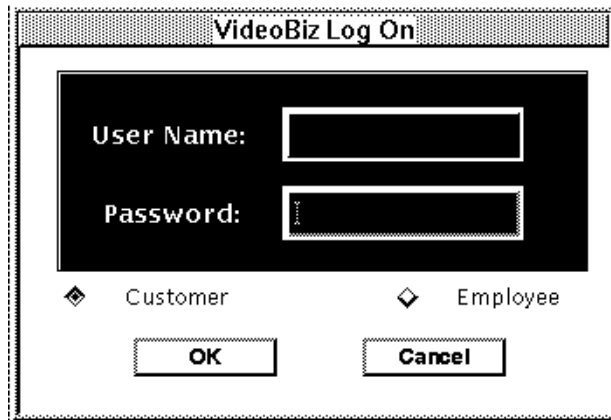


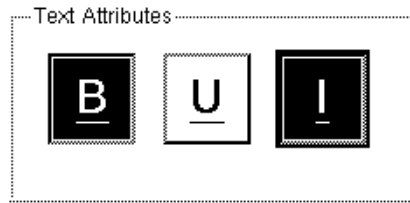
Figure 20-3 The user can choose one radio button: Customer or Employee.

## Using Toggle Buttons

Toggle buttons are in/out graphical buttons. At runtime, when a toggle button is pushed, a particular behavior is turned on. The button remains on—or in the “in” position—until it is selected again. It then appears to be in the off or “out” position—so the behavior is toggled off.

By default, any number of toggle buttons can be selected. You can redefine the number of possible selections by grouping the buttons and setting the # of Selections property for the group.

Figure 20-4 illustrates how toggle buttons can be used to turn a specific behavior on or off. The screen, typical of many word processing applications, allows the user to choose **B** to implement bold text, U to underline, and *I* to italicize.



**Figure 20-4** Both the **B** and **I** toggle buttons are selected—turning both bold and italic attributes on.

Use the Label property to define the literal text that displays on the toggle button. By default, the text is centered. In Motif, you can change the default alignment of the text by changing the value in the Panther distributed resource file (refer to “Aligning Button Labels in Motif” [on page 10-6](#) for information).

If your application runs on a GUI platform, you can assign pixmaps to a toggle button so that an image appears on the button instead of text; assign one to display when the button is selected, another to indicate that it is not armed, and still another to indicate that the button is inactive. Refer to “Displaying a Picture on Widgets” [on page 21-10](#) for more information on displaying pictures on buttons.

---

## Grouping Selection Widgets

---

When you group selection widgets, you can define behavior for the group as a single entity, as opposed to defining behavior for individual widgets. By virtue of creating a group, you gain access to properties that define the group as a whole—such as group name, number of selections, tabbing behavior, group entry, exit and validation functions—and you can use group-specific library functions at runtime.

This section describes:

- How to create a group (only selection widgets of the same type can be grouped).

- How to set group properties.
- How to change group members—adding and removing widgets.

Depending on the widget type and the property specification for the group, a user can make one or more selections from the group.

## Creating Selection Groups

### How to Create a Selection Group

1. Create the widgets that are to be a part of the group—these can be any number of check boxes, radio buttons, or toggle buttons, or a single list box.

All widgets in the group must be of the same type. They can be close together or far apart on the screen.

2. Select the widget or widgets that you want to assign to the group.
3. Choose **Edit**→**Group**→**Create**→**Selection Group**. An invisible group widget is created; notice that the **Type** field at the top of the **Properties** window indicates that a **Group** is selected. You now have access to the group properties.

## Identifying the Members of a Group

To identify and assign property values to all widgets that comprise a group, you must first find all members of the group. You can set properties for all members and thereby ensure that they are the same color, font, size, etc.

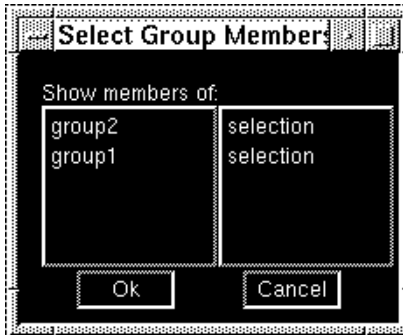
### How to Identify the Members of the Group

1. Do either of the following:
  - Choose **View**→**Widget List** and select the group widget from the **Name** list
  - Select any widget or widgets associated with a group.
2. Choose **Edit**→**Group**→**Select Members**.

All members of the group are selected, and you can set the intersecting properties for the widgets.

If you select widgets associated with two or more different groups, the Select Group Member dialog box (Figure 20-5) opens. Select the group from the left hand column and choose OK.

All members that comprise the group are selected.



**Figure 20-5** The Select Group Members dialog box displays all the groups with which the selected widget is associated.

## Specifying Group Properties

While you can define unique property settings for each member of a group, you can also define how the group as a whole should behave through its properties. There are two methods you can use to access a group's properties:

**Note:** Although a group is selected, its members are not visibly selected on the workspace screen.

### How to Select a Group from the Widget List

1. Choose View→Widget List.
2. Select the desired group from the Name list. A group name is, by default, `group` followed by a number. The hidden group identifier precedes the members that comprise that group. Be sure that no other widgets are selected. Only the group identifier should be highlighted in the Widget List to access its properties.



## How to Select a Group via a Group Member

1. Select any member of the group.
2. Choose Edit→Group→Select Groups. The Properties window now displays the properties associated with the group.

Table 20-1 lists the group properties alphabetically along with the heading under which they can be found in the Properties window.

**Table 20-1 Group properties**

<b>Property</b>	<b>Heading</b>
# of Selections	Identity
Alt Tab Stop	Focus (dependent on setting of Next and/or Prev Tab Stop)
Autotab	Focus
C Type	Identity
Entry Function	Focus (for details, refer to “Widget Events” <a href="#">on page 17-12</a> in <i>Application Development Guide</i> )
Exit Function	Focus (for details, refer to “Widget Events” <a href="#">on page 17-12</a> in <i>Application Development Guide</i> )
Inherit From	Identity
Init Selections	Identity
Memo Text [1-9]	Identity
Name	Identity
Next Tab Stop	Focus
Prev Tab Stop	Focus
Validation Func	Validation (for details, refer to “Widget Events” <a href="#">on page 17-12</a> in <i>Application Development Guide</i> )

## Naming a Group

Panther assigns a default name to a group when it is created—`group` followed by a consecutive number. The groups on a screen are numbered in the order in which they are created. So, the first group you create on the screen is identified as `group1`.

### How to Assign a Name to the Group

1. Select the group. (For methods of selection, refer to “How to Select a Group from the Widget List” [on page 20-12](#) or “How to Select a Group via a Group Member” [on page 20-13](#).)
2. Under Identity, type the name in the Name property.

Group names follow the same conventions as widget names; they can be up to 31 character long, and must start with an alphabetic character. A group and widget cannot share the same name on the screen.

## Setting the Number of Selections Allowed in a Group

### How to Define the Number of Selections that a User Can Make in a Group

1. Select the group. (For methods of selection, refer to “How to Select a Group from the Widget List” [on page 20-12](#) or “How to Select a Group via a Group Member” [on page 20-13](#).)
2. Under Identity, set the # of Selections property to the desired number. Each widget type has a default setting for the number of selections. There are three possible settings:
  - 0 or 1—Specifies that no selection or only a single selection is required. This is the default specification for grouped radio buttons.
  - 1—Specifies that one and only one selection must be made.
  - Any—Specifies that zero or many selections can be made. This is the default specification for check boxes, toggle buttons, and list boxes.

**Note:** The ZOOM logical key is supported only for a list box in a Panther selection group with the # of Selections property set to 0 or 1.

## Specifying an Initial Selection

### How to Identify an Initial Selection for a Group

1. Select the group. (For methods of selection, refer to “How to Select a Group from the Widget List” [on page 20-12](#) or “How to Select a Group via a Group Member” [on page 20-13](#).)
2. Under Identity, enter the number of the selection in the Init Selections property. Members of a group are numbered left to right and top to bottom on your screen, starting with 1.

Use the Widget List to determine the number that corresponds to the desired widget. For groups that allow more than one selection, separate the numbers with a comma in the property specification.

For example, to specify that the second widget in a group be the initial selection on screen entry, enter 2 in the Init Selections property. Figure 20-6 illustrates how that might look in a group consisting of four check boxes—two-by-two. The second check box in the group is already selected on screen entry.



**Figure 20-6** The second widget in the selection group is identified as the initial selection for the group.

## **Adding and Removing Group Members**

To change the content of a group—by either adding or removing widgets—you essentially redefine the group as a whole.

### **How to Add a Widget or Widgets to an Existing Group**

1. Select all members of the existing group, by either:
  - Selecting the group widget from the Widget List.
  - Selecting at least one member of the group.
2. Choose Edit→Group→Select Members.
3. Select (Shift+click) any additional widgets that you want to add to the group.
4. With all the widgets selected that are to be grouped, choose Edit→Group→Update Group Members.

The existing group now includes the additional widgets, and the group's properties are displayed in the Properties window.

### **How to Remove a Widget from a Group**

1. Select members of the group that you wish to keep in the existing group by either:
  - Clicking on the desired widgets in the group.
  - Selecting at least one member of the group and choosing Edit→Group→Select Members. Then deselect any of the members that should be eliminated from the group.
2. With the widgets selected that are to be grouped, choose Edit→Group→Update Group Members.

The existing group is redefined to include the new set of widgets, and the group's properties are displayed in the Properties window.

# 21 Graphics Widgets

Adding graphical elements to your screens can enhance the look and, in addition, can serve to illustrate, organize, and group information—thus achieving an attractive presentation as well as a functional one.

This chapter describes how to:

- Use the graphic widgets: Lines and Boxes.
- Add borders and titles to character mode widgets.
- Display pictures on different types of widgets. For information about displaying pictures on toolbar items, refer to [page 25-14](#), “Displaying Pictures on Toolbar Items.”
- Include customer drawn widgets.

---

## Using Boxes and Lines

---

Drawing a line or box on your screen is as simple as dragging the mouse.

### How to Create a Line or Box

1. Choose Create→Line or Box, or select the icon on the Create toolbar.

2. Drag the mouse horizontally, vertically, and, in the case of a box, diagonally to the desired length or size.

The widget acquires a native line style (Default). You can change the style, as well as assign foreground colors and background colors.

## How to Control the Line or Box Placement

To control where the line or box starts and/or ends on the screen:

1. Select the line or box widget.
2. Under Geometry, define the start and end positions in the following properties.

Use grid positions if portability is an issue, or use fractional specifications if desired. (Refer to Table 9-1 [on page 9-7](#) for a list of valid units of measurement.)

- Start Row
- End Row (not available on horizontal lines)
- Start Column
- End Column

**Note:** Lines and boxes align on grid coordinates by default.

## Specifying Styles for Boxes and Lines

When you draw a line or box on your screen, a default style is assigned which is appropriate to the native environment. However, you can choose other styles as well as assign a style that can, ultimately, make your specification portable to other platforms.

## How to Define a Line Style for a Box or Line Widget

- Under Identity, set the Line/Box Style property to the desired style.

You can specify a line name alias by choosing Other in the Line/Box Style. The Other Style subproperty is displayed. Enter the name as defined in the configuration map file for your application.

To control line/box style mapping and alias names, you can assign the desired specification in the configuration map file. Refer to “Defining Line and Box Styles” [on page 45-34](#) in *Application Development Guide* for more information on assigning and mapping line/box styles.

## Character Mode Lines and Boxes

Styles 0 through 9 are native to Panther running in character mode. Style 1 is defined as the default line style. When you assign a character-specific style as the Style property value for a line or box style in the editor, that style is mapped to style 1 on non-character mode Panther applications. GUI-specific styles map to style 1 when running in character mode.

## GUI Lines and Boxes

Motif's native line styles are: single, double, dash, double dash, etched in, etched in dash, etched out, etched out dash, and none. The Default setting maps to a single line style. Motif's box styles are: etched in, etched out, in, and out; the Default setting maps to Etched In.

Native styles for lines and boxes in Windows are: single, dash, dot, dashdot, and dashdotdot. The Default setting maps to a single line style for both graphical elements.

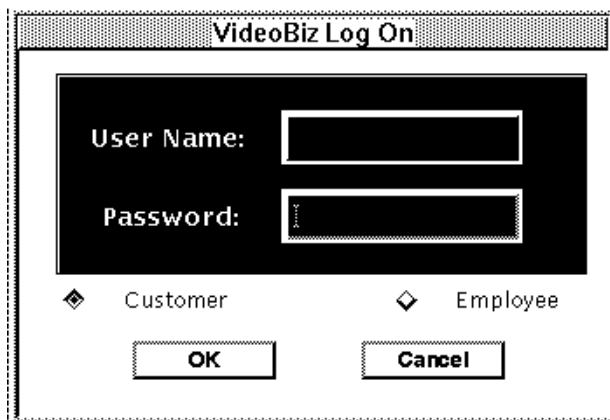
If the 3D feature is enabled, the etched in style is also available under Windows. Refer to “Giving Screens a 3D Appearance” [on page 10-23](#) for an explanation of how line and box appearance is affected on 3D screens on these platforms.

## Creating Frames

Frames can help you design attractive, logically structured, and easy-to-use screens. You create a frame by using the box widget. The box can be titled or untitled. Use frames around widgets on your screen to:

- Physically group related widgets together.
- Create areas of information on the screen.

**Note:** Don't confuse the Box widget with borders. For character mode applications, you can define border properties for list boxes and multiline text widgets.



**Figure 21-1** Boxes can be used to enclose widgets, creating a physical group of related widgets as opposed to a functional group.

## How to Include a Title on a Box

When you use a box to enclose, for example, related widgets, you can also provide a title on the box that identifies the fields as a group.

**Notes:** The Label property is not supported for boxes in Panther reports.

1. Select a box widget.
2. Under Identity, enter the text in the Label and choose OK. The Justification subproperty is displayed.  
By default, the label text is centered on the top border of the box.
3. To change the text alignment, under Identity, set the Justification subproperty:
  - Left—Aligns the label text to the box widget's left border.
  - Right—Aligns the label text to the box widget's right border.
  - Centered—Aligns the label between the box widget's left and right borders.





**Figure 21-2** The box label can be a title or instructions to the user.

If you want to control how widgets resize and shift if you port your screens, you can use the screen's positioning properties, or you can identify the box widget as a positioning region in and of itself.

4. (Optional) Under Geometry, set the Position Region property appropriately.
  - Yes—(default) Defines the box as a positioning region. Additional subproperties are available under the Positioning heading. Use these properties to define how much space you want around widgets and how much of a margin should exist between the box edge and the widgets within its borders (refer to “Using Boxes as Positioning Regions” on page 21-6 for more information on Positioning properties).
  - No—The box and any widgets within its border rely on the screen's positioning properties to define how much space is maintained between objects. The box widget's positioning properties are not available.

## Adding Lines

Add lines to your screen to create physical definition and provide organization to the content of the screen. Like the box widget, you can choose the style of line.

---

# Using Boxes as Positioning Regions

---

Panther uses a positioning algorithm to recalculate screen size and widget positions (refer to “Controlling Widget Positions and Screen Size” [on page 6-13](#) for more information) when you open a screen in an environment that is different from the one in which it was created. This algorithm uses the settings of the screen's Positioning properties (subproperties under Geometry).

Similarly, each box has its own set of Positioning properties, available when the Position Region property is set to Yes, that can be set independently of the screen. When you open a screen in a new environment, Panther recognizes each box as an autonomous positioning region and recalculates its size and the positions of widgets within it.

When a box widget's Position Region property is set to Yes, its Positioning properties let you:

- Control spacing between widgets
- Set a minimum margin within the box's border
- Control adjustments in box size and widget positions.

**Note:** Positioning properties are recognized only in GUI environments.

## Spacing Widgets within a Box

Two properties, Minimum Horizontal Space and Minimum Vertical Space, let you specify the minimum amount of space between widgets in any direction. You can set the amount of space in any unit of measurement (refer to Table 9-1 [on page 9-7](#)). If all available whitespace is used up between widgets, Panther moves them or adjusts the box size in order to maintain the minimum spacing required.

### Minimum Horizontal Space

Determines the minimum amount of space maintained between widgets that are horizontally contiguous. The default 0 allows widgets to touch. If you place widgets closer than the distance specified by this property, Panther does not try to increase the space between them.

### Minimum Vertical Space

Determines the minimum amount of space maintained between widgets that are vertically contiguous. The default value is 1 grid unit. If you place widgets closer than the distance specified by this property, Panther does not try to increase the space between them.

## How to Define a Margin Within the Box Frame

The Region Margin property creates a buffer of white space between the box's inner border and its contents.

1. Select a box widget.
2. Under Geometry, ensure that the Position Region property is set to Yes. The Positioning properties are then available.
3. Under Geometry, expand the Positioning heading and set the Region Margin property to the desired amount of space.

The default 0 allows widgets to touch the box border. If you place widgets within the margin, Panther does not try to increase the space between them and the box's border.

You can set the amount of space in any unit of measurement (refer to Table 9-1 on page 9-7). For example, enter 4 to indicate four pixels or 3g to indicate three grid units.

## How to Control the Box Dimensions and Widget Start Positions

When you open a screen in an environment different from the one in which it was created, Panther attempts to minimize differences between the presentation of the same screen on different platforms. In so doing, it typically adjusts a box widget's dimensions and the start positions of other widgets within it.

1. Select a box widget.
2. Under Geometry, ensure that the Position Region property is set to Yes. The Positioning properties are then available.
3. Under Geometry, expand the Positioning heading and set the Horizontal Shrinking and/or Vertical Shrinking subproperties accordingly:
  - Decrease region size—Adjust widget positions and box size according to changes in font name and point size. This is the default setting.
  - Keep region size—Keep the box size unchanged;

Panther adjusts widget start positions in order to simulate their original proximity; when moving from character mode to GUI modes, this is likely to leave extra white space at the box's borders.

- Prevent grid shrinking—Keep the box size and widget start positions unchanged. When moving from character mode to GUI modes, the white space between widgets is liable to increase.

---

# Adding Borders to Widgets

---

## How to Define a Widget Border and Style

The list box and multiline text widgets both have properties that allow you to specify a border style as well as a title. These properties have no effect on the GUI platforms; however, you can achieve the same look in the GUI environments by using a box widget around the widget.

1. Select a list box or multiline text widget.
2. Under Format/Display, set the Border property.
  - Yes—(default) Border subproperties are displayed. Continue to step 3.
  - No—Removes the border from the selected widget.
3. Under Border, specify the desired line style (0 through 9) in the Style property (default is 1 which maps to a single line style).

The numeric style specifications are defined in your video file (refer to “Borders and Line Drawing” [on page 7-44](#) in *Configuration Guide* for information on border styles).

## How to Include a Title on a List Box or Multiline Text Widget

1. Select a list box or multiline text widget.
2. Under Format/Display, set the Border property to Yes. The additional border properties are displayed.
3. Under Border, enter a title or any descriptive information in the Title subproperty. The text is aligned to the left corner of the border.

---

## Displaying a Picture on Widgets

---

In general, a textual label is displayed on push buttons, toggle buttons, radio buttons, check boxes, static and dynamic labels.

Panther provides properties for you to specify the name of a bitmap image that will display instead of the text on these particular widgets. These property settings have no effect in character mode.

To create a bitmap, use the image editor or paintbrush utility on Windows. On Motif, use the `bitmap` utility provided with X, or create a pixmap file in the standard pixmap format, either as a text file or via a utility provided with your GUI. Most GUI platforms also provide sample image files that you can use.

Panther supports BMP (`.bmp`), GIF (`.gif`), and JPEG (`.jpg`) files under Windows and Motif, XPM (`.xpm`, `.xpm1`, `.xpm3`) and XBM (`.xbm`) files on Motif.

### Specifying the Image

Under the Format/Display heading in the Properties window, there are three image-related properties. Via these properties, you can assign one (static labels take only one) or more images (dynamic labels have only two of these properties) to a widget with each picture depicting a different state: active, armed, or inactive.

**Note:** Specify a textual label in the widget's Label property as well. If the Active Pixmap file is not found at runtime or is not supported by the local GUI, or if you run your application in character mode, the widget's label is displayed instead.

### How to Display a Picture on a Widget

1. Select the widget (push button, radio button, check box, toggle button, static label, or dynamic label).

2. Under Format/Display in the Active Pixmap property, enter the name of the image file or choose More to select the file from the Select Library Member dialog box. Choose OK. Depending on the widget type, additional pixmap properties are displayed.

The image displays immediately if the image file is in one of your open libraries.

At runtime, the specified image is displayed on the widget whenever the widget appears and as long as the widget is active (that is, it can get focus). If the widget is made inactive at runtime, you can specify another image to indicate the widget's state.

3. Under the Active Pixmap property in the Armed Pixmap subproperty, enter the name of the image file or choose More to select it from the Select Library Member dialog box.

The specified image is displayed on the widget when the widget is activated (or armed), that is, when it is pushed, selected, or checked. This property is not available for static or dynamic label widgets because they cannot get focus.

4. Under the Active Pixmap property in the Inactive Pixmap subproperty, enter the name of the image file or choose More to select it from the Select Library Member dialog box.

The specified image is displayed on the widget when the widget is set to be inactive (via the Active property under Identity) or is made inactive at runtime.

**Note:** Via the Select Library Member dialog box, you can add image files on the system to an open library by choosing the Add button.

## Portability of Images Files

To facilitate portability and provide maximum flexibility, omit the filename extension when you specify an image file. At runtime, Panther searches for files in all open libraries using all possible image extensions supported on the local platform.

On the other hand, if you have multiple images of the same name but of different type (for example, `dog.bmp` might be a bitmap image while `dog.jpg` is a photograph), include the extension to ensure that the correct image is displayed at runtime.

**Note:** Under Windows, if you compile the image into your resource-definition file, the resource id must match the Wallpaper Pixmap property setting exactly.

## Storing the Image Files

Under Motif, if the file is not found in an open library, then the path indicated in the resource file is used to search for X bitmap files (for the `XmGetPixmap` resource). Refer to the *OSF/Motif Programmer's Reference*.

Under Windows, Panther searches for bitmaps in these locations, in the order given:

1. The application's resource file.
2. Any DLLs loaded into memory by `sm_slib_load`.
3. Open libraries.

## Sizing the Image

Bitmap images display by default at the size they were created. Default-sized widgets (those having no `Height` or `Width` property settings) will resize to accommodate the image. Once you attach the image, resize the widget to the desired size by either dragging the widget or by setting its `Height` and `Width` properties to accommodate the image.

Under Windows, bitmap images automatically scale to fit within the specified geometry of the widget. Under Motif, the bitmap image is truncated if it cannot fit within the specified dimensions of the widget.

To ensure that the widget size remains fixed and is unaffected by changes to the pixmap image, set explicit values in the widget's `Height` and `Width` properties.

---

# Using Customer Drawn Widgets

---

You can use Panther's push buttons, toggle buttons, and dynamic label widgets to display dynamically drawn content by your application, instead of a label or a static pixmap. To let Panther know that you have installed a drawing function, set the `Customer Drawn` property under `Format/Display` to `Yes`. Use the library function



`sm_attach_drawing_func` to attach the drawing function. This property setting only has an effect in GUIs. When running your application in character mode, the default Panther widgets display.

The widgets behave as they are defined through their properties. In other words, the push button and toggle button are pushed to activate, and dynamic labels are protected from getting focus at runtime. Panther handles all processing for these widgets as it normally would, except for drawing them. However, the shading, where applicable, is still handled by Panther.



# 22 Table Views and Links

Table views and links are widgets used to provide Panther with the information it needs to support database access, specifically for the transaction manager in database applications. A table view is a group of widgets on a screen, generally from the same database table. If there is more than one table view on the screen, links define the relationships between two table views.

This chapter describes using and creating table views and links, and how their property settings are used by Panther's transaction manager to effect automatic SQL generation. Refer to Chapter 33, “Using Automated SQL Generation,” in *Application Development Guide* for details on automated SQL generation.

To use the transaction manager:

- Use the screen wizard or build application screens from scratch using widgets from repository entries that were created as the result of importing your database tables into Panther (refer to “Populating a Repository with Database Objects” on page E-25 for information on populating the repository from a database).

The table views and links on your application screens provide the transaction manager with the information it needs to access your database.

- Gain access to the transaction manager via application or test mode.
- Make sure that you are attached to the database. This can be done via server initialization in JetNet/Oracle Tuxedo applications, or via a direct connection (choose File→Open→Database in the editor or Database→Connect in test mode).

---

# Using Table Views

---

A table view—since it is a group widget—is not visible on the screen in the screen editor or at runtime; however, the widgets that make up the table view are visible. The table view is a set of related widgets, usually associated with and named for a single database table. The table view must be present on the screen if you want to use the transaction manager to provide automatic database access.

A table view is automatically created on each repository entry when you import a database table into a repository. When you copy a database-derived widget from a repository entry, Panther creates a table view on the destination screen, provided the copied widget belonged to a table view in the repository entry.

You can also include additional members (widgets) in a table view that are not a part of the database table in order to display derived data.

## Accessing Table View Properties

To access a table view's properties, do any of the following:

- Choose View→DB Interactions. Select the desired table view from the DB Interactions window.
- Choose View→Widget List. Select the desired table view from the list of widgets.
- Select any widget that is a member of the table view. Then choose Edit→Group→Select Groups.

The Properties window displays the properties associated with the selected table view.

The transaction manager uses table view properties (and link properties, discussed later in this chapter) to execute its commands. Table views are identified by the following properties, many of which are automatically defined through the import process. Refer to “Populating a Repository with Database Objects” on page E-25 for information on the import process.

## Identity Properties

The following properties define the name of the table view and its source of inheritance, and provide for additional comments and programming use.

### Name

Under the Identity properties, specify the name of the table view which generally is the same as the database table. A table view that you create manually by grouping widgets, as opposed to copying it from a repository entry, is assigned a name—`tview` followed by a number. Table views are numbered sequentially. Rename manually created table views to something more descriptive and useful for your development needs.

### Inherit From

Use this property to specify the table's source of inheritance if you manually created the table view. Use the format `repositoryEntry!repositoryTable View Name`. This value is provided automatically when a table view is copied from a repository entry.

### Memo Text

Attach up to nine separate lines of text (`memo1 . . . memo9`) for additional comments or for entering string expressions to use programmatically.

## Transaction Properties

The following properties define how the selected table view is used by the transaction manager:

### Updatable

Defines whether the database table participates in `INSERT`, `UPDATE`, and `DELETE` statements during automatic SQL generation. The possible settings are:

- Yes—(default) The table view is updatable, and therefore, the transaction manager requires that the widgets associated with primary keys (defined in the Primary Key property) exist in the selected table view or in a parent table view on the screen. If widgets associated with the primary keys are not included on the screen, the transaction manager generates an error message.
- No—Protects the database table from being updated.

#### Function

Usually blank. Use this property to specify the name of a transaction event function which will be called instead of the transaction model for the selected table view. For more information on creating custom functions, refer to Chapter 32, “Writing Transaction Event Functions,” in *Application Development Guide*.

#### Model

Usually blank. Use this property to specify the name of the transaction model to use for the selected table view or screen. When the property is left blank, the transaction manager uses the model specified at the next level.

You can specify a transaction model in the Model property under Transaction at the following levels:

- For the table view—If you name a model for the table view, this model takes precedence over the one specified for the screen.
- For the screen—If you name a model for the screen as a whole, this specification takes precedence over the default model of the default database engine. If no model is named, the transaction manager uses the default model of the default engine in combination with the common model.

## Database Properties

The Table, Columns and Primary Key properties define how the selected table view is defined in the database; they are automatically set when a database table or view is imported. The Insert Handling, Update Handling and Delete Handling categories determine how processing occurs for these operations.

#### Table

Specifies the name of the database table. This name is provided automatically if the table view was copied from a repository entry generated by the import process. The table name is used by the SQL generator in the FROM clause of the SQL statement.

#### Columns

Lists the columns belonging to the database table associated with the selected table view. This information is provided automatically if the table view was copied from a repository entry generated by the import process. The column list is not used for SQL generation. It is used during help processing to provide a list of possible columns for use in the link Relations dialog box.

### Primary Key

Lists the columns composing the database table's primary key. This information is generated automatically if it was available from the database engine. Choose More to see all primary keys. The columns listed are used:

- In the WHERE clause of the SQL UPDATE and DELETE statements generated for the selected table view.
- During database import processing to define the Relations property for link widgets.

### Insert Handling

The following settings are available for inserting data in the transaction manager:

- SQL Statement Generation—(default) Generates a SQL statement based on property settings. The Regenerate SQL sub-property can be set to Yes, No or Default.
- Function Call—Calls the specified function. You can specify a function for insert processing and a function for save processing.
- Nothing—No processing is done for the statement.

### Update Handling

The following settings are available for updating data in the transaction manager:

- SQL Statement Generation—(default) Generates a SQL statement based on property settings. The Regenerate SQL sub-property can be set to Yes, No or Default.
- Function Call—Calls the specified function. You can specify a function for update processing and a function for save processing.
- Nothing—No processing is done for the statement.

### Delete Handling

The following settings are available for deleting data in the transaction manager:

- SQL Statement Generation—(default) Generates a SQL statement based on property settings.
- Function Call—Calls the specified function. You can specify a function for delete processing and a function for save processing.

- Nothing—No processing is done for the statement.

## Server View Properties

The following properties define how the server view fetches data. A server view is either a single table view or a group of table views connected with server links. The setting for the Method property determines which sub-properties are displayed.

### Select Handling

Under Method, specify how the transaction manager will fetch data:

- SQL Statement Generation—(default) Generates a SQL statement based on property settings.
- Function Call—Calls the specified function. You can specify a function for `SELECT` processing and a function for `CONTINUE` processing.
- Nothing—No processing is done for the statement.

### Sort Widgets

Specify the sort order by entering a list of widget names and optional order specifiers. The order specifiers are `ASC`, for ascending (default), and `DESC`, for descending, and are case insensitive.

The SQL generator uses the widget name to determine the associated column or select expression to be sorted when generating a SQL `SELECT` statement.

### Distinct

Defines whether duplicate rows are included or omitted from query results.

- Yes—Includes the `DISTINCT` keyword in the automated SQL `SELECT` statement. This eliminates duplicate rows in the results.
- No—(default) Duplicate rows are included in query results.

Refer to “Eliminating Duplicate Rows in a Result Set” on page 33-13 in *Application Development Guide* for more information.

### Directions (`fetch_directions`)

Specifies whether the following commands are available in the transaction manager (in two-tier architecture only): `CONTINUE_BOTTOM`, `CONTINUE_DOWN`, `CONTINUE_TOP`, and `CONTINUE_UP`. If they are available, the transaction manager creates a continuation file for select (query) results



so that you can scroll forward and backward through the data. The following options are available:

- **Default**—The screen's Directions property is used to specify the value. If the screen's value is also Default, this is the equivalent of Down only-all modes.
- **None**—This eliminates the possibility of doing CONTINUE-command processing on a server view. Since CONTINUE functionality can consume system resources, you have better control or how a SELECT is issued against the table view.
- **Down Only-all modes**—Only the CONTINUE command fetches additional data. CONTINUE\_DOWN is not permitted.
- **Up/Down-view mode**—CONTINUE\_BOTTOM, CONTINUE\_DOWN, CONTINUE\_TOP, and CONTINUE\_UP are permitted in addition to CONTINUE if the current transaction mode is view.
- **Up/Down-all modes**—CONTINUE\_BOTTOM, CONTINUE\_DOWN, CONTINUE\_TOP, and CONTINUE\_UP are permitted in addition to CONTINUE if the current transaction mode is update or view. However, in update mode, data must be re-fetched after a SAVE command in order for updates to be displayed from the continuation file.

#### Count Select

Determines whether the transaction manager finds the size of the result set before fetching data.

- **Yes**—Determines whether there is a Warning message (yes or no). If Warning is set to Yes, you can specify the maximum size of the result set in the Threshold property.
- **No**—(default) The transaction manager fetches data regardless of the size of the result set.

## Service Properties

(For JetNet/Oracle Tuxedo only) The following properties define the type of transaction manager command that a service implements in a three-tier architecture. These Service properties are automatically set for the root table view on client screens and selection screens (if any) when you use the screen wizard to generate your screens.

If you specify these properties in the editor, they must only be set for the root table view. Check that the service component contains all needed widgets from the client screen for all server views.

**Select Service**

Specifies a service that implements a Select operation which retrieves information stored in a database table.

**Insert Service**

Specifies a service that implements an Insert operation which adds information to the database table.

**Update Service**

Specifies a service that implements an Update operation which allows you to make modifications to the data stored in the tables.

**Delete Service**

Specifies a service that implements a Delete operation which removes information from the database table.

## Setting Table View Properties to Generate SQL

Panther's SQL generator uses properties found under Database and Server View in the Properties window to generate SQL statements. Panther's transaction manager—via the SQL generator—can construct `SELECT`, `INSERT`, `UPDATE`, and `DELETE` statements using the property specifications for table views on the screen. To control or change the generated SQL, you can change the Database or Server View properties for widgets representing database columns, table views, or server views.

When you copy widgets from a repository entry that was created as the result of the import process, most of the properties that the SQL generator needs are set automatically. If you want to modify the SQL, you can change the property specifications. For more information about automatically generated SQL statements, refer to Chapter 33, “Using Automated SQL Generation,” in *Application Development Guide*.

## Creating and Linking a Table View

When you populate a screen with widgets from a repository that was created as the result of the import process, a table view is automatically copied to your application screen as well. You can have multiple table views on a screen—each representing a different database table.

However, you can create table views manually if you want to:

- Execute a self-join; that is, define a relationship that a table has with itself.
- Define a relationship between a database column in a table view and another database table that is not represented on your screen.

### How to Create a New Table View

1. Create (or copy) and name the widgets that will be members of the new table view if you have not already done so.

2. Select those widgets that are to be members of the table view.

**Note:** If any widget in your selection set is currently a member of another table view, it will be removed from that table view and added to the new one. In other words, a widget cannot be a member of two different table views.

3. Choose Edit→Group→Create→Table View.

The table view is created and currently selected. It is given a default name, `tview#`.

4. (Optional) Rename the table view to reflect its content (in the Name property under Identity in the Properties window).
5. To connect the new table view to the database, create a link and set its properties accordingly:
  - Choose Create→Link.
  - Edit the link's Transaction properties, specifically the Parent, Child and Relations properties (refer to “Editing Link Properties” on page 22-13 for details).
  - Identify in what order inserts, updates, and deletes should occur for the two table views represented in this link.

## Manipulating Table View Members

To display or identify those widgets that compose a table view on your screen, you can use either of two methods: select the table view or select one member of the table view.

### How to Identify the Members of a Specific Table View

1. Choose View→DB Interactions.
2. Select the table view from the DB Interactions window.
3. Choose Edit→Group→Select Members. All widgets that are members of the table view are selected.

### How to Identify the Table View Members Associated with a Specific Widget

1. Select at least one member of the table view.
2. Choose Edit→Group→Select Members. All widgets that are members of the table view are selected.

## Adding Table View Members

You can add members to an existing table view to:

- Include a widget in the SQL generation.
- Create a "virtual" column—that is, use a widget on the screen to display information from the database even though the widget does not have an associated column in the database.
- Add a database column back to the table view that you previously removed from the table view.

### How to Add Widgets to a Table View

1. Use one of the methods described above for displaying table view members.

2. With all members still selected, select the widget or widgets you want to add to the table view. The selection set should include the table view members and any new widgets. Remember to name the new widgets.
3. Choose Edit→Group→ Update Group Members. The table view is now redefined.  

If you omitted a widget that was originally a table view member, it will no longer be a part of any table view. Repeat the procedure to update the table view's membership.
4. (optional) To check which widgets make up the table view: With at least one member of the table view selected, choose Edit→Group→Select Members.
5. Choose OK.

## Identifying the Root Table View

The root table view determines the basis of the event processing occurring on a screen. In most cases, the transaction manager sets the root table view automatically. A root table specification must be set (to Default or to a table view name) if you want the transaction manager to perform the screen's transaction processing.

If the transaction manager is unable to determine the root table view for the screen, either because the Root property is set to None, or Default and the transaction manager is unable to determine the root table view, the following error message is displayed: "Root table view name not supplied or not valid." Make sure the Root property under Transaction for the screen specifies the correct table view.

In a master-detail screen, the database table which is the master would be the root table view.

## How to Identify A Table View As The Root Table For The Screen

1. Select the screen (no other widgets on the screen can be selected).
2. Specify the table view's name in the Root property under Transaction for the screen. The Setting field option menu lists the names of all table views on the selected screen. You can choose from the following options:
  - Default—When set to Default, the transaction manager automatically determines at runtime, based on the link properties, which of the table

views on the screen is the root table view. This is the initial setting for this property. However, if the default root table view is ambiguous, and the transaction manager is unable to determine the root table view, the following error messages are issued: `Root table view name not supplied or not valid` and `Transaction unspecified or unavailable`. Specify the root table view by name instead.

- None—The screen's transaction processing as performed by the transaction manager is turned off. Accessing transaction manager commands results in an error: `Transaction unspecified or unavailable`.
  - A table view name—Select one of the listed table views by name.
3. Choose OK.

---

## Using Links

---

A link defines the relationship between two table views. If a screen contains more than one table view, you need to have a link describing the relationship between the two tables.

When you import a database table to the repository, the import process automatically creates links on the repository entry that correspond to the foreign key definitions for that database table. When you copy database-derived widgets to your application screens, you can copy the links directly from the repository entry or create links on your screen.

## Creating Links

If a link does not exist for a pair of table views on your screen, you need to create it manually. The ability to join information from multiple database tables is based on the premise that a relationship exists between each of the represented tables on your screen. Therefore, for every pair of table views on your screen that will participate in

a join operation with each other, you need a link widget to define the relationship. Links have their own properties—you can define the type of link as well as the parent-child relationship of the table views.

Use the DB Interactions window to display a graphical representation of the relationships between multiple tables on your screen. The link properties defined for a screen apply only to a single screen. Therefore, on other application screens, you can rearrange the relationship between the tables.

You cannot have a cycle appearing in the link specifications. For example, if `link1` declares the `customers` table to be the parent and the `rentals` table to be the child, `link2` cannot have the `rentals` table be the parent and the `customers` be the child.

## How to Create a Link on Your Screen

1. Choose Create→Link. Click on the screen to place the widget.
2. Edit the link's Transaction properties, specifically the Parent, Child, Type of link, and Relations properties, if you want to join the table views.
3. Identify in what order inserts, updates, and deletes should occur in the database for the two table views represented in this link.

## Editing Link Properties

To access and edit link properties, select the link widget on your screen by doing any of the following:

- Click on it directly on your screen.
- Choose View→Widget List. Select the link from the list.
- Choose View→DB Interactions to display a graphical representation of your table views and links. Select the graphical image (an equal or less than sign) of the link from this window.

## Identity Properties

The following properties define the name of the link, the link's source of inheritance, and a location for additional comments as well as programmatic use:

Name

Use this property to specify the name of the link if you manually created the link. If you copied the link from a repository entry, a name is provided as the result of the import process. Its default name is in the form `K#table_name`. For example, the repository entry `titles` has a link widget named `K1titles`. If there was a second link on the entry, its name would be `K2titles`.

Inherit From

Use this property to specify the link's source of inheritance if you manually created the link. Use the format `repositoryEntry!linkName`. This value is provided automatically when a link is copied from a repository entry.

Memo Text

Attach up to nine separate lines of text (`memo1 . . . memo9`) for additional comments or for entering string expressions to use programmatically.

## Transaction Properties

The transaction manager uses the following properties to tell the SQL generator to build the appropriate SQL statement. Most of these properties obtain their values automatically via the import process.

Parent

Use this property to specify the parent table view. Panther gets this information during the import process from the primary and foreign key information in the database tables represented in the link. You can change this specification to change the relationship between the two table views.

Child

Use this property to specify the child table view. Panther gets this information during the import process from the primary and foreign key information in the database tables represented in the link. You can change this specification to change the relationship between the two table views.

Type

Use to specify the type of link.

- Sequential—Widgets belonging to the parent and child table views are populated by two separate SQL `SELECT` statements. The `SELECT` for the parent table view is performed first, followed by the `SELECT` statement for the child table view. The joining of the two table views



occurs because the results from the first `SELECT` statement provide the information needed by the second `SELECT` statement. Refer to “How to Join Two Tables with a One-to-Many Relationship” on page 22-17 for details on creating a sequential join.

For example, to find all the video rentals for a customer, you would specify a Sequential link type between the `customers` and `rentals` tables. One row in the `customers` table has the potential of having several associated rows in the `rentals` table. The processing for the `customers` table is done first, followed by the processing for the `rentals` table.

- **Server**—Widgets from both the parent and child table views are populated by a single SQL `SELECT` statement. Server links are possible when there is a one-to-one relationship between the rows in each of the table views. Refer to “How to Join Two Tables Using a One-to-One Relationship” on page 22-18 for details on creating a server join.

For example, to find the title for each video rented by a customer, you would specify a Server link type between the `rentals` and `titles` tables. For every rented video, there is an associated title; there is a one-to-one relationship between the `title_id` in the `rentals` table and the `title_id` in the `titles` table.

If the Type property is set to Server, there is a subcategory called Join Type which lets you take advantage of SQL join facilities. You can use this to control the join operation of any `SELECT` statement which combines information from two database tables. The four choices are:

- **Inner (default):** `PV_INNER`—Fetches all possible pairs, but excludes those rows that do not meet the matching column condition for the join.
- **Left Outer:** `PV_LEFT_OUTER`—Compares the two tables, fetches all possible pairs and also those rows from the left table having no matching value in the right database table (NULL values are used).
- **Right Outer:** `PV_RIGHT_OUTER`—Fetches all possible pairs, including unmatched rows from the rightmost database table using NULL values if no match is found in the left table.
- **Full Outer:** `PV_FULL_OUTER`—Fetches rows from both tables even if when there is no (NULL) match.

For an example using the various join types, refer to “Specifying the Join Type” on page 33-28 in *Application Development Guide*.

#### Relations

The Relations property opens the Relations dialog box where you define how the two table views represented in the link are related and how they are used by the SQL generator, that is, what are the columns in the two tables that are related. Refer to page 22-17, “Joining Database Tables.”

In addition to defining the join condition in a `WHERE` clause, you can also define a lookup specification whereby the data entered by the user at runtime can be used to fetch database column information when new or updated records are added to the database (refer to page 22-20, “How to Define a Lookup Specification for Either Link Type”).

#### Insert Order

Use this property to specify which table view receives the `INSERT` statement first: Parent First (default) or Child First. SQL statements are generated for both the parent table and the child table.

#### Update Order

Use this property to specify which table view receives the `UPDATE` statement first, Parent First or Child First (default). SQL statements are generated for both the parent table and the child table.

#### Delete Order

Use this property to specify which table view will receive the `DELETE` statement first, Parent First or Child First (default). SQL statements are generated for both the parent table and the child table.

## Service Properties

(JetNet/Oracle Tuxedo only) The following property defines which service the transaction manager uses to implement a Validation operation (in a three-tier architecture). This property is automatically set for link widgets on client screens when you use the screen wizard to generate your screens.

#### Validation Service

Specifies a service to implement a Validation operation that allows you to verify whether data entered in New or Update mode exists in the linked database table.

## Joining Database Tables

In general, joins are built by comparing pairs of columns from two joined tables by testing the data from both columns for equality or other comparisons. Sometimes these joins have a one-to-one relationship while others have a one-to-many relationship. Most common multi-table queries use parent/child relationships created by primary keys and foreign keys. Panther lets you define these relationships by letting you define the join type and the relationship between table views.

### How to Join Two Tables with a One-to-Many Relationship

Joining two tables with a one-to-many relationship creates a sequential join:

1. Select the link widget. Under Transaction, specify Sequential in the Type property.
2. Select the Relations property to define the join relationship. The Relations dialog box opens.
3. Enter or modify the Parent and Child column names. The column names are case-sensitive. Choose Help to display and select from a list of columns associated with Parent and/or Child tables.

When the SQL statement is generated, the `WHERE` clause associated with the Child table's column uses the value in the Parent table's column.

4. (Optional) If the Parent's column is represented on the screen by two different widgets, enter the preferred widget's name instead using the following format (including the square brackets and the literal `+0`):

```
::widget_name[+0]
```

5. Define the type of relationship—`join`—in the Rel (center) column of the Relations dialog box. The only relationship that can be specified for a join is the word `join`.

For each join relation specified, the `WHERE` clause will include one expression of the form:

```
widget_data_in_parent_tableview = child_table.child_column
```

6. Choose OK to save the specifications and close the Relations dialog box.

## How to Join Two Tables Using a One-to-One Relationship

Joining two tables with a one-to-one relationship creates a server join, which displays multiple records using a single condition.

1. Select the link widget. Under Transaction, specify Server in the Type property.
2. If the database engine supports outer joins, specify the appropriate Join Type: Left Outer, Right Outer, or Full Outer (the default type is Inner). For more information about the Join Type property, refer to “Specifying the Join Type” [on page 33-28](#) in *Application Development Guide*.
3. Select the Relations property to define the join relationship. The Relations dialog box opens.
4. Enter or modify the Parent and Child column names. The column names are case-sensitive. Choose Help to display and select from a list of column names associated with the Parent and/or Child tables.

The columns you specify are used to build the SQL join condition in a `WHERE` clause.

5. Define the type of relationship—`join`—in the Rel (center) column of the Relations dialog box. The only relationship that can be specified for joins is the word `join`.

For each join relation specified, the `WHERE` clause will include one expression of the form:

```
parent_table.parent_column = child_table.child_column
```



**Figure 22-1** The Relations dialog box lists the parent and child relationships for the selected link.

If there are multiple joined columns, then the expressions are connected by the keyword `AND`. If more than one table view in the server join represents the same database table, the SQL generator automatically supplies table alias names as needed. Therefore, self-join expressions are automatically handled.

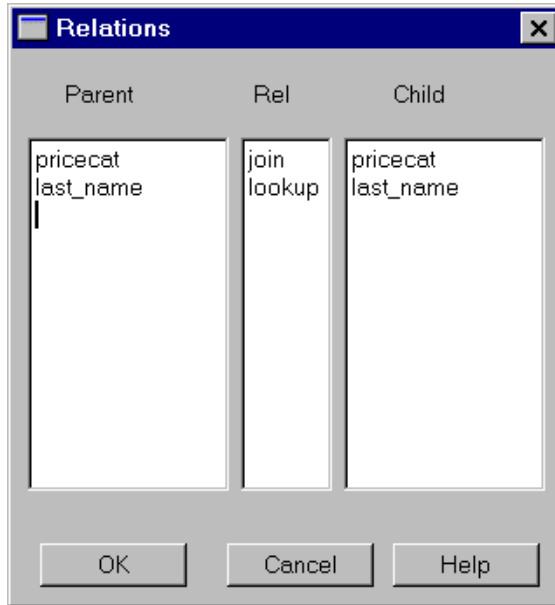
6. Choose OK to save the specifications and close the Relations dialog box.

Optionally, you can look up a Child column in the database based on a value the user enters into the widget associated with the Parent column. The lookup is based on the specified join relationship that you defined in steps 3 and 4 for server or sequential joins.

**Note:** Lookup specifications are used only when adding (inserting) and updating records.

## How to Define a Lookup Specification for Either Link Type

1. Under Transaction, select the Relations property. The Relations dialog box appears.
2. Enter the Parent column name under Parent in the Relations dialog box.  
  
If the parent column is represented on the screen by two different widgets, enter the widget's name instead using the following format (including the square brackets and the literal +0):  
  
`::widget_name[+0]`
3. Enter the keyword `lookup` in the center column of the Relations dialog box.
4. Enter the Child column name under Child in the Relations dialog box.



**Figure 22-2** This specification takes the customer id entered by the user to find the corresponding last name in the database table.

5. Choose OK to save the specifications and close the Relations dialog box.

## Setting Validation Links

The links that are defined for a screen can be used to specify validation links. When a validation link exists, you can enter a value in a field, in either New or Update mode, and the transaction manager looks up that value in the linked database table. If the value exists, it displays data for any widgets in the child table view. If the value does not exist, it displays the error `Invalid Entry`.

### How to Specify a Validation Link

1. Create the desired link if it does not exist and name the link.
2. Select the text widget that requires the validation link.
3. Under Database, set the Validation Link property to the desired link.

The parent table view in the link contains the text widget. The database table associated with the child table view contains the pre-existing values.

Validation link processing is only performed in New and Update modes, as part of the `NEW`, `COPY` or `SELECT` commands, when you are entering or updating data. Since the data in the child table view should already be entered in the database, set the child table view in a validation link to be non-updatable.







# Part IV Editors

Styles Editor

JIF Editor

Menu Bar Editor



# 23 Styles Editor

An essential component of a well-designed application is consistency in presentation and behavior. This consistency helps users learn and use the application more quickly and more reliably. Not surprisingly, the larger and more complicated the project, the greater the need and the difficulty of maintaining this consistency. The styles editor addresses this issue. It allows you to specify globally how an application will appear and behave when it uses the transaction manager commands.

The transaction manager applies a set of default classes and styles to your application screens. The styles editor is used to edit these default settings or to define new styles and classes.

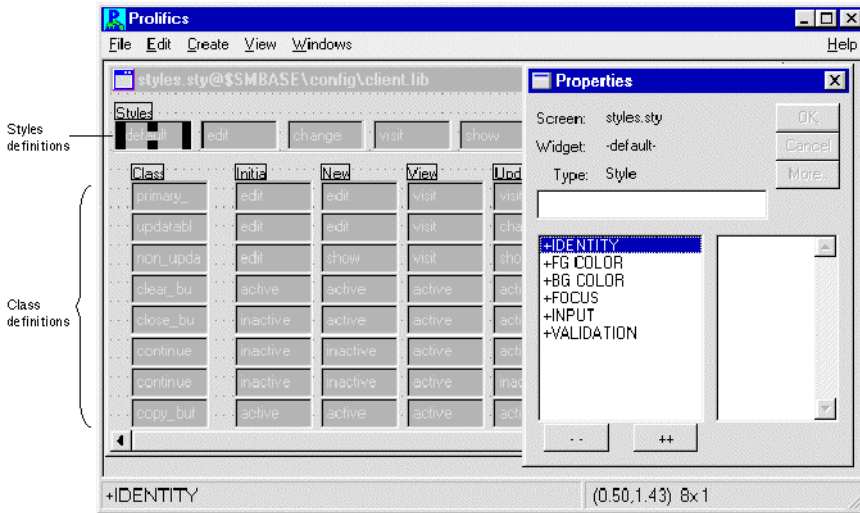
This chapter explains how the default styles and classes are applied as well as how to create a new style or class.

---

## Starting the Styles Editor

---

Invoke the styles editor by choosing Tools→Styles Editor. On opening the styles editor, Panther looks in all open libraries for the binary file called `styles.sty` (Figure 23-1 illustrates Panther's default styles file) and opens it. If the file isn't found, a new, untitled styles file opens. The `styles.sty` is distributed in the client library.



**Figure 23-1** The styles editor opens the default styles.sty file.

The styles editor provides a workspace with up to five components:

#### Styles Editor window

A workspace area is divided into two parts: style definitions and class definitions, one for each transaction mode. Generally, you create new styles first. Then, you create a new class and assign the appropriate style for each transaction mode. You can have more than one working styles file open at once; however, an application can access only one styles file.

#### Styles Editor menu

Provides options to open and create style files and libraries, save style files in libraries, maintain style files under source control management, edit styles and classes, and view the properties set for the selected style.

#### Properties window

Displays the properties applicable to a selected style or class.

#### Widget List

Displays the styles, classes, and the styles associated with each transaction mode of the current styles file.

---

# Transaction Modes

---

When an application calls a Panther transaction manager command, the transaction manager checks the current transaction mode and either changes the mode or reports an error if the current mode is invalid for the command. Since you can define a different style for each mode, the accessibility or appearance of widgets in your application screen can change as you change transaction modes.

Each transaction mode is represented in the `styles.sty` file. The transaction manager supports the following modes:

`new`

Mode where the user can enter new data. Saving new data typically generates SQL `INSERT` statements.

`update`

Mode where the application has fetched data to be edited. Update mode distinguishes between two types of data: fetched data and new data. For example, if an application is updating a sales order, it fetches the order information and permits updates to the existing data. It might also allow the user to enter new data, such as adding additional items to the order. The styles editor refers to fetched data as “update occ” data. All other transaction data is considered “update” data.

Before fetching data, Panther applies the `update` styles to the widgets. After occurrences containing data are fetched to a widget, Panther applies the `update_occ` style to the widget.

Saving updated data typically generates SQL `UPDATE` statements for altered data, SQL `DELETE` statements for deleted data, and SQL `INSERT` statements for new data.

`view`

Mode where the application has fetched data and prevents the user from making any changes to it. Styles are applied before data is fetched.

`initial`

Mode associated with a newly opened screen or a closed transaction.

qbe

Mode reserved for a future release.

## How Transaction Manager Commands Affect Modes

The following transaction manager commands change the current transaction mode, thus reflecting the state of the transaction:

Command	Changes mode to:
<code>CLOSE</code>	initial
<code>COPY</code>	new
<code>NEW</code>	new
<code>SELECT</code>	update
<code>VIEW</code>	view

When an application calls one of these transaction manager commands, by way of a push button, menu item, or function key execution, the transaction manager checks the Class property (set in the screen editor Properties window under Transaction) of every onscreen widget and applies the appropriate style for the mode.

The most common style change is protection edits for data entry type widgets or availability status for menu items and push buttons. For example, if the `NEW` command is selected, the application needs to permit input in some or all data entry widgets on a screen, make a “save” button active, and gray out (deactivate) the “continue” button. On the other hand, if the `VIEW` command is selected, the application needs to protect some or all data entry widgets from input and make a “save” button inactive.

Some commands do not change the current transaction manager mode but require a particular current mode:

Command	Requires mode:
<code>CONTINUE</code>	view or update
<code>SAVE</code>	new or update

Since the transaction mode does not change for these commands, no style changes are made. Whatever styles were set by the required mode (such as `view`, `update`, or `new`) remain in effect. Therefore, when the `NEW` transaction manager command is selected, the user can enter data in the unprotected fields and choose the save button. If the save button calls `sm_tm_command( "SAVE" )`, the data is saved to the database. Until the application calls one of the commands which changes the transaction mode, the screen's data entry fields permit input and the save button remains active.

## Using Styles and Classes

Each widget on your application screen has a `Class` property setting (under `Transaction` in the screen editor's `Properties` window). Each menu item has a `TM Class` property. This property determines how the widget or menu item behaves in each of the transaction manager modes. A class is a matrix of styles and modes. A style is a set of properties that are imposed on the widget.

For example, if you create the following styles:

- `view`—Foreground Color = Red; Focus Protect = Yes
- `edit`—Foreground Color = Blue; Focus Protect = No; Input Protect = No

You can then create a class and set a style for each mode:

Transaction Class	Initial	New	View	Update Occ	Update
<code>test</code>	<code>edit</code>	<code>edit</code>	<code>view</code>	<code>edit</code>	<code>edit</code>

You can now set a widget's `Class` property to `test`. Then, when the screen is in initial, new, and update modes, the widget with a `Class` property setting of `test` has a blue foreground and permits focus and keyboard input. In view mode, however, the same widget has a red foreground and does not permit focus.

## Using the Default Styles Settings

Panther has a series of predefined styles and classes. Initially, Panther uses its internal styles to define the default classes. The internal styles set protections only. Table 23-1 lists the internal styles and their property settings that are used text widgets.

**Table 23-1 Predefined styles and their associated property settings**

<b>Style</b>	<b>Property settings</b>
change	allow focus allow input allow clearing allow validation
edit	allow focus allow input allow clearing allow validation
show	prevent focus prevent input prevent clearing allow validation
visit	allow focus prevent input prevent clearing allow validation

The styles editor also supports the values Default and None. When a style property is set to Default, Panther determines the value at runtime based on the value saved in the screen editor.

## Using the Default Transaction Classes

You are not required to define transaction classes for an application; your application can use the default classes supplied by Panther or not use any classes.



When a widget's transaction class is set to Default, Panther determines the widget's class at runtime based on the properties set in the screen editor for the widget and for the widget's table view.

There are three default classes:

- `non_updatable`
- `primary_key`
- `updatable`

If a widget is a member of a non-updatable table view, its default Class property is `non_updatable`.

If a widget is in an updatable table view and it is part of the table view's primary key, its default Class property is `primary_key`.

If a widget is in an updatable table view and is not a part of the table view's primary key, its default Class property is `updatable`.

These default Class properties are assigned the following styles in each of the respective transaction modes:

<b>Transaction manager class</b>	<b>Initial</b>	<b>New</b>	<b>View</b>	<b>Update occ</b>	<b>Update</b>
<code>non_updatable</code>	<code>edit</code>	<code>show</code>	<code>visit</code>	<code>show</code>	<code>show</code>
<code>primary_key</code>	<code>edit</code>	<code>edit</code>	<code>visit</code>	<code>visit</code>	<code>edit</code>
<code>updatable</code>	<code>edit</code>	<code>edit</code>	<code>visit</code>	<code>change</code>	<code>edit</code>

## Setting Classes for Menu Items and Push Buttons

There are predefined classes that you can apply to push buttons and menu bar items on your application screens. However, they are not applied when the Class property (for buttons) and TM Class property (for menu items) are set to Default. You must explicitly set the Class property under Transaction in the screen editor for each push button, and the TM Class property in the menu bar editor for each menu item. The following table lists the predefined classes and their active/inactive setting in each mode.

<b>Transaction manager class</b>	<b>Initial</b>	<b>New</b>	<b>View</b>	<b>Update and Update_occ</b>
clear_button	active	active	active	active
close_button	inactive	active	active	active
continue_button	inactive	inactive	active	active
continue_view_button	inactive	inactive	active	inactive
copy_button	active	active	active	active
delete_button	inactive	inactive	inactive	active
new_button	active	inactive	active	inactive
save_button	inactive	active	inactive	active
save_immed_button	active	active	inactive	active
view_button	active	inactive	active	inactive
view_once_button	active	inactive	inactive	inactive

The styles for each of these classes make the push button or menu bar active or inactive according to the current mode.

---

## Defining a New Transaction Style

---

To create a new style, choose Create→Style. A style widget is created in the styles editor window under the Styles heading. With the widget selected, you can access its style properties in the Properties window.

You can also copy style widgets from other style files that are currently open in workspace. Use the Copy and Paste options on the Edit menu.

## Style Widget Properties

The Properties window in the styles editor supports a subset of properties available in the screen editor. In addition to the values available in the screen editor, the styles editor also supports the value Default. When you set a style property to Default, Panther determines the value at runtime based on the value defined and saved with the individual widgets in the screen editor.

## Identity Properties

Identity properties include the following:

### Name

Any valid Panther identifier. By default, the styles editor assigns a name to each new style in the form `style#x`, where `x` is a number. Consider renaming the style with a more descriptive name.

### Button/Menu Status

Determines the button or menu status. Select one of the following values:

- Active—The button or menu is ungrayed. When the user chooses an active button or menu, the control string associated with the object is executed.
- Inactive—The button or menu is grayed. The object cannot get focus, and therefore the associated control string cannot be executed.
- Default—The active/inactive status is derived from the setting saved with the screen.

### Function

Can be either a JPL or prototyped C function (s,i,i), if you want to change widget properties that are not available in the styles editor. The function is called after Panther has applied the other properties defined by the style. The function is passed the following parameters:

- Style name.
- An integer which identifies the current mode. The possible settings are:

0	initial
---	---------

1	new
2	view
3	update occ
4	update

- Field number where style was applied.
- Field occurrence number if the transaction mode is `update occ`; otherwise, it is set to 0.

For menu options, the function is passed the following parameters (s,i,s,i): style name, transaction mode, menu name, and menu item.

## Color Properties

The color and display attributes under the color properties correspond to Panther's basic color specifications. Setting foreground and/or background colors in the styles editor overrides any basic, scheme, or extended color assigned to a widget on your application screen. Color changes can provide users with a visual cue that a field is protected. For instance, if the screen is in update mode, primary keys can be displayed with white text (foreground) on a blue background, while updatable fields can be displayed with blue text on a white background.

## Focus Properties

This category lists the following property:

### Focus Protection

Determines if the widget allows focus. Set the property to **Yes** to prevent the user from tabbing or clicking in the widget. Set it to **No** to allow the widget to get focus. Set the property to **Default** to leave the widget's current focus protection unchanged.

## Input Properties

Input properties include the following:

### Keystroke Filter

Determines what keystroke filters are placed on the user's keystrokes. If you set this property to Default, the widget's filter is not changed from the value set in the screen editor.

Even though you can set the enforcement of a filter that you have defined in the screen editor, you cannot change the edit mask string or regular expression edit in the styles editor. In order to temporarily change these values, you can redefine the property specification programmatically.

For example, in the screen editor, the widget `order_num` has the edit mask specification of `\A\A-\9\9\9\9` and its Class property set to `criteria`. The `criteria` class has the following styles:

Initial mode	New mode
Foreground=Blue	Foreground=Cyan
Keystroke Filter=Unfiltered	Keystroke Filter=Default

In initial mode, the `order_num` widget has a blue foreground and permits any keystrokes. In new mode, `order_num` has a cyan foreground. Since the Keystroke Filter for new mode is set to Default, Panther restores the original keystroke filter, `\A\A-\9\9\9\9`.

For descriptions of keystroke filters, refer to [page 14-2](#), “Defining the Input with Keystroke Filters.”

### Required

Determines if the widget requires data in order to pass validation. Set the property to Yes to force the user to enter data in the widget. Set it to No to allow the widget to remain empty. Set the property to Default to leave the widget's current setting unchanged.

### Input Protection

Determines if the widget permits data entry. Set the property to Yes to prevent the user from entering data in the widget. Set it to No to allow the user to enter data. Set the property to Default to leave the widget's current input protection unchanged.

#### Clearing Protect

Determines if the user can clear the widget's contents by pressing the CLR (clear all) or FERA (clear field) keys. Set the property to Yes to prevent the widget's contents from being cleared. Set it to No to allow clearing. Set the property to Default to leave the widget's current setting unchanged.

## Validation Property

This category has the following property:

#### No Validation

Determines if Panther performs validation on the widget. Specify Yes to prevent validation. Specify No to allow validation. Set the property to Default to have the widget's protection edit unchanged.

---

# Creating a New Transaction Class

---

To create a new class, choose Create→Class. A new row in the class section of the styles editor window is created.

1. Select the class widget (first in the row under the Class column) to access its properties.
2. Assign a name to the class; this can be any valid Panther identifier.
3. For each of the modes, select the corresponding widget and assign a style name. The setting field in the Properties window will list all of the available styles for the current `styles.sty` file. If you do not want to assign a style name for a particular mode, set it to None. If you select a group of mode widgets, you can assign them the same style.

**Note:** QBE mode is reserved for a future release.

4. If you are defining a class for a menu item or push button, specify the same style for `update` and `update occ` modes.

A class should use a set of compatible styles. Therefore, for best results, the styles for a particular class should set the same properties. For example, if the new mode style sets the Foreground Color Name to Blue, the other styles in the same class should also set Foreground to a non-default value.

---

## Testing Styles

---

With a style widget selected, choose File→Test Style to view and test the properties you assigned to the style widget. The Test Style screen displays one of each Panther widget type, and illustrates how the style's properties will affect those widgets at runtime.

While in test mode, you can enter data in the data entry widgets, choose the selection widgets to test either the current color or protection settings, or check whether the appropriate widgets are inactive or active given the selected style. In addition, a sample menu bar is provided so that you can determine the status of menu items.

To change any of the settings, exit test mode by choosing Close from the Test Style window's system menu.

---

## Saving a Styles File

---

Although the styles editor allows you to save your styles file under any name, the file must be named `styles.sty` to be accessed by your application.

The styles editor has two save options available on the File menu:

Save

Saves the `styles.sty` file (in binary format) to its source in a library. If the file is new, Panther opens the Save As Library Member dialog box where you

can provide a name and select the library in which to save it. In general, it is recommended that `styles.sty` reside in the client library.

### Save As

Opens the Save As Library Member dialog box where you can save the file to a library with a new name or to a different library. The file is saved in binary format (with the default `.sty` extension).



# 24 JIF Editor

For JetNet and Oracle Tuxedo applications, the JIF is a file that contains information about services and service groups. For Oracle Tuxedo applications, the JIF can also contain information about reliable queues. The JIF is a central repository of all such information and is accessed by Panther to determine the requirements and specifications of services and queues. The JIF is stored in the distributed common library and is accessible to the team.

The JIF is a critical component of your JetNet and Oracle Tuxedo applications. It is accessed:

- When a client makes a service call to determine the parameter types, number, and direction expected by a service, and to verify the validity of the service call.
- When a server needs to determine what code should be executed to process a service call, that is, which JPL or C procedure maps to the service.
- When the server needs information regarding a service's return data and parameters, and to identify the service component associated with the service.
- When a server advertises its services.
- When a service forwards data to another service, when a service receives data, and when data is returned to a client from a service.
- (Oracle Tuxedo only) When a message queuing request is made, to identify the type of reliable queue and the expected data type of the message (if the queue is associated with a service), and the names and types of the data if the queue is independent, that is, not associated with a service.

If you use the screen wizard to create your client screens and service components, the wizard specifies service information on these screens. However, if you create your screens outside of the screen wizard, you need to specify the service information on

the appropriate screens. In both instances you need to define the services in the JIF. The information provided in the JIF must match the information in the client screens and service components associated with a particular service.

You need to create a JIF or update it to define services, service groups, or queues. This chapter shows you how to create and edit a JIF using the graphical JIF editor.

The utility `jif2asc` is provided to allow you to convert a JIF between binary and ASCII formats.

---

## Starting the JIF Editor

---

The JIF editor is an interactive tool used to create a new JIF or edit an existing one. You can invoke the JIF editor by choosing Tools→JIF Editor, or you can follow the directions for your environment:

### How to Start the JIF Editor

- In Windows:

Double-click on the JIF editor icon.

- In Motif and Character Mode:

Type the following at the command line:

```
jifedit [jif-file]
```

Specify *jif-file* to start up the editor with an existing JIF loaded.

**Note:** If no *jif-file* is specified, Panther opens the JIF specified by the setup variable `SMTPJIF`. If you invoke the editor with no options, you can load a JIF file from the editor File menu.

# The JIF Editor Workspace

At startup, the JIF editor workspace consists of three components: the View Services screen containing a list of services, a menu bar containing six pulldown menus (Panther) and seven pulldown menus (Oracle Tuxedo only), Help, and a toolbar.

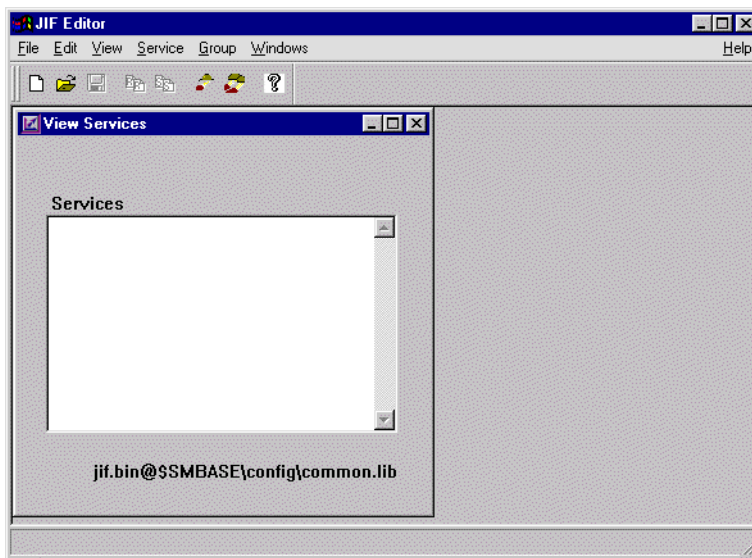


Figure 24-1 The JIF editor workspace: the View Services screen, menu bar, and toolbar.

## View Services Screen

The View Services screen displays the `jifname@current_library` name at the base of the screen. If you have invoked the editor naming an existing JIF, or if `SMTPJIF` is set to a JIF, the services appear in the list box.

## Menu Bar

The JIF editor provides you with easy access to the operations and commands that help you define a JIF. This section provides a brief overview of each menu option and its contents.

### File menu

File menu commands include:

- **New**—Creates a new JIF, a new library, or a new remote library.
- **Open**—Opens a JIF, a library, or a remote library. You can also connect to the middleware API if you are not already connected to it.
- **Close**—Closes a library. You can also disconnect from the middleware API.
- **Remove**—Deletes a JIF in a library.
- **Save**—Saves the JIF with its current name. If the JIF is untitled, the Save As dialog box opens and you can save it with a name to a library. JIFs are by default saved in binary format. For more information on converting JIFs between ASCII and binary formats, refer to [jif2asc on page A-19](#) in *Application Development Guide*.
- **Save As**—Saves the JIF under a new name in a library.
- **Revert**—Reverts to the last saved version of the JIF. An untitled JIF reverts to an empty JIF.
- **Source Mgmt**—Provides access to your installed source code management tool (support for SCCS and PVCS). This option is only available if an open library is under source control management. Choose this option to check-in or check-out a JIF from a library. You can also choose to cancel the check-out request. The JIF is made read-only, and the lock on the JIF is released so other users can check out the JIF for editing purposes. For further information on source control management, refer to “Maintaining Libraries Under Source Control” [on page 10-4](#) in *Application Development Guide*.
- **Exit**—Ends your session in the JIF editor. You are prompted to save the JIF that you created or changed but did not save.

#### Edit menu

Edit menu commands let you generate service code output, written to a Motif or Windows clipboard, or a file in character mode.

#### View menu

View menu options permit you to access lists of defined services, service groups, and queues in the JIF. You can toggle the options on and off to display or hide a particular screen. The `jifname@current_library` name is displayed at the base of each of the View screens. The View menu option screens provide a starting point for the update procedures of services, service groups, and queues.

#### Service menu

Service menu commands let you create, update or delete a service.

#### Group menu

Group menu commands allow you to create, update or delete a service group.

#### Queue menu

(Oracle Tuxedo only) Queue menu commands let you create, update or delete a queue.

#### Windows menu

Windows menu commands provide access to all open windows in your screen editor workspace. Select an item from the list to bring focus to that window.

#### Help menu

Help option provides general overview help or help on the currently active window.

---

## **Defining and Updating Services**

---

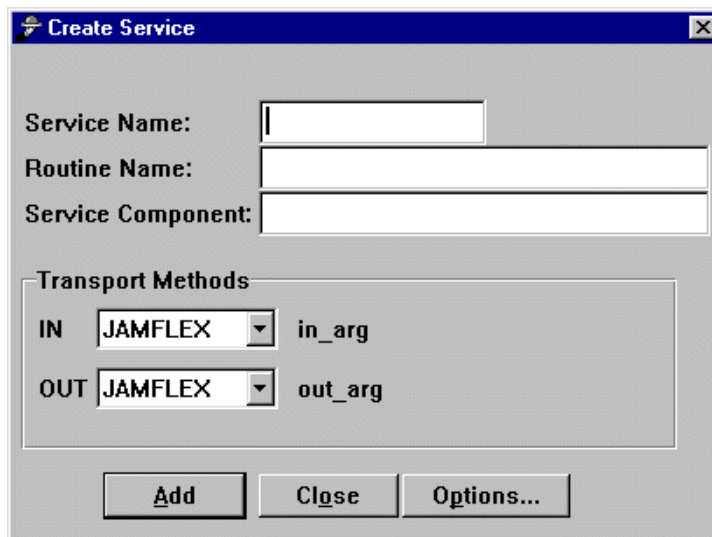
You can specify service information on the client screens and service components via the screen wizard or independent of the wizard. In both instances, you use the Create Service screen to define new services. On the Create Service screen, you enter and select data that describe a service. The service description and the service options in the JIF must match the information in the client screens and service components

associated with the described service. These services can later be included in service groups. For further information on service groups, refer to [page 24-12](#), “Defining and Updating Service Groups.”

## How to Define or Update a Service

1. Depending on the desired task, do either of the following:
  - To define a new service, choose Service→Create (or the Create Service button on the toolbar). The Create Service screen opens.
  - To update an existing service, choose View→Services. The View Services screen opens. Select the service that you want to edit and choose Service→Update, or double-click on the desired service. The Update Service screen opens.

**Note:** The Update Service screen contains information for the specified service. It is identical to the Create Service screen except for its title bar.



The screenshot shows a dialog box titled "Create Service". It has a title bar with a close button. The main area contains three text input fields labeled "Service Name:", "Routine Name:", and "Service Component:". Below these is a section titled "Transport Methods" which contains two rows of input. The first row is "IN JAMFLEX in\_arg" and the second is "OUT JAMFLEX out\_arg", where "JAMFLEX" is in a dropdown menu. At the bottom of the dialog are three buttons: "Add", "Close", and "Options..."

**Figure 24-2** Define a service with the Create Service screen.

2. Enter or modify the name of the service in the Service Name field. Each service name must be unique in the JIF and can be no greater than 15 characters.

If you used the screen wizard to generate your screens, the service name includes a suffix that indicates the particular transaction manager operation that a service implements. In this case, when you tab into the Routine Name field or the Service Component field, the JIF editor automatically enters the appropriate information into both these fields.

**Table 24-1 JIF defaults for screens generated by the screen wizard**

Operation type	Service Name	Routine name	Service component
Select	<i>servicePrefix_s</i>	select	<i>servicePrefix</i>
Insert	<i>servicePrefix_i</i>	insert	<i>servicePrefix</i>
Update	<i>servicePrefix_u</i>	update	<i>servicePrefix</i>
Delete	<i>servicePrefix_d</i>	delete	<i>servicePrefix</i>
Link_addtnlTable	<i>servicePrefix_ll</i>	val_link	<i>servicePrefix</i>
Choose_addtnlTable	<i>servicePrefix_cl</i>	choose	<i>servicePrefix</i>

If you created your screens without the screen wizard, the service name should not include the suffix that indicates the transaction manager service. In this case, when you tab into the Routine Name field or the Service Component field, the JIF editor automatically duplicates the name of the service in both the fields. To enter a different name, edit the text in the appropriate field.

3. Tab to the Routine Name field. Enter or modify the name of the JPL module or C routine which contains the service code that implements the service. Each routine name can be no greater than 31 characters.

**Note:** If you used the screen wizard to generate your screens, you will find the appropriate routine name in this field.

4. Tab to the Service Component field. Enter or modify the name of the service component associated with the service.

**Note:** If you used the screen wizard to generate your screens, you will find the appropriate service component name in this field.

5. Define the transport methods. Select or change the buffer type that is used to transport data to and from the service. The direction of the flow of information between the client and the server for the service is as follows:

- IN corresponds to incoming data to the service.
- OUT corresponds to data returned from the service to the client.

Transport methods are initialized to `JAMFLEX`, the default datatype. The available IN/OUT buffer types are as follows:

- JetNet—Choose either `JAMFLEX` or, if no data is required, `none`.
- Oracle Tuxedo—Choose `JAMFLEX`, `FML`, `FML32`, `STRING`, or, if no data is required, `none`. If you use a `STRING` buffer type, a field appears in which you must enter the name of the service parameter as it is defined in the service code.

6. Choose Add.

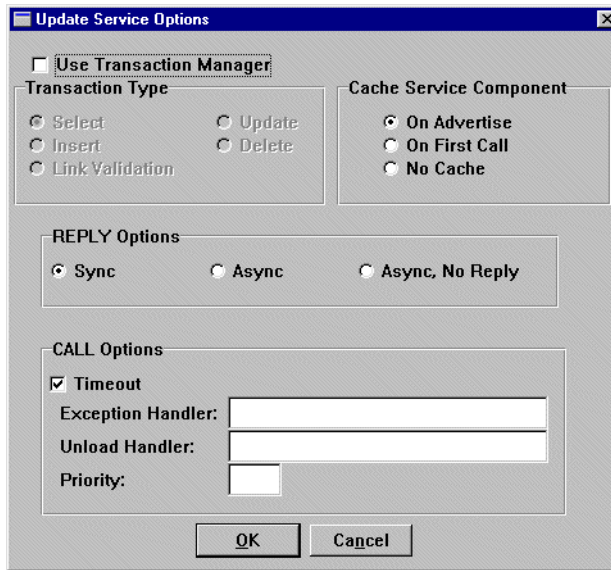
The View Services screen is automatically updated as services are defined or modified during an editing session.

## Controlling Service Behavior

The service options allow you to control which transaction a service implements, when a service component is opened and cached to memory, and the behavior of the service itself when it is called.

To specify service options, choose the Options button on the Create Services screen or the Update Services screen, as the case may be. The Update Service Options screen opens.





**Figure 24-3** Control service behavior with the Update Service Options screen.

You can control the behavior of services by specifying service options on the Update Service Options screen. The options are as follows:

## Transaction Type

The transaction options determine the type of transaction manager operation that a service implements. To specify the transaction type, choose the Use Transaction Manager check box. The transaction type options are activated.

The default is set to `select`. The options are as follows:

- `Select`
- `Insert`
- `Update`
- `Delete`
- `Link Validation`

**Note:** If you used the screen wizard to create your screen, the corresponding transaction type is automatically set based on the service name.

## Cache Service Component

The options control the relationship between a service and when its service component gets opened, closed, and cached to memory. The default is set to On Advertise. The options are as follows:

### On Advertise

Opens a service component when a service is advertised and caches the service component to memory. When the service is called, the service component gets selected; the service component gets deselected after the service returns.

### On First Call

Opens the service component when a service is first called and caches the service component to memory. If the service component is already cached in memory, this option selects the service component when the service is called.

### No Cache

Opens the service component for the duration of the service and closes it when the service completes. The container is not cached.

## Reply Options

The reply options determine whether or not the sender of the request waits for a reply. The default is set to Sync. The options are as follows:

### Sync

A synchronous service call is when the sender of the request waits for the reply (when the service call command returns after the service being called completes).

### Async

An asynchronous service call is when the sender of the request does not wait for the reply (when the service call command returns before the service being called completes).

### Async, No Reply

An asynchronous service call with no reply is when the sender of the request does not expect a reply (when the service call command returns before the service being called completes and there is no reply for the service).

## Call Options

The call options determine the order in which services are processed, the time taken for services to be processed, where the service call is executed, which handler should be used, and what priority a service should hold. You can set or unset the following call options:

### Timeout

Select to specify that a service call on encountering a blocking situation should be subject to the default blocking timeout of 60 seconds (default). Deselect to specify that a service call on entering a blocking situation can wait indefinitely and is not subject to the blocking timeout. A blocking timeout specifies how much time elapses before a service call times out and returns to its caller. However, in Oracle Tuxedo applications, transaction timeouts remain in effect. For further information on setting the default blocking timeout, refer to Default Blocking Timeout [on page 3-12](#) in *JetNet/Oracle Tuxedo Guide*.

### Outside Transaction

In Oracle Tuxedo applications, select to specify that the service call should be executed independently of the active transaction, if one exists. Deselect to specify that the service call be executed within the active transaction (default).

### Exception Handler

Specify the name of an exception handler (either a Panther variable or a string) to handle any exceptions that might occur as a result of either sending the request or receiving the response. If you do not specify an exception handler, the default is the application-wide exception handler.

In Oracle Tuxedo applications, if you do not specify an exception handler, the default is the transaction handler. However, if a transaction handler is not specified when a transaction is active, then the default is the application-wide handler. For more information on exception events and handlers, refer to “Exception Events” [on page 6-11](#) in *JetNet/Oracle Tuxedo Guide*.

#### Unload Handler

Specify the name of an unload handler (either a Panther variable or a string) to handle any unload events that might occur as a result of receiving the response from the service. If you do not specify an unload handler, the default is the application-wide unload handler.

In Oracle Tuxedo applications, if you do not specify an unload handler, the default is the transaction handler. However, if a transaction handler is not specified when a transaction is active, then the default is the application-wide handler. For more information, refer to “Unload Handlers” on page 6-29 in *JetNet/Oracle Tuxedo Guide*.

#### Priority

Establishes a priority that determines the order in which service calls get carried out. The value can be an absolute (unsigned) integer between 1 and 100, inclusive (default is 50). The higher the value, the greater the priority of the service call.

## How to Delete a Service

1. Choose View→Services. The View Services screen opens.
2. Select the service and choose Service→Delete.

The View Services screen is automatically updated as services are deleted during an editing session.

---

# Defining and Updating Service Groups

---

A service group is a collection of services; services can be grouped in several different ways. Services are grouped to facilitate advertising services in servers. Services contained within a unique server can be defined as a service group. In addition, services can also be grouped into multiple service groups within one server. If you

have multiple service groups in a server, it should be a logical grouping based on the application's tasks. For example, in a banking application, you might want to group all the services that can be accessed from an ATM client.

You use the Create Group screen to create groups and to include services in these groups. You can create service groups without any services in them; however, you can add existing services to these service groups. For further information, refer to [page 24-5](#), “Defining and Updating Services.”

## How to Define or Update a Service Group with Services

1. Depending on the desired task, do either of the following:
  - To define a new service group, choose Group→Create (or the Create Group button on the toolbar). The Create Group screen opens.
  - To update a service group, choose View→Groups. The View Groups screen opens. Select the service group that you want to edit and choose Group→Update, or double-click on the desired service group. The Update Group screen opens.

**Note:** The Update Group screen displays the services currently in the group. It is identical to the Create Group screen except for its title bar.

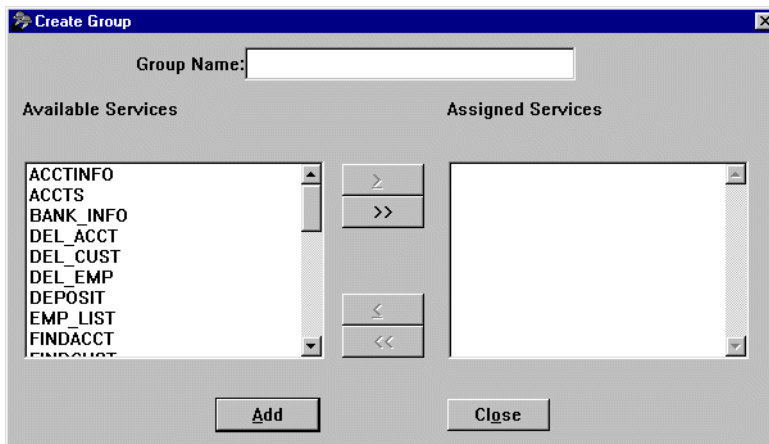


Figure 24-4 Define a service group with the Create Group screen.

2. Enter or modify the name of the group in the Group Name field. The group name must be unique in the JIF and can be no greater than 31 characters.
3. To add services to the group or remove unwanted services from the group, do the following as needed:
  - Select services in the Available Services list and choose the Assign button to transfer selected services to the Assigned Services list, or double-click on a service in the Available Services list to transfer it to the Assigned Services list.
  - Select the unwanted services in the Assigned Services list and choose the Remove button, or double-click on an unwanted service in the Assigned Services list.

**Note:** You can click+drag or Shift+click to select contiguous services or use Ctrl+click to select non-contiguous services in the list boxes.

The Assign All and Remove All buttons provide shortcuts to include all services in the group or remove all services from the group.

4. Choose Add.

The View Groups screen is automatically updated as service groups are defined or modified during an editing session.

## **How to Delete a Service Group**

1. Choose View→Groups. The View Groups screen opens.
2. Select the service group and choose Group→Delete.

The View Groups screen is automatically updated as services are deleted during an editing session.

---

# Defining and Updating Queues

---

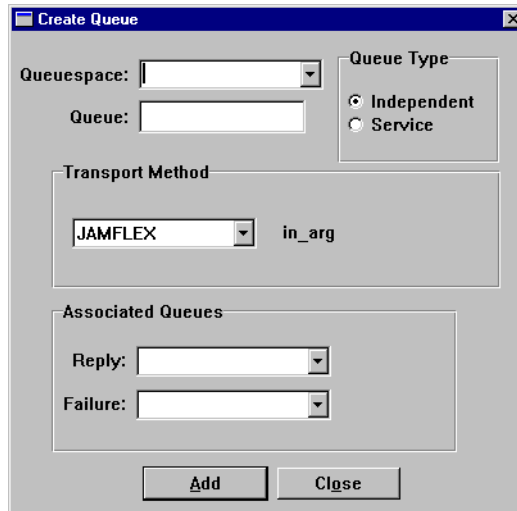
The Oracle Tuxedo middleware adapter uses a System/Q feature that allows messages to be queued—stored in stable memory and processed later. A queuespace contains queues, and queues contain messages. Messages are enqueued (pushed onto a queue), dequeued (released from a queue), and forwarded to either clients or servers, as the case may be. For further information about the System/Q feature, refer to “Reliable Queues” on page 8-11 in *JetNet/Oracle Tuxedo Guide* and refer to the *Oracle Tuxedo/Q Guide* in the Oracle Tuxedo documentation.

To process messages, you need to identify your queues and their respective queuespaces in the JIF. Queues are uniquely identified by their name and the name of the queuespace to which they belong. A queue can be either associated with a service or be independent. The parameters for a service queue are defined via its associated service, while the parameters for an independent queue are self-defined. In a service queue, enqueued messages get forwarded to the associated service having the same name as the queue. In an independent queue, messages are enqueued and dequeued explicitly by an agent.

## How to Define or Update an Independent Queue

1. Depending on the desired task, do either of the following:
  - To define an independent queue, choose Queue→Create. The Create Queue screen opens.  
The default queue type is specified to be an independent queue.
  - To update an independent queue, choose View→Queues. The View Queues screen opens. Select the queue to be updated and choose Queue→Update, or double-click on the selected entry. The Update Queue screen opens.

**Note:** The Update Queue screen displays the independent queue's current definition. It is identical to the Create Queue screen for an independent queue except for its title bar.



**Figure 24-5 Define an independent queue with the Create Queue screen.**

2. Enter or select the queuespace name in the Queuespace combo box. Each queuespace name can be no greater than 15 characters.  
**Note:** You cannot modify the queuespace name if you are updating a queue.
3. Enter or modify the name of the queue in the Queue field.  
**Note:** Queues are uniquely defined by their name and the queuespace to which they belong. Queue names need only be unique within a given queuespace.
4. For Transport Methods, select or change the argument type from the option menu and, for string arguments, enter the name in the field that appears.
5. Optionally enter or select a reply queue in the Reply combo box and/or a failure queue in the Failure combo box.
  - Reply queue—The queue might define a reply queue onto which replies will be enqueued. If there is one defined, its queuespace is also provided.
  - Failure queue—The queue might define a failure queue onto which failure notification messages will be enqueued. If there is one defined, its queuespace is also provided.



**Note:** If the reply queue or failure queue does not exist in the current queuespace, then it is created in this queuespace when the user chooses OK.

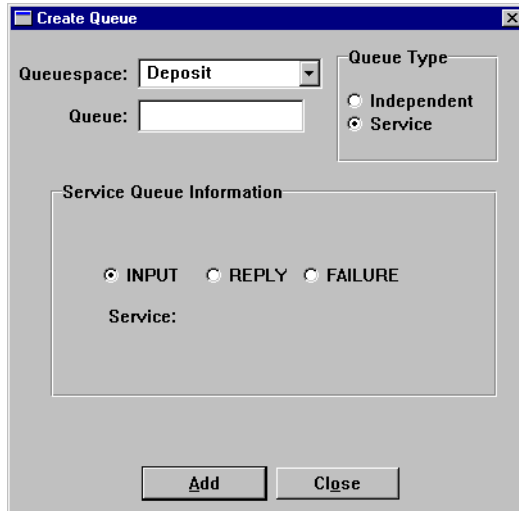
6. Choose Add.

The View Queues screen is automatically updated as independent queues are defined or updated during an editing session.

## **How to Define or Update a Queue Associated with a Service**

1. Depending on the desired task, do either of the following:
  - To define a service queue, choose Queue→Create. The Create Queue screen opens. Choose the Service radio button. The screen updates to show the new service queue.
  - To update a service queue, choose View→Queues. The View Queues screen opens. Select the queue to be updated and choose the Modify button, or double-click on the selected entry. The Update Queue screen opens.

**Note:** The Update Queue screen displays the service queue's current definition. It is identical to the Create Queue screen for a service queue except for its title bar.



**Figure 24-6** Define a service queue with the Create Queue screen.

2. Enter or select the queuespace name in the Queuespace combo box. Each queuespace name can be no greater than 15 characters.

**Note:** You cannot modify the queuespace name if you are updating a queue.

3. Enter or modify the name of the queue in the Queue field. Each queue name can be no greater than 15 characters.

For INPUT queues, the name of the queue must be the same as the name of the service associated with it. Thus, the associated service must already be defined and you must know its name before you enter the name of the queue.

**Note:** Queues are uniquely defined by their name and the queuespace to which they belong. Queue names need only be unique within a given queuespace.

4. Choose or change the queue type by selecting one of the following options:
  - INPUT— The data from the input parameters of the associated service is assigned to the queue.
  - REPLY—The data from the output parameters of the associated service is assigned to the queue.

- FAILURE—The data from the output parameters of the associated service is assigned to the queue.
5. Record or select the associated service in the Services option box depending on the chosen queue type. It is as follows:
    - INPUT—The name of the queue gets dynamically recorded as the associated service name.
    - REPLY—Select the associated service from the Services option box.
    - FAILURE—Select the associated service from the Services option box.

The data type of the message buffer that the queue uses is determined from its associated service.

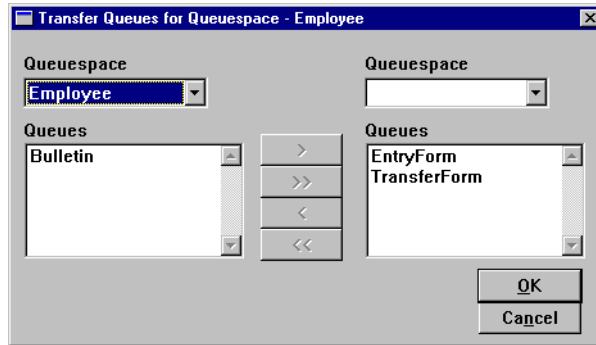
6. Choose OK.

The View Queues screen is automatically updated as service queues are defined or updated during an editing session. This screen lists the queues belonging to the selected queuespace.

## How to Transfer a Queue to Another Queuespace

You can only change a queue's membership if there are two or more queuespaces.

1. Choose View→Queues. The View Queues screen opens.
2. Select the queue you wish to move from one queuespace to another and choose the Transfer button on that screen. The Queue Transfer screen appears.



**Figure 24-7** The Queue Transfer screen is used to move a queue to a different queuespace.

3. Select two queuespaces from their respective option boxes.
4. Select the queues that you wish to move from the list under the appropriate queuespace.  
**Note:** You can click+drag or Shift+click to select contiguous queues or use Ctrl+click to select non-contiguous queues in the list boxes.
5. Choose the single arrow button that points in the direction toward the queue list to receive the queues.

If you want to move all queues from one queuespace to another, choose the double-arrow button that points in the direction of the queue list to receive the queues.

6. Choose OK.

The View Queues screen is automatically updated as queues are transferred between queuespaces. This screen lists the queues belonging to the selected queuespace.

## How to Delete a Queue

1. Choose View→Queues. The View Queues screen opens.
2. Select the queue and choose Queue→Delete.

The View Queues screen is automatically updated as queues are deleted during an editing session.

---

## **JPL Code Generation**

---

When you create your screens in the screen wizard, the JPL code for services and service requests is automatically generated and accessible to the screens. However, if you create your client screens and service components from scratch, then you need to generate your own service definition code and service request code. The JIF editor provides an easy-to-use mechanism for generating JPL code for services and service requests. The generated JPL code can be implemented as a:

- Widget or screen module.
- Library module.
- Text file outside of Panther.

## **Service Code Generation**

The JIF contains the necessary information about services that you defined via the Create or Update Services screen and can output this information to a template that generates the appropriate JPL code. You can generate the following two types of service code:

- Service definition code.
- Service request code.

In Windows and Motif, the service code is generated to the clipboard. In character mode, the code is written to a file. The output for code generated to either the clipboard or a file is as follows:

## Output For Service Definition Code

```
proc routine-name
  receive args (service-parameters)

  service_return (return-data)
```

*routine-name* is the name of the JPL or C procedure which will perform the named service, identified in the Routine Name field of the Create or Update Service screen.

*service-parameters* are the representations for the IN parameters.

*return-data* are the representations for the OUT parameters.

## Output For Service Request Code

```
service_call service-name (argument-list)
```

*service-name* is the name of the service, as identified in the Service Name field of the Create or Update Service screen.

*argument-list* are the representations of the IN and OUT parameters.

## Generating Service Code

You can generate service code to the clipboard or file, as the case may be. However, for character mode, you have to set the configuration variable `SMTPCLIPIFILE` either in the environment or in `SMVARS`. You need to set the value of this variable to the full path name of the file where you want the code written. If `SMTPCLIPIFILE` is not set, the default file name is `clip.dat`.

### How to Generate Service Code

1. To generate service code, you need to make the service information available. Depending on whether you have defined your service or not, do either of the following:
  - If you have an already defined service, select the service in the View Services screen.
  - If you have not defined your service, choose Service→Create and enter the service data; then select the service in the View Services screen.
2. Do either of the following to generate the desired code:

- Generate service definition code. Choose Edit→Copy Service (or the Copy Service button on the toolbar). The output JPL code is written to the clipboard for GUIs and to the `SMTPCLIPIFILE` file for character mode.
  - Generate service request code. Choose Edit→Copy Request (or the Copy Request button on the toolbar). The service call code is written to the clipboard for GUIs and to the `SMTPCLIPIFILE` file for character mode.
3. Paste the code, from the clipboard, to a file within a text editor. You can also paste code from the clipboard to the JPL Program Text window.

In character mode, remember to copy the contents of the file before the facility is used again, since `SMTPCLIPIFILE` is overwritten each time code generation is done.

## Format of Template

The format of the template used to generate code can be found in the `config/msgfile`. The values that undergo variable substitutions are enclosed in double angle brackets ("`<< . . . >>`"). These *directives* are replaced by the values that have been designated in the service's definition in the JIF. You can edit the template code in the `msgfile`, but you cannot create new directives. The template message strings that you can edit are as follows:

- `TP_JED_CLIP_TMPL_REQUEST`
- `TP_JED_CLIP_TMPL_SVC`

**Table 24-2 Recognized directives in the template**

Directive	Replaced with
<code>&lt;&lt;service_name&gt;&gt;</code>	Name of the service, as given in the Service Name field.
<code>&lt;&lt;routine_name&gt;&gt;</code>	The JPL or C routine that contains the code for the service, as given in the Routine Name field.
<code>&lt;&lt;service_component&gt;&gt;</code>	The name of the service component associated with the service, as given in the Service Component field.
<code>&lt;&lt;arg_list&gt;&gt;</code>	A list of 0, 1 or 2 arguments.

**Table 24-2 Recognized directives in the template (Continued)**

Directive	Replaced with
<<args_in>>	The arguments that are sent to the service.
<<args_out>>	The arguments that are returned by the service.

The text that replaces the argument directives listed in Table 24-2 depends upon the transport mechanism that they correspond to:

- <none>: the empty string.
- STRING: the appropriate (IN or OUT) parameter name.
- FML, FML32, JAMFLEX: the FML default mapping representation is { . . . }. If the buffer has not yet been defined, an unknown indicator is used, { ??? }, which will have to be edited by the user.

The JetNet-specific buffer types are JAMFLEX and none. The Oracle Tuxedo-specific buffer types are JAMFLEX, FML, FML32, STRING and none.

Two arguments in <<arg\_list>> are separated by a comma.

If directives are unintentionally created that are unrecognized, they are stripped of their angle brackets. Also, directives can be `escaped' (for example, to be used in a comment) by enclosing them in another set of angle brackets. The following are examples of how text with embedded angle brackets in the template code is interpreted by the JIF editor.

Text	Replaced with
<<unrecognized>>	unrecognized
<<<service_name>>>	<service_name>
<<<<service_name>>>>	<<service_name>>



# 25 Menu Bar Editor

With Panther's menu bar editor, you can create menus with pulldowns and submenus and attach them to your screens as menu bars. You can optionally specify items in a menu bar to appear simultaneously on a toolbar. Menus can also be invoked as popups from a screen or field, or as external submenus used by other menus at runtime.

After you define a menu, you can attach it to a screen or widget through its Properties window in the screen editor. Alternatively, you can load and install a menu at runtime through Panther's library functions. The library functions also let you change a menu's runtime behavior and appearance.

This chapter shows how to create menus with the menu bar editor. For information about Panther's menu system and runtime options, refer to Chapter 15, "Including Menus and Toolbars," in *Application Development Guide*.

---

## Starting the Menu Bar Editor

---

### How to Invoke the Menu Bar Editor

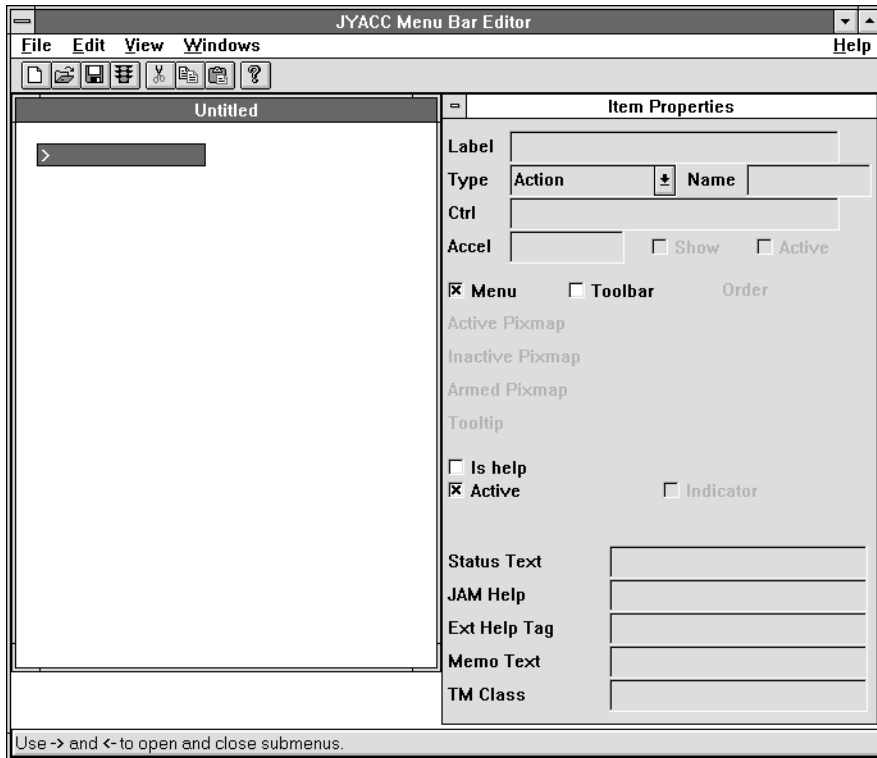
Do any one of the following:

- In the Panther editor, choose Tools→Menubar Editor.
- Type `mbedit` (or the full pathname of the menu bar editor executable).

To invoke the menu bar editor and open a specific menu file, use this command:

```
mbedit [menuFile]
```

*menuFile* must be a binary menu file, either created with the menu bar editor or converted from an ASCII menu script with `m2asc`. If you specify a file in an open library, the menu bar editor opens it; otherwise, it opens with a new untitled menu:



**Figure 25-1** The menu bar editor workspace, with the editor menu, toolbar, a new untitled menu, and the menu item's Properties window.

The menu bar editor workspace contains up to six components:

### Editor menu bar

The menu bar editor has its own menu bar with five pulldown menus: File, Edit, View, Windows, and Help. Use these menus to open and create menus and libraries, save menu files in libraries, connect to the middleware, maintain menu files under source control, add or remove items from the menu you are creating, and manipulate the workspace.

#### Editor toolbar

The menu bar editor has its own toolbar with buttons that execute often-used operations such as Open, Save, Cut, and Paste.

#### Design window

The design window is the area in which you build menus. The window title shows the file in which the menu is stored—in the case of a new unsaved menu, Untitled. You can edit only one menu bar at a time.

When you are in edit mode, a menu appears in its popup format—that is, the top-level menu items are displayed vertically. Test mode shows the menu in its menu bar format.

#### Item Properties window

The Item Properties window allows access to different sets of properties, according to the item's type. Some properties are valid for most items: the item type, its label, and so on. Other properties are dependent on the type you specify for the item.

This window automatically opens when you start the menu bar editor. You can close and open it through View→Item Properties.

#### Menu Properties window

The Menu Properties window displays the properties set for the current menu. Each menu in the menu file—for example, the main menu and its submenus—has its own set of properties.

You can open and close this window through View→Menu Properties.

#### Menu List window

The Menu List window lists all menus that are defined in the current file. Refer to “Using the Menu List Window” on page 25-16 for a detailed discussion of this window.

---

## Creating Menus

---

When you enter the menu bar editor, it displays a new untitled menu.

## How to Create A New Menu

1. Choose File→New (or the New button on the toolbar).
2. The menu bar editor displays a new menu with an end-of-menu marker (>).
3. In the Label property, enter the text of the first menu label and select which type of menu item it is.

## How to Insert New Menu Items

1. Select the item that is to be under the new item. To place an item at the bottom of the menu, select the end-of-menu marker (>).
2. Choose Edit→Insert. The editor places the new item above the previous selection with the label UNTITLED.

---

# Creating Submenus

---

A menu item can invoke a submenu if its Type property is set to Submenu. Items in a submenu can themselves invoke other submenus.

When you set a menu item's type to Submenu, the menu bar editor displays a submenu indicator (->) on the item; it also creates an empty menu—named and incrementally numbered (Submenu\_1, Submenu\_2, and so on)—and sets this name in the Submenu property of the parent item. You can accept this name or change it. If you enter a name that is currently undefined in this file, the newly created menu is assigned this name.

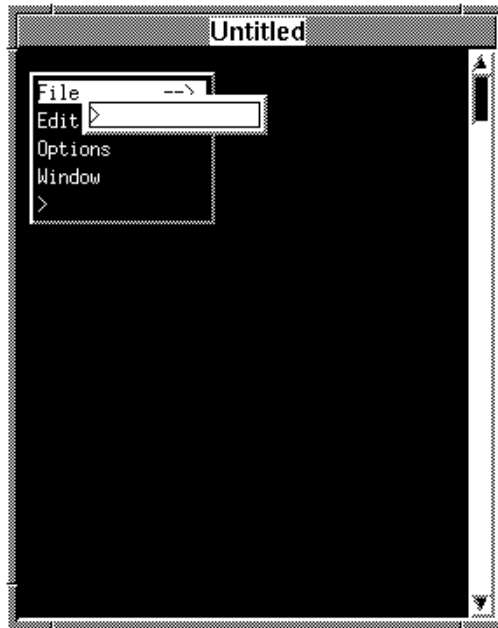
After you create a Submenu item, you can define its submenu in one of three ways:

## How to Attach An Existing Menu

Set the item's Submenu property. The property is set through a combo box, whose drop-down menu contains the names of all menus defined in the current file. Select one of these menus or enter the name directly; Panther attaches it to the submenu item.

## How to Edit The New Menu

Add items to the newly created submenu; select the submenu item and either click on it or press right arrow. An empty menu displays to the right and below the parent item. Edit this menu by inserting items and changing their properties.



**Figure 25-2** After the File menu item is defined as a Submenu item type, the editor attaches a new empty submenu to it.

## How to Specify An External Menu

Set the Submenu property to the name of a menu that is not defined in the current file. At runtime, Panther assumes that an undefined submenu is an *external menu* and searches for it among all menu files that are loaded into memory. This option is useful for invoking the same submenu in different menus, for example, a generic Edit menu.

---

## Menu Properties

---

Menus have their own properties, some of which apply to the menu, while others set default values for new items in that menu.

The following menu properties apply to menu items; refer to “Item Properties” [on page 25-7](#) for their descriptions:

```
Accelerator
Show Accelerator
Separator Style
Indicator State
Accelerator Active
Active
```

The following properties identify the menu and define its behavior:

### Menu Name

The name of this menu, up to 252 characters. If you rename a menu, Panther posts a warning message that this menu might be referenced by other items' Submenu property and asks for confirmation. Panther leaves references to the previous name unchanged; either change each reference to the new name, or make sure that the reference is resolved at runtime as an external menu.

### Menu Title

A title to display with the menu when displayed as a popup.

### Tear-Off

A check box that specifies whether to allow users to tear this menu off, valid only for Motif.

**Note:** Menu properties are accessible at runtime through Panther library functions. For more information on these, refer to Chapter 15, “Including Menus and Toolbars,” in *Application Development Guide*.

---

## Item Properties

---

All menu items have a set of properties. Some of these are valid for all menu items; others are type-specific. At runtime, you can use Panther library functions to access menu item properties. For more information, refer to “Changing Menus at Runtime” on page 15-9 in *Application Development Guide*.

### Label

Enter the text of this menu item. Labels can contain up to 255 characters. You can specify any letter within the label text as the item's mnemonic by prefixing that letter with an ampersand (&)—for example, `File` and `&Open`. Users can select the menu item from the keyboard by typing this character. To include an ampersand character in the label, precede it with another ampersand.

You can specify a label for all item types except separators and those types whose labels are system-defined—for example, `Edit` or `Cut`.

**Note:** Motif only receives the character of the mnemonic, not its position, and underlines the first occurrence of that character. For example, `Col&or` will underline the first occurrence of the letter `o` in Motif applications.

### Type

Choose the item type that determines this item's purpose—for example, whether it invokes an action, a submenu, or acts as a separator. Refer to “Item Types” on page 25-11 for descriptions of menu item types.

### Name

Enter the name of this menu item, up to 255 characters.

### Ctrl

For action and toggle items, enter a control string that specifies the action that occurs when this item is selected.

### Accel

For action and toggle items, optionally enter an accelerator string. This string specifies the keyboard equivalent for selecting this menu item and has the following syntax:

*keyboard-modifier -key*

where *keyboard-modifier* can be `Ctrl` or `Alt`, and *key* can be any alphanumeric or Panther function key—for example, `Ctrl-P` or `Alt-PF1`

Accelerator keys for edit-type items such as Edit Cut or Edit Paste are set by the GUI platform—for example, in Windows, through the Panther initialization file; on Motif, in the Panther file. To change edit item accelerators, modify the appropriate GUI file.

If you specify an accelerator string, two check box subproperties become available:

- **Show**—Specify whether a menu item displays the accelerator key next to the item label.
- **Active**—Specify whether to process the accelerator key as a menu item selection or as a Panther logical key. Set this property to off if the accelerator string is already mapped to a Panther logical key.

**Caution:** If an item's `Accel`→`Active` subproperty is set to on, do not map its accelerator string to a Panther logical key; otherwise, use of the accelerator key puts the application into an infinite loop.

### Indicator

Use this check box to set a toggle item's initial state to on. This property is valid only for toggle item types.

### Menu

Use this check box to display the item in the menu bar. Set this property to off in order to omit this item from the menu bar's initial display—for example, if you want the item to appear only in the toolbar, or display the item programmatically only in certain contexts.



## Toolbar

Use this check box to display the item in one of the toolbars. This property is valid only for action, toggle, and edit item types.

If you set this property, all toolbar subproperties become available:

- **Order**—Set the order in which this item appears on the toolbar. The default value is 100. You can enter any value between 0 and 200, inclusive. If all toolbar items are set to the same value, they appear in the same order as they do in the menu.
- **Active Pixmap**—Enter the name of an image file to be shown for an active toolbar item—that is, accessible but not pressed. Press Ctrl+Z to invoke the Select Library Member dialog box to choose a file. Refer to Table 25-1 on page 25-15 for valid file types. File extensions are optional. For more information on file types and platform usage, refer to [page 25-14](#), “Displaying Pictures on Toolbar Items.”
- **Inactive Pixmap**—Enter the name of an image file to be shown for an inactive or unavailable (grayed) item. Press Ctrl+Z to invoke the Select Library Member dialog box to choose a file. For more information, refer to [page 25-14](#), “Displaying Pictures on Toolbar Items.”
- **Armed Pixmap**—Enter the name of an image file to be shown for an armed item—that is, in its pressed state. Press Ctrl+Z to invoke the Select Library Member dialog box to choose a file. For more information, refer to [page 25-14](#), “Displaying Pictures on Toolbar Items.”

If this property is blank, Motif uses the Active Pixmap property for the item's armed state. Windows uses a modified version of the Active Pixmap property to display a toolbar item's armed state and ignores this property.

- **Hot Pixmap**—(Windows only) Enter the name of an image file to be shown as the mouse moves over an active toolbar item (that is, a toolbar item that is accessible but not pressed).
- **Tooltip**—Enter tooltip help to display when the cursor remains over the toolbar item. On Motif, the tooltip does not display for inactive items.

## Is Help

Use this check box to display the item as the rightmost item on the menu bar. You can specify this display option for only one menu item and only if it

appears on a menu bar. To designate another item as the help item, first set the previous help item's Is Help property to No; otherwise, Panther posts an error message.

The position of the corresponding toolbar item is unaffected by this property.

**Note:** Panther always shows the help item in its correct position during test mode. When you exit test mode, Panther automatically repositions the help item as the last item.

### Active

Use this check box to allow or disallow user access to the menu item. If unchecked, the menu item is initially greyed out; nothing happens when users click on the item.

### Submenu

Identify the menu to display when users select this item. This property displays only for submenu item types.

The drop-down menu of this property's combo box lists the menus that are defined in the current file. Choose one to use an existing menu. To define a new menu or specify an external menu, enter its name. In either case,

Panther creates an empty menu. You can add items to this menu or leave it empty.

**Note:** If you leave a menu empty, Panther discards it when you save. At runtime, Panther assumes that the referenced menu exists in another file and looks for it accordingly.

### Separator

Choose the style used by a separator item. This property displays only for separator item types. Choose a style from the option menu. For more information, refer to [page 25-13](#), "Separator Styles."

### Status Text

Enter the text to display on the screen's status line when this item has focus. On Motif, inactive items cannot get focus and therefore display no status line text.

### Help Screen

Enter the name of a Panther screen to invoke as a help screen.

Ext Help Tag

Enter a help context identifier that specifies the help to invoke from an external help program.

Memo Text

Enter a string expression. You can use the contents of this field programmatically.

TM Class

Enter a transaction manager property. Refer to “Setting Classes for Menu Items and Push Buttons” [on page 23-7](#) for valid arguments.

---

## Item Types

---

A menu item's type determines the action that occurs when users select the item. Each type is described in the following sections.

Separator

Draws a separator between the previous and next menu items, according to the specified separator style. Refer to “Separator Styles” [on page 25-13](#) for descriptions of different separator display options.

Action

Invokes an action through a control string.

Toggle

Changes an item's state through a control string and indicates whether the state is on or off with a toggle indicator. If the item is included on the toolbar, Panther uses the item's Armed Pixmap and Active Pixmap properties to show its on and off states.

You set the indicator's initial state through the item's Indicator State property. The indicator's state can be determined and reset through [sm\\_mnitem\\_get](#) and [sm\\_mnitem\\_change](#), respectively. Refer to “Changing the State of Toggle Items” [on page 15-10](#) in *Application Development Guide* for details on controlling the item's state at runtime.

In character mode, toggle items reserve a single space to the left of a label for the toggle indicator. To change the mark character for character mode applications—typically an x or check mark—assign a new value to `MARKCHAR` in the video file.

Motif ignores this setting for top-level items in a menu bar.

The menu items under **View** on the screen editor's menu bar are all toggle items. For example, when the **Widget List** is open, the menu item indicates that it is on, or open. When the item is toggled, the **Widget List** window closes.

### Submenu

Specifies that this menu item invokes another menu. You enter the name of the menu name in the `Submenu` property.

### Windows Ops

Invokes the platform-specific windows submenu:

- In **Windows**, it contains these items: `Cascade`, `Tile`, and `Arrange Icons`. These let you arrange screens and icons within the frame.
- In **Motif**, it contains a `Raise All` item that raises all Panther screens to the top of the display, and layers them according to the window stack.

Applications running on character mode platforms ignore this item.

### Windows List

Lists all accessible windows, with the last-opened window listed first. When users select a screen from this menu, Panther brings it to the top of the display. Windows applications omit this listing from popup menus.

## Edit Types

The menu bar editor provides standard editing types for menu items. These item types are valid only on GUI platforms:

### Edit Cut

Cuts selected text to the clipboard.

### Edit Copy

Copies selected text to the clipboard.

**Edit Paste**

Pastes the clipboard contents.

**Edit Delete**

Deletes the selected text.

**Edit Sel All**

Selects the current widget's contents.

**Edit Clear**

Replaces the selected text with blank spaces.

The following restrictions apply to edit item types:

- The Label property for edit items cannot be set; each GUI sets the labels for these items—for example, in Windows, through the Panther initialization file, or on Motif, in the `PROLIFICS` file. To change edit item labels, modify the appropriate GUI file.
- Use edit items only on pulldown and popup menus. If these item types appear on the menu bar, their state is set to inactive and they are greyed out.
- In character mode, edit-type items are not displayed. If a submenu is composed entirely of edit-type items, both the submenu and its parent item do not display.

**Note:** In test mode, edit item types are always inactive because no text is available to manipulate.

## Separator Styles

You can separate menu items with separator items—that is, items whose `Item Type` property is set to `separator`. You can specify the display characteristics of the separator item by setting the `Separator Style` property to one of these values:

Single  
Double  
Double Dashed  
Etched In  
Etched Out  
Dashed  
Etched In Dashed  
Etched Out Dashed  
None

In character-based applications, `single` and `double` can use characters defined in the video file. If no definition exists in the video file, Panther uses its own default values: `_` and `=` for vertical menus, `|` and `||` for horizontal menus.

On GUI platforms, each GUI determines which separator options are valid and how to display them. For information about GUI display of menu separator styles, refer to Table 45-10 on page 45-35 in *Application Development Guide*.

---

## Displaying Pictures on Toolbar Items

---

The menu item Properties window contains properties that let you specify the name of a bitmap image to display on a toolbar item:

- **Active Pixmap**—The image to show for an active item—that is, accessible but not pressed.
- **Armed Pixmap**—The image to show for an armed item—that is, in its pressed state. (Motif only)
- **Inactive Pixmap**—The image to show for an inactive or unavailable (grayed) item.
- **Hot Pixmap**—The image to show when the mouse passes over the item. (Windows only)

### Windows

Windows controls display of toolbars. Panther Windows applications support three pixmap states: Active, Inactive, and Hot. For Armed Pixmaps, Windows uses the pixmap specified in the Active Pixmap property. For Inactive Pixmaps, if inactive pixmaps are not specified for any of the toolbar items, a grayed version of the Active Pixmap property is used.

For each toolbar state that you want to indicate in your application—active, inactive and hot—you must supply a pixmap for each toolbar item.

## Motif

If you leave the Armed Pixmap or Inactive Pixmap properties blank, Motif uses the Active Pixmap image to display the push button in its armed (pressed) and inactive states. If the Active Pixmap property is blank, the menu item text is displayed.

## Image Size

Under Motif, bitmaps and pixmaps display at the size at which they were created. If you use different-sized images for an item's active, armed, and inactive states, Motif uses the largest of the three for the item's frame. In Windows, bitmap images need to be the same size. Panther uses a size of 16 x 15 pixels.

## Image File Types

Panther supports several file formats for pixmap properties. Table 25-1 shows which file types are valid on each platform:

**Table 25-1 Supported file formats for pixmap properties.**

File format	File extension	Windows	Motif
Windows bitmap	.bmp	•	•
Motif bitmap	.xbm		•
Motif pixmaps	.xpm		•
Graphics Interchange Format	.gif	•	•
Joint Photographic Experts Group	.jpg .jpeg	•	•
Portable Network Graphics	.png	•	•

## Filename Extensions

Pixmap properties can optionally contain filename extensions. To facilitate portability and provide maximum flexibility, you can omit the filename extension.

If you omit the file extension, Panther searches for files in all open libraries using all possible image extensions supported on the local platform.

Keep the extension if you have multiple images that use the same name but different types. For example, `dog.bmp` might be a bitmap image while `dog.jpg` is a photograph. In this case, include the extension to ensure that the correct image displays at runtime.

**Note:** Under Windows, if you compile the image into your resource-definition file, the resource id must match the pixmap property setting exactly.

Panther looks for the file according to the following platform-specific search procedures:

- Under Motif, if the pixmap is not found in an open library, X bitmaps are searched for as system files. The search path for this is dependent on environment variables. Refer to the `XmGetPixmap` function in the *OSF/Motif Programmer's Reference* for details.
- Under Windows, Panther searches for pixmaps in the following order of precedence:
  - a. The application's resource file.
  - b. Any DLLs loaded into memory by `sm_slib_load`.
  - c. Open libraries.

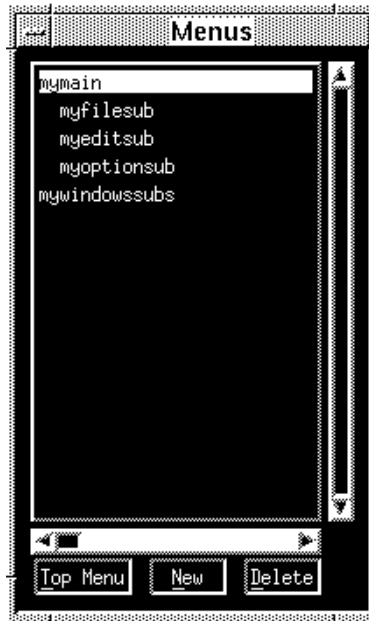
---

## Using the Menu List Window

---

The Menu List window lists the names of all menus defined in the current file. It provides a hierarchical view of the menu in outline form, where the topmost menu is the first entry, and its submenus are listed directly below it:





**Figure 25-3 Hierarchical listing of menus defined in a menu file. The last menu is flush left and outside the hierarchy, indicating that it is unused.**

The Menu List window also lists all menus that are defined in the current file but are unattached to a submenu item type. These names appear flush left at the bottom of the window. Unused menus are always accessible for later usage, either through the editor or at runtime through Panther library functions. For information about runtime menu installation, refer to the library function `sm_menu_install` in the *Programming Guide*.

Three push buttons let you create and delete menus, and manipulate the menu hierarchy:

**New**

Creates a menu and adds its name to the group of unused menus. You can attach this menu to a submenu item and add items to it.

**Delete**

Removes the selected menu's definition from the menu file. If the deleted menu has submenus nested within it, they join the group of unused menus.

#### Top Menu

Moves the selected menu to the top of the menu hierarchy; all of its submenus move up with it. Parent and sibling menus in the previous hierarchy join the list of unused menus.

---

## Testing Menus

---

You can test a menu and its toolbar in one of two ways:

- In the menu bar editor, choose File→Test Mode (PF2) or the Test Mode button on the toolbar

This lets you view the appearance of a menu and its toolbar independently of an application.

During test mode, a selected menu or toolbar item simulates the behavior associated with its type and property settings: submenu types invoke submenus, toggle and action items invoke messages with their control strings, tear-off menus can be torn off, and so on. You can also view a toolbar item's tooltip text by positioning the cursor over the item.

To exit test mode, use EXIT—Ctrl+E on Motif, Esc on Windows.

- Open or test a screen that uses the menu

This method lets you test the menu's functionality—for example, verify that a menu item's control string executes correctly. For more information, refer to [page 6-27](#), “Attaching a Menu Bar.”

---

# Saving Menus

---

The menu bar editor has two save options available on the File menu:

## Save

Saves the menu (in binary format) to its source file in a library. If the menu is new, the Save As Library Member dialog box opens where you can provide a name and specify a library. If no changes occurred since the last save, this menu item is greyed out.

## Save As

Opens the Save As Library Member dialog box where you can save the menu (in binary format) with a new name to a different library. This menu becomes the current one for this editing session.

You can convert binary menus to ASCII files and vice versa through the utility `m2asc`. The menu bar editor can only read binary files; in order to modify an ASCII menu file through the editor, first convert it to binary with `m2asc` and then use `formlib` to put it in a library.

Information about `m2asc` is [on page A-22](#) in *Application Development Guide*. Information about `formlib` is [on page A-14](#) in *Application Development Guide*. For information about ASCII menu format, refer to “Outputting Menu Definitions to ASCII” [on page 15-14](#) in *Application Development Guide*.





# Part V Appendices

Keyboard Interface

Wizard Output



# A Keyboard Interface

Panther provides a keyboard interface that allows you to use the editor in character mode without a mouse. The editor provides:

- Cursor movement keys—Provide positional manipulation of widgets and screens.
- Keyboard shortcuts—Serve as commands when editing a screen.
- Menu accelerators—Easily execute menu items.

The logical keys referred to in this appendix, such as `XMIT` or `EXIT`, are defined in your key translation file. Refer to Chapter 6, “Key Translation File,” in *Configuration Guide* for more information on key mapping.

---

## Navigating Between Screens in the Editor

---

With the keyboard you can navigate:

- Between your application screens.
- Between system screens, for example, the Properties window and your application screens.
- Between system screens.

This can be done via the menu bar mnemonics or via accelerators. To access the menu bar, press Alt followed by the mnemonic (the underlined or highlighted character) indicated on the menu choice—for example, to access the Windows menu and execute Next System Screen, press Alt+W, then press S. Table A-1 illustrates using mnemonics and accelerators for moving among screens in the editor.

**Table A-1 Navigating in the screen editor**

To access	Choose or Press	Accelerator
Properties window	Enter* (from an application screen)	
An open System menu	ALSYS	
Menu bar	MNBR	
Menu bar options	MNBR, then the menu option mnemonic	
Application screens	<u>W</u> indows→Next <u>U</u> ser Screen	F9
Panther windows	<u>W</u> indows→Next <u>S</u> ystem Screen	F10

\*Opens the Properties window, if not already open, and brings focus to it.

## Manipulating Screens

To move, resize, and scroll a screen, access the system menu (in the upper left corner of the screen):

1. Press ALSYS. The system menu contains the items you need to manipulate the screen that currently has focus.
2. Activate an action by pressing the mnemonic associated with that item. The system menu now displays as a menu bar. For example, press M to choose the Move option. Asterisks appear at the corners of the active screen.
3. Use the Panther logical arrow keys (such as LARR, RARR, DARR, UARR) to move, resize, or scroll the screen.
4. To carry out another action, press the mnemonic associated with that item.
5. Press XMIT to return focus to the screen and restore the default menu bar.



---

# Creating Widgets

---

Panther provides several ways to create a widget with the keyboard

To create a widget without using a mouse:

1. Use either of the following methods:
  - Create Toolbar—Choose Windows→Next System Screen until the Create toolbar box has focus. (If the Create toolbar is not already open, choose Options→Configure Toolbars, and select the Create box.) Press TAB or the Panther logical arrow keys to select the desired icon.
  - Create menu—Press `MNBR`, then `Alt+C` to open the Create menu. Press the Panther logical arrow keys or press the mnemonic associated with the desired widget. Press `NL` to accept.
2. Return focus to your application screen (for example, by pressing `F9` until the desired screen has focus).
3. Do any of the following to place the widget:
  - Move the cursor (use cursor movement keys) to the desired location on screen and press `NL`. A default sized widget is placed at the cursor position.
  - Move the cursor (use cursor movement keys) to the desired location on your screen. Press `S` to go into select mode and use the Panther logical arrow keys to drag the widget to the desired size. Press `NL` to accept.
  - Press `NL` to create a default sized widget. The widget is placed in the upper left corner of your screen.

If you have specified Multiple Create Mode on the Options menu, you can move to another location on the screen and press `Enter` or `S` to create another widget of the same type.

---

# Selecting and Manipulating Widgets

---

Use the following shortcuts while in an application screen to select objects. Once an object is selected, you can copy, move, and resize it as well as gain access to its properties. These keyboard shortcuts are case-insensitive.

**Table A-2 Manipulating widgets from a standard keyboard**

To	Description
Move the cursor	Use cursor movement keys (Panther logical arrow keys LSHF, RSHF, SPGU, and SPGD).
Select a screen	Position the cursor in an empty space on the screen and press + (plus).
Select a widget	Position the cursor on the widget and press + (plus).
Move to next widget	Press TAB.
Move to previous widget	Press BACK.
Select multiple widgets	Press S to start selection box; move the cursor to extend the selection box. Press XMIT to accept the selection box, or EXIT to cancel the selection.
Deselect a widget	Position the cursor on the widget and press - (hyphen).
Toggle selection	Press Spacebar to select/deselect the widget under the cursor.
Move a widget	Press M to move selected widgets; use cursor movement keys to position the selection set. Press XMIT to accept the move, or EXIT to cancel the move.
Copy a widget	Press C to copy selected widgets; use cursor movement keys to move the selection set. Press XMIT to accept the copy and place the widgets, or EXIT to cancel the copy.

**Table A-2 Manipulating widgets from a standard keyboard** (*Continued*)

<b>To</b>	<b>Description</b>
Resize a widget	Press <b>R</b> to resize a selected widget. Use cursor movement keys to increase the size of the widget in the desired dimension. Press <b>XMIT</b> to accept the size. Use <b>EXIT</b> to cancel the resize.
Cut a widget	Choose <b>Edit</b> → <b>Cut</b> or the accelerator <b>Ctrl+X</b> to cut a widget to the clipboard.
Paste a widget	Choose <b>Edit</b> → <b>Paste</b> or the accelerator <b>Ctrl+V</b> . The widget is pasted at the location from which it was cut. If another widget occupies that position, the pasted widget is shifted down and right by one grid unit.
Delete a widget	Press <b>DELE</b> to remove selected widgets.



# B Wizard Output

This appendix lists the properties that are set on screens, widgets, and reports generated by the screen and report wizards. It also describes the function of some of the property settings.

---

## Screen Wizard Output Property Specifications

---

The screen wizard sets several screen and widget properties for two-tier and three-tier applications. This section lists some of the important properties that control appearance and behavior.

### Two-Tier Property Specifications

#### Screen Property Specifications

The screen wizard sets screen properties that control screen behavior. You can edit these properties; however you must be aware that any changes to these properties will alter the behavior of your screens. Some of the important screen properties are as follows:

- Under Focus, the Entry Function property is set to `enter_screen`. This is a call to a simple screen entry procedure that publics the screen wizard's JPL module, `smwizard.jpl`.
  - Under Focus, the JPL Procedures property includes the `enter_screen` procedure. This is a one line procedure invoked on screen entry.
  - If you requested Menu/Toolbar controls, two more screen properties are set:
    - Under Focus, the Menu Script File property displays the name (`smwzmenu`) of the screen wizard's template menu file.
    - Under Focus, the Menu Name property is set to the name (`wzmain`) of the topmost menu within that file.
- Note:** The menu bar/toolbar does not display in the screen editor; it is visible in test mode.

## Widget Property Specifications

The screen wizard sets properties for the widgets that control their behavior and appearance.

You can edit the widget properties that control the appearance of your widgets. Some of the properties are as follows:

- Label widgets—Under Geometry, the Horiz. Anchor property is set to Right. This permits all label widgets to appear right-aligned and next to their corresponding text widgets.
- Grid widget—The following are some of the visual properties that are set:
  - Under Format/Display, the Stripe Current Row property is set to Yes. A highlight bar indicates the current row in the grid.
  - Under Format/Display, the Column Titles property is set to Per Column. Database column names appear across the top of the grid widget.
  - Under Identity, each column gets its title by way of its Column Title property.

You can also edit the widget properties that control screen behavior. Some of the important properties are as follows:

- Single line text widgets (or widget type adopted from repository)—The following properties are set on any widget of the First table that is a foreign key to an Additional table:
  - Under Help, the Selection Screen property specifies the name of the selection screen for the Additional table.
  - Under Database, the Validation Link property specifies the name of the link widget which connects the First table with the Additional table.

**Note:** If you have already set the Validation Link property for the widget in the repository, the screen wizard does not overwrite it.

- Grid widget—The following are some of the properties that are set:
  - Under Focus, the Row Entry Func is set to `fill_child`. This procedure ensures that all table views that are descendants of the current grid are refreshed whenever the grid stripe moves.
  - Under Focus, the Row Exit Func is set to `save_new_record`. This procedure saves the current row in the grid widget if it is a new row and it is non-empty.

## Selection Screen Property Specifications

For a two-tier architecture, the selection screens (.itm) generated by the screen wizard have the following features:

- The grid widget contains the columns you selected from that table and additional tables. The table's primary key is leftmost in the grid widget; the other columns continue off to the right.
- Two push buttons inside a positioning region box are under the grid: OK and Cancel. The positioning region keeps the buttons centered horizontally on the screen.

You can edit any of the following selection screen properties that control selection screen behavior.

- Grid members associated with the primary key have the Frozen Columns property set. This prevents these grid members from being moved. They can receive focus, but the content cannot be edited at runtime.

- Grid members associated with the primary key have their Double Click property (under Validation) set to translate a mouse double-click to the `XMIT` logical key. (The `XMIT` key behaves like an `OK`.)
- Non-primary key grid members have the Focus Protection property set to Yes. They cannot be updated or receive focus.
- The screen has the Entry Function property (under Focus) set to a function that populates the grid.
- The `OK` button's Control String property (under Validation) is set to a function that places the cursor back into the grid widget, and then causes an `XMIT` to happen. Choosing `OK` at runtime returns the selected item to the calling screen.  
**Note:** This control string works because the first column in the grid is also the first field on the screen. If you place any widgets above the grid widget, you will have to modify the call to `sm_o_gofield`.
- The Cancel button's Control String property is defined to close the screen.

If the primary key of the table consists of more than one column (sometimes known as a composite key), then two additional screen properties are set:

- Under Focus, the Control Strings property has one entry that maps the `XMIT` logical key to call the `copy_back` function.
- Under Focus, the JPL Procedures property contains the function `copy_back()`. This function copies all components of the primary key back to the calling screen.

## Three-Tier Property Specifications

### Screen Property Specifications

The screen wizard sets screen properties for three-tier client screens and service components that control screen behavior. You can edit these properties; however you must be aware that any changes to these properties will alter the behavior of your screens.

Some of the important three-tier client screen properties are as follows:



- Under Focus, the Entry Function property is set to `enter_screen`. This is a call to a procedure that publics the screen wizard's standard JPL module, `smwizard.jpl`.
  - Under Focus, the JPL Procedures property includes the `enter_screen` procedure, a one-line procedure invoked on screen entry.
  - If you requested Menu/Toolbar controls, two additional screen properties are set:
    - Under Focus, the Menu Script File property is set to `smwzmenu`, the screen wizard's template menu file.
    - Under Focus, the Menu Name property is set to `wz3tmain`, the name of a menu within the `smwzmenu` file.
- Note:** The menu bar/toolbar does not display in the screen editor; it is visible in test mode.
- Under Transaction, the Model property is set to the Middleware API transaction model. For further information on the Middleware API transaction model, refer to Chapter 7, “Transaction Model for JetNet,” in *JetNet/Oracle Tuxedo Guide*.

Some of the important service component properties are as follows:

- Under Focus, the Entry Function property is set to `enter_service`. This is a procedure that publics the screen wizard's service component JPL module, `smwizsrv.jpl`.
- Under Focus, the JPL Procedures property includes the `enter_service` procedure; a one-line procedure invoked on screen entry.

## Widget Property Specifications

For client screens in a three-tier architecture, the screen wizard sets properties for the widgets on the screen that control screen appearance and screen behavior.

You can edit the widget properties that control the appearance of your screens. Some of the properties are as follows:

- Label widget—Under Geometry, the Horiz. Anchor property is set to Right. This permits all label widgets to appear right-aligned and next to their corresponding text widgets.
- Grid widget—The following are some of the properties that are set:

- Under Format/Display, the Stripe Current Row property is set to Yes. A highlight bar indicates the current row in the grid.
- Under Format, the Column Title property is set to Per Column. This specifies that the column names appear across the top of the grid widget.
- Under Identity, each column gets its title by way of its Column Title property.

You can also edit widget properties that control screen behavior. Some of the important properties are as follows:

- Single line text widget (or widget type adopted from repository)—The following properties are set on any widget of the First table that is a foreign key to an Additional table:
  - Under Help, the Selection Screen property specifies the name of the selection screen for the Additional table.
  - Under Database, the Validation Link property specifies the name of the link widget which connects the First table with the Additional table.

**Note:** If you have already set the Validation Link property for the widget in the repository, the screen wizard will not overwrite the present value.
- Grid widget—The following are some of the properties that are set:
  - Under Focus, the Row Entry Func is set to `fill_child`. This procedure ensures that all table views that are descendants of the current grid are refreshed whenever the grid stripe moves.
  - Under Focus, the Row Exit Func is set to `save_new_record`. This procedure saves the current row in the grid widget if it is a new row and it is non-empty.
- Link widget—Under Service, the Validation Service property specifies the name of the service. The service receives data from the client into the service component, verifies that the primary key exists in the database and returns detail data for that primary key to the client. The service name includes the Service Prefix and the suffix `_1#` that indicates the validation link operation that the service implements.
- Table view widget—Service properties are set on the First Master table views only. The service properties specify the service names that are called for the different types of transaction manager operations or SQL operations.

- Delete Service—Specifies a service to implement a Delete operation that removes information from database tables.
- Insert Service—Specifies a service to implement an Insert operation that adds information to database tables.
- Select Service—Specifies a service to implement a Select operation that retrieves information stored in database tables.
- Update Service—Specifies a service to implement an Update operation that makes modifications to the data stored in database tables.

## Selection Screen Property Specifications

For a three-tier architecture, the selection screens (.citt) generated by the screen wizard have the following features:

- The grid widget contains the columns you selected from the Additional Table and any of its descendants. The table's primary key is leftmost in the grid widget; the other columns continue off to the right.
- Two push buttons inside a positioning region box are under the grid: OK and Cancel. The positioning region keeps the buttons centered horizontally on the screen.

You can edit any of the following selection screen properties; however you must be aware that any changes to these properties will alter the behavior of your selection screens.

- Grid members associated with the primary key have the Frozen Columns property set. This prevents these grid members from being moved.
- Grid members associated with the primary key have their Double Click property (under Validation) set to map a double-click to the `XMIT` logical key. (The `XMIT` key behaves like an OK.)
- Non-primary key grid members have the Focus Protection property set to Yes. They cannot be updated or receive focus.
- The screen has the Entry Function property (under Focus) set to a function call that populates the grid.

- The screen has the Model property (under Transaction) set to `jetrb1`, the transaction model for JetNet and Oracle Tuxedo applications that generates service requests.
- The OK button's Control String property (under Validation) is set to a function that places the cursor back into the grid widget, and then causes an `XMIT` to happen. Choosing OK at runtime returns the selected item to the client screen.  
**Note:** This control string works because the first column in the grid is also the first field on the screen. If you place any widgets above the grid widget, you will have to modify the call to `sm_o_gofield`.
- The Cancel button's Control String property is defined to close the screen.
- Under Service, the Select Service property is set for the first master table view with the service name specified for the corresponding `CHOOSE` operation on the Service Definition dialog. The service implements a Select operation that retrieves information stored in database tables.

If the primary key of the table consists of more than one column (sometimes known as a composite key), two additional properties set on the screen:

- Under Focus, the Control Strings property has one entry that maps the `XMIT` logical key to the `copy_back` function call.
- Under Focus, the JPL Procedures property contains the function `copy_back()`. This function sends all parts of the composite key back to the calling screen. This function copies all components of the primary key back to the calling screen.

---

## Report Wizard Output Property Specifications

---

The report wizard automatically sets a number of properties both on the report file itself and on individual widgets. This section briefly describes some of the properties that are set by the report wizard.

#### Report file

The report wizard defines and sets the following properties for the report file:

- Under Inclusions, the JPL Procedures property makes public the template JPL file `rwwizard.jpl`.
- Under Transaction, the Root property is set to the table view selected on the Table Selection dialog box.

#### Page orientation

If the width of the report exceeds 8.5 inches, the Orientation property is set to Landscape. Otherwise, the property value is set to Portrait.

#### Data fetching

For each detail node, the Data Source property is set to TM View. This indicates that the transaction manager executes `VIEW` and `CONTINUE` commands to fetch the report data. When the transaction manager executes these commands, data is fetched for any dynamic output widget that is a member of the current transaction.

#### Table views

For the root table view, the Sort Widgets property contains the names of widgets not included in the report totals. If you delete a widget from the report, you also must delete the widget from the Sort Widgets property. Otherwise, you get an error.

#### Data grouping

For each group node, the Break On property is set to a database column chosen in the Data Group Order dialog box. The hierarchy of group nodes corresponds to the order selected on that dialog.

#### Output

For each print node, the Area property is set to the appropriate layout area.



# Index

- (hyphen)  
in Properties window [3-29](#)

## Symbols

# (pound sign)  
with field number [9-15](#)

%A  
display attributes in messages [13-3](#)

%B  
bell for messages [13-4](#)

%K  
key label in message [13-4](#)

\* (asterisk)  
in regular expressions [15-12](#)  
indicator in color palette [12-5](#)

+ (plus sign)  
in Properties window [3-29](#)

??? (in properties window) [3-28](#), [3-29](#)

@date  
in Calculation property [9-28](#)

@length  
in Calculation property [9-29](#)

@sum  
in Calculation property [9-28](#)

[ ] (square brackets)  
selection indicator in character mode [2-14](#),  
[10-1](#)

\ (backslash)  
in regular expressions [15-9](#)

^ (caret)  
in regular expressions [15-10](#)  
{ } (curly braces)  
selection indicator in character mode [10-1](#)

## Numerics

3D property  
graph widget [14-22](#)  
screen [11-23](#)  
screens  
background color [11-24](#)

## A

Accel property  
menu item [26-8](#)

Accelerator  
assigning to menu item [26-8](#)

Action menu item [26-11](#)

Active Pixmap property  
menu items [26-14](#)  
push button widget [20-4](#)  
toolbar items [26-9](#)  
widget [22-11](#)

Active property [9-7](#)  
menu item [26-10](#)  
push button widget [20-7](#)

ActiveX controls  
calling an event handler [19-11](#)

- calling methods 19-9
- embedding in screens 19-2
- runtime license 19-4
- setting ActiveX properties 19-3
  - at runtime 19-6
- setting color properties 19-5
- Adding
  - methods for components 8-4
- Address label reports 6-11
  - setting dimensions 6-22
- Align command 10-14
- Aligning widgets 10-14
  - on grid coordinate 10-15
  - with Snap to Grid 10-15
- Alphabetic data 15-4
  - specifying range 15-13
- Alphanumeric data 15-5
- Alt Scroll Func property 9-25
- Alt Tab property 9-14
- Alternative scroll driver 9-25
- Application model selection
  - in screen wizard 5-17
- Architecture
  - specifying for screens in screen wizard 5-17
- Armed Pixmap property
  - menu items 26-14
  - push button widget 20-4
  - toolbar items 26-9
  - widget 22-11
- Array
  - behavior of 9-20
  - creating 11-14
  - horizontal 11-14
  - offscreen occurrence specification 11-15
  - required data and 15-15
  - scrolling 11-15
  - spacing between occurrences 11-14
  - synchronizing 9-20
  - value source for graph widget 14-25
- Array Size property 11-14
  - dynamic label widgets 14-4
  - synchronized group 9-23
- Auto Help property 13-8
- Auto Horiz Resize property 10-9
- Auto Item property 13-10
- Auto Vert Resize property 10-9
- Autonumbering
  - grid columns 16-7
  - grid rows 16-9
- Autotab 9-19
- Axis
  - graph widget 14-15
  - labelling 14-17
  - positioning 14-16
  - tick marks 14-18
- AxView 19-13
- B**
- Background color 12-5
- Bar chart
  - specifying in report wizard 6-20
- Bar Type property
  - graph widget
    - bar/line graph 14-41
- Bar/line graph 14-39
  - bar type 14-41
  - creating 14-39
  - data series style 14-26
  - legend 14-10, 14-11, 14-30
- Basic colors
  - defined 12-2
  - listed 12-4
- Bell
  - in status line text 13-4
- Bitmap
  - custom mouse cursor shapes 7-21
- Blank numeric field 11-22
- Blink display attribute
  - setting 12-7



- BMP files [22-10](#)
- Bold property [11-10](#)
- Border property
  - character mode screens [7-22](#)
  - list box widget [21-4](#)
  - screens [7-22](#)
- Border Style property
  - screens [7-22](#)
- Border Width property
  - graph widget legend [14-11](#)
- Box widget [3-24](#), [22-1](#)
  - 3D (in Windows) [11-25](#)
  - default style [22-2](#)
  - including title [22-4](#)
  - specifying alias style [22-2](#)
  - specifying style [22-3](#)
- Button/Menu Status property [24-9](#)
- Buttons
  - types of [3-22](#)
- C**
- Calculation
  - in fields [9-26](#)
  - using a date [9-28](#)
- Calculation property [9-26](#)
- Cancel push button
  - creating [20-6](#)
- Card Entry Function property [17-9](#)
- Card Exit Function property [17-9](#)
- Card Number property [17-8](#)
- Centering text
  - on box widgets [11-6](#)
- Character classes
  - in regular expressions
    - term [15-10](#)
  - special characters in [15-10](#)
- Character mode
  - multiple select mode [3-9](#)
  - selecting multiple widgets [10-2](#)
- Character-level regular expression [15-8](#)
- Characters
  - as unit of measurement [10-7](#)
- Chart Type property
  - graph widget [14-5](#)
- Check box widget [3-23](#), [21-2](#)
  - 3D (in Windows) [11-25](#)
  - appearance of [21-2](#)
  - displaying image on [22-10](#)
- Check Digit property [9-29](#)
- Check Overlap on Screen Save menu option [2-17](#)
- Child List window [2-32](#)
- Child property
  - table views [23-14](#)
- Child widget
  - finding [2-32](#)
  - finding parent of [2-31](#)
  - turning inheritance on/off [2-30](#)
- Circular property
  - array [11-15](#)
  - grid widget [16-9](#)
  - synchronized group [9-23](#)
- Class property
  - menu items [24-7](#), [26-11](#)
  - widgets [24-4](#), [24-5](#)
- Clearing Protect property [9-10](#)
  - in screen editor [15-16](#)
  - in styles editor [24-12](#)
  - scale widgets [15-20](#)
- Client service call code
  - written to clipboard [25-23](#)
  - written to file [25-23](#)
- Clipboard
  - writing code from JIF to [25-21](#)
- Clock Type property [11-18](#)
- Close Item property [7-25](#)
- Closing
  - framesets [18-9](#)
- CLR key (clear all)
  - clock update and [11-18](#)

- CLSID
  - generating new 8-10
- Color Name property 12-3
- Color properties
  - 3D effect on 11-24
  - changing 12-3
  - container color
    - specifying on Color palette 12-5
  - display attributes
    - setting 12-6
  - graph widget data series 14-29
  - in frameset 18-6
  - screens 3-32
  - transaction styles 24-10
  - types defined 12-1
  - widgets 3-30
- Column Click Action property 16-11, 16-13
- Column click behavior
  - functions
    - specifying 16-13
- Column Move Resize property 16-13
- Column property 2-28, 2-29
- Column reports 6-5
- Column Separators property 16-8
- Column Title property
  - set by screen wizard B-2, B-6
- Column Titles property
  - for grid widgets 16-7
  - set by screen wizard B-2
- Columns property
  - for splitter widgets 18-4
  - table view 23-4
- COM components 8-9
  - saving 8-11
- Combo box widget 3-21, 15-20
  - 3D (in Windows) 11-25
  - and autotab behavior 9-19
  - assigning double-click event to 9-11
  - controlling size of 15-22
  - populating 15-21
  - scrolling 15-22
  - specifying initial text 15-24
- Comments property
  - assigning in screen wizard 5-22
  - screens 7-30
  - widgets 9-30
- Component Interface 3-9
- Component interface
  - defining 8-4
- Composition properties
  - widgets 3-32
- Conceal Tabs property 17-7
- Configure Toolbars menu option 3-18
- Connecting
  - to database
    - in Editor 2-12
  - to JetNet 2-9
- Constant data 15-21
- Container color 12-2
  - defined 12-5
- Continuation file
  - specifying availability of 23-6
- CONTINUE
  - dbms commands 23-6
- Control panel (Windows)
  - defining color scheme 12-2
- Control string
  - assigning to menu item 26-8
  - executing from menu item 26-11
  - list box widget 21-7
  - push button widget 20-7
  - syntax 20-8
- Controls
  - specifying in screen wizard 5-21
- Convert Case property 15-13
- Count Select property
  - specifying size of select set 23-7
- Create menu 2-13
- Create toolbar 2-13
- Creating
  - ActiveX control 19-2
  - application components (screens, reports,

- service components) 2-4
  - data entry widgets 15-1
  - framesets 18-1
  - grid 16-1
  - libraries 2-8
  - link 23-13
  - menu 26-3
  - push buttons 20-1
  - report
    - with report wizard 6-1
  - repository 2-20
  - screen 7-2
    - with screen wizard 5-1
  - service components 8-1, 8-2
  - table view 23-9
  - widget
    - widget types 3-19
    - widgets 2-13
  - Ctrl property
    - menu items 26-8
  - Currency format 11-21, 15-15
  - Currency Symbol property 11-21
  - Cursor
    - movement
      - in arrays 9-21
  - Custom spacing 10-17
  - Customer Drawn property 22-12
- D**
- Data
    - clearing field of 15-15
    - fetching
      - specifying size of select set 23-7
    - protecting 15-16
  - Data entry
    - required 15-14
    - widgets 3-20, 15-1
    - with input widget 3-21
  - Data filter 15-2
  - Data Formatting property
    - date/time specification 11-17
    - numeric specification 11-20
  - Data series
    - graph widget 14-23
  - Data type
    - specifying 9-5
  - Database
    - closing connections
      - in Editor 2-13
    - connecting to
      - in Editor 2-12
    - importing to a repository 2-25
  - Database columns
    - choosing in report wizard 6-14
    - importing to repository 2-25
    - viewing imported value 2-29
  - Database connections
    - closing
      - in Editor 2-13
    - opening
      - in Editor 2-12
  - Database properties
    - table view 23-4
    - widgets 3-31
  - Database tables
    - importing to repository 2-25
  - Date/time format 11-17
    - defaults 11-17
    - defining custom format 11-18
    - examples of custom formats 11-19
    - system update 11-18
    - variables for custom format 11-19
  - DB Interactions window 3-9
  - Debugger
    - enabling from screen editor 3-11
  - Decimal Places property 15-20
  - Decimal symbol 11-21
  - Decorations
    - on list box widget 21-4

- on screen borders 7-21
- Default push button
  - creating 20-6
- Default/Cancel property 20-6
- Defining
  - component interface 8-4
  - methods for components 8-4
  - properties for components 8-6
- Delete Order property 23-16
- Deleting
  - from a library 2-6
  - from a repository 2-23
- Depth property
  - graph widget 14-22
    - bar/line graph 14-43
    - high/low chart 14-51
    - pie chart 14-33
    - XY plot 14-47
- Design considerations
  - screens 7-1
- Detail specification
  - adding another table view 5-12
  - in screen wizard 5-11
- Detail-only report 6-19
- Dialog box
  - creating 7-5
- Dialog property 7-5
- Diameter property
  - graph widget
    - pie chart 14-34
- Digits only filter 15-2
  - and check digit calculation 9-29
  - and justification 15-2
- Dim display attribute
  - setting 12-7
- Direction property
  - graph widget
    - pie chart 14-35
- Directions property
  - screen 23-7
  - table view 23-6
- Display attributes
  - setting 12-6
- Display properties
  - screens 3-33
- Distinct property
  - table view 23-6
- Docking
  - toolbars 3-18
- Dominant widget
  - defined 10-1
- Double Click property 9-12
  - set by screen wizard B-4, B-7
- Double-click event
  - specifying 9-11
- Drop-down Data property 15-21
- Drop-down Screen property 15-22
- Drop-down Size property 15-22
- Drop-down Source property 15-21
- Dynamic label widget 3-20, 14-3
  - accessing with keyboard 9-4
  - assigning double-click event to 9-11
  - displaying image on 22-10
  - resizing 14-4

## E

- Edit Mask 15-5
- Edit menu 3-6
- Edit menu items 26-12
  - setting label 26-13
- Editor
  - starting 2-2
- Empty Format property 11-22
- Empty numeric field 11-22
- Entry Function property
  - grid widget 16-15
- Environment variables
  - defining pixmap location 26-16
- Event handlers
  - for ActiveX controls 19-11

- Events
    - for tab cards 17-9
    - in framesets 18-7
  - Exit Function property
    - grid widget 16-15
  - Expose Function property
    - for tab cards 17-9
  - Extended colors
    - defined 12-3
    - specifying in Color palette 12-5
  - External Help Tag property
    - menu items 26-11
    - widgets and screens 13-14
  - External menu 26-6
  - External screen 15-22
- F**
- FERA key (clear field)
    - clock update and 11-18
  - Field
    - alphanumeric filter 15-5
    - digits only filter 15-2
    - edit mask 15-5
    - numeric filter 15-4
    - regular expression 15-7, 15-8
    - select on entry 15-15
    - with yes/no entry 15-3
  - Field number
    - assignment 9-2
    - relative referencing in tab properties 9-15
    - specifying in tab properties 9-15
  - Field validation
    - using table lookup 13-13
  - Fill at Init 15-23
  - Fill at Popup 15-23
  - Fill Character property 11-22
  - Floating point
    - in calculations 9-26
  - Focus
    - menu items 26-10
    - push button widgets 20-6
  - Focus properties
    - screens 3-33
    - widgets 3-31
  - Focus Protection property 9-8
    - grid widgets 16-15
    - in styles editor 24-10
  - Font
    - aliasing 11-9
    - aliasing names for portability 11-7
    - application default 11-8
    - graph widget 14-8
    - grid widgets 16-5
    - GUI-specific names 11-11
    - italic 11-10
    - Panther-specific 11-9
    - point size 11-9
    - screen 11-8
    - setting bold attribute 11-10
    - setting point size 11-9
    - specifying 11-7
  - Font Name property 11-9
  - Font properties
    - screens 3-32
    - widgets 3-30
  - Foreground color 12-5
  - Form Name property 18-5
  - Format selection
    - in screen wizard 5-3
  - Format Type property
    - date/time fields 11-17
    - numeric fields 11-20
  - Format/Display properties
    - numeric format 11-20
    - text format 11-3, 11-17
    - widgets 3-31
  - Frames
    - creating 22-3
  - Framesets 18-1
    - closing 18-9
    - creating 18-1

- entry processing 18-7
- opening 18-6
- properties in web applications 18-14
- runtime properties 18-12
- sample application 18-16
- using in web applications 18-14

Frequency property 11-18

Frozen Columns property 16-7

Function property

- transaction hook 23-4

Functions

- column click behavior
  - specifying 16-13

**G**

Geometry properties

- screens 3-32
- widgets 3-30

GIF files 22-10

Graph properties 3-31

Graph widget 3-20, 14-4

- bar/line graph 14-39
- converting between chart types 14-30
- creating 14-1, 14-5
  - with report wizard 6-20, 6-26
- data series 14-23
- displaying in 3D 14-22
- fonts 14-8
- high/low chart 14-48
- including in report 6-7
- label text 14-13
- legend 14-10, 14-30
- minimizing chart re-draw 14-25
- orientation 14-14
- pie chart 14-31
- subtitle 14-10
- text size 14-9
- title 14-9
- X and Y axes 14-15

Graphics file

- supported for toolbar items 26-15
- supported formats 22-10

Graphics widgets 3-24

Grid (screen)

- align widget on 10-15
- and wallpaper pixmaps 7-18
- as unit of measurement 10-7
- defining size of 7-10
- snapping to 10-15

Grid Align command 10-15

Grid column

- autonumbering 16-7
- defining properties of 16-5
- frozen 16-7
- moving/resizing at runtime 16-13
- positioning 16-6
- titles 16-6
- using as row title 16-9

Grid Column property 16-6

Grid display layout

- specifying in screen wizard 5-16

Grid Height property

- screens 7-10

Grid members

- and focus protection 9-9
- moving/resizing 16-13
- removing 16-3
- selecting 16-3

Grid property 16-2

Grid row

- autonumbering 16-9
- defining properties of 16-8
- height
  - changing 16-11
- margin
  - adjusting 16-11
- specifying number of 16-8
- striping 16-10
- titles 16-9

- Grid Style property
    - graph widget tick marks [14-21](#)
  - Grid widget [3-21](#), [16-1](#)
    - and focus protection [9-8](#)
    - assigning titles to columns [16-6](#)
    - column click behavior [16-11](#)
      - function [16-12](#)
      - sorting [16-11](#)
    - defining number of occurrences [16-8](#)
    - deleting [16-4](#)
    - font specification [11-8](#)
    - hiding [11-4](#)
    - horizontal lines [16-10](#)
    - member types [16-4](#)
    - row margin [16-11](#)
    - scrolling [16-8](#)
    - selecting members in [16-3](#)
    - setting scrolling behavior [16-9](#)
    - sizing to content [16-5](#)
    - specifying number of rows [16-8](#)
    - vertical lines [16-8](#)
  - Grid Width property
    - screens [7-10](#)
  - Group
    - autotab [9-16](#)
    - changing members of [21-16](#)
    - check box widget [3-23](#)
    - creating [21-11](#)
    - identifying members of [21-11](#)
    - list box widget [3-23](#)
    - naming [21-14](#)
    - properties [21-13](#)
    - radio button widget [3-23](#)
    - selecting [21-12](#)
    - selection widgets [3-22](#), [21-10](#)
    - specifying allowable number of selections [21-14](#)
    - specifying initial selection [21-15](#)
    - toggle button widget [3-23](#)
  - Group member
    - changing [21-16](#)
    - identifying [21-11](#)
  - Groups
    - specifying in report wizard [6-16](#)
  - GUI-specific colors
    - using [12-3](#), [12-5](#)
  - GUI-specific line/box styles [22-3](#)
- ## H
- Height property
    - defined [10-6](#)
    - widgets [10-6](#)
  - Help
    - multilevel [13-8](#)
  - Help properties
    - menu items [26-10](#)
    - screens [3-33](#)
    - widgets [3-31](#)
  - Help screen [13-5](#)
    - attaching [13-8](#)
    - displaying [13-8](#)
    - external [13-14](#)
    - populating [13-6](#)
    - positioning [13-9](#)
    - to enter data [13-6](#)
  - Hidden property [11-4](#)
    - for tab cards [17-8](#)
    - for tab decks [17-7](#)
  - Hide Function property
    - for tab cards [17-9](#)
  - Hiding
    - horizontal lines in grid widgets [16-10](#)
    - row titles in grid widgets [11-5](#)
    - text [11-5](#)
    - vertical lines in grid widgets [16-8](#)
    - widgets [11-4](#)
  - High/low chart [14-48](#)
    - creating [14-49](#)
    - data series style [14-26](#)
    - legend [14-10](#), [14-11](#), [14-30](#)

- Horiz Rotation property
  - graph widget [14-22](#)
  - bar/line graph [14-42](#)
  - high/low chart [14-51](#)
  - XY plot [14-47](#)
- Horizontal Anchor property
  - widgets [10-19](#)
- Horizontal array
  - setting tab order in [9-16](#)
- Horizontal lines
  - in grid widgets [16-10](#)
- Horizontal property [11-14](#)
- Horizontal Scroll Bar property
  - grid widgets [16-6](#)
  - shifting fields [11-16](#)
- Horizontal Shrinking property
  - screen [7-14](#), [22-8](#)
- Hot Pixmap property
  - menu items [26-14](#)
  - toolbar items
    - toolbar property [26-9](#)

## I

- Icon
  - file types supported for [7-9](#)
  - identification [7-9](#)
- Icon property [7-9](#)
- Identity properties
  - screens [3-32](#)
  - widgets [3-30](#)
- Import
  - database objects [2-25](#)
  - re-importing database objects [2-28](#)
- In Data Space property
  - graph widget legend [14-12](#)
- Inactive Pixmap property
  - menu items [26-14](#)
  - push button widget [20-4](#)
  - toolbar items [26-9](#)
  - widget [22-11](#)

- Inches
  - as unit of measurement [10-7](#)
- Independent queue
  - creating in JIF [25-15](#)
- Indicator property
  - menu items [26-8](#)
- Indicator symbol
  - setting initial state on menu item [26-8](#)
- Inh (Inherit) button [2-29](#), [3-29](#)
- Inherit From property [2-28](#)
  - removing specification [2-30](#)
- Inheritance
  - controlling [2-29](#)
  - displayed in properties window [3-29](#)
  - finding child objects [2-32](#)
  - finding parent objects [2-31](#)
  - maintained when copying [10-11](#)
  - preventing [2-30](#)
  - restoring [2-31](#)
  - tooggling for a specific property [3-29](#)
  - turning on/off [2-29](#)
- Init Selections property
  - radio buttons [21-8](#)
  - selection group [21-15](#)
- Initial Text property
  - embedding punctuation [15-4](#)
  - list box widget [21-6](#)
  - widgets [11-2](#)
- Initial Value property
  - scales [15-20](#)
- Initialization property [15-22](#)
- Input devices
  - for data [15-19](#)
- Input filters [15-2](#)
- Input properties
  - widgets [3-31](#)
- Input Protection property [9-10](#), [15-16](#)
  - in styles editor [24-11](#)
- Insert Order property [23-16](#)
- Insufficient space
  - widgets [10-17](#)



Is Help property  
  menu item 26-9  
Italic property 11-10  
Item Selection key (ITSEL) 13-10  
ITSEL key (item selection) 13-10

## J

JetNet  
  connecting to middleware 2-9  
JetNet/TUXEDO  
  service components 8-15  
JIF  
  client access 25-1  
  message enqueueing 25-15  
  server access 25-1  
JIF editor  
  generating JPL code 25-21  
  menu bar description 25-4  
  source management 25-4  
  starting 25-2  
  workspace with list of services 25-3  
Join  
  table views 23-16  
JPEG files 22-10  
JPL  
  naming conventions 2-19  
JPL module  
  generated by wizards 4-5  
Justification  
  with digits only filter 15-3  
Justification property 11-5  
  box widgets 22-4

## K

Key  
  logical  
    displaying in message 13-4

Keyboard interface A-1  
Keystroke Filter property 15-2  
  alphabetic 15-4  
  digits only 15-2  
  edit mask 15-5  
  in styles editor 24-11  
  numeric 15-4  
  regular expression 15-7  
  yes/no entries 15-3

## L

Label Location property  
  graph widget  
    pie chart 14-37  
  graph widget axes 14-17  
Label property  
  box widgets 22-4  
  check box widget 21-2  
  for tab cards 17-7  
  graph widget 14-5  
  graph widget axes 14-17  
  menu items 26-7  
  push button widget 20-2  
  static label widgets 14-2  
  toggle button widget 21-10  
Label Source property  
  graph widget  
    pie chart 14-36  
  graph widget tick marks 14-20  
Label widgets  
  creating 14-1  
Layout selection  
  in screen wizard 5-15  
LDB  
  and widget names 9-4  
Legend property  
  graph widget 14-10  
  graph widget data series 14-30  
  identifying the data series 14-30  
  placement by location 14-12

- placement by position [14-13](#)
- Length property
  - defined for widgets [10-6](#)
  - shifting fields [11-16](#)
- Letters only [15-4](#)
- Library
  - creating [2-8](#)
  - creating remote [2-9](#)
  - opening [2-7](#)
  - opening remote [2-7](#)
  - table of contents [2-5](#), [3-8](#)
  - viewing contents (TOC) [2-5](#)
- Library TOC [3-8](#)
  - opening [2-5](#)
- Line graph
  - Bar/line graph [14-4](#)
  - Graph widget [14-4](#)
- Line Style property
  - graph widget data series [14-28](#)
- Line widget [3-24](#), [22-1](#)
  - 3D (in Windows) [11-25](#)
  - default style [22-2](#)
  - specifying alias style [22-2](#)
  - specifying style [22-3](#)
- Line Width property
  - graph widget data series [14-28](#)
- Line/Box Style property [22-2](#)
  - 3D effect on [11-25](#)
- Link widget [3-24](#)
  - creating [23-12](#)
  - properties [23-13](#)
  - selecting [23-13](#)
- Links
  - noncyclic [23-13](#)
  - specifying in screen wizard [5-13](#)
  - validation [23-21](#)
- List box widget [3-23](#), [21-3](#)
  - 3D (in Windows) [11-25](#)
  - and autotab behavior [9-19](#)
  - as part of combo box [3-21](#)
  - assigning double-click event to [9-11](#)

- attaching an action to [21-7](#)
- decorations [21-4](#)
- label [21-4](#)
- populating [21-6](#)
- scrolling [21-5](#)
- selecting items [21-3](#)
- title [21-4](#)
- Listbox Type property [21-3](#), [21-7](#)
- Locating
  - sources of inheritance [2-31](#)
- Location property
  - graph widget axes [14-16](#)
- Lookup specification
  - in Relations dialog box [23-19](#)
- Lookup table [13-13](#)

## M

- Major Increment property
  - graph widget tick marks [14-19](#)
- Master (only) format [5-4](#)
  - specifying layout [5-16](#)
- Master-Detail format [5-4](#)
  - defining link for [5-13](#)
  - specifying layout [5-16](#)
- Master-Detail-Subdetail format [5-4](#)
  - specifying layout [5-16](#)
- Math expression [9-26](#)
- Matrix reports [6-9](#)
  - setting the matrix style [6-22](#)
  - specifying headings [6-17](#)
  - wrapping text overflow [6-21](#)
- Max Data Length property
  - shifting fields [11-16](#)
- Max Occurrences property
  - grid widgets [16-8](#)
  - list boxes [21-6](#)
  - scrolling array [11-15](#)
  - synchronized group [9-23](#)

- Max Size property 10-9
- Max/Min property 7-8
- Maximize option
  - on screens 7-8
- Maximum Decimals property 11-21
- Maximum property
  - graph widget axes 14-19
- Maximum Value property
  - scales 15-19
  - setting 15-13
- Measurement
  - valid units of 10-7
- Memo Text property
  - menu items 26-11
  - screens 7-30
  - widgets 9-31
- Menu
  - ASCII format 26-19
  - attaching to screen as menu bar 7-27
  - creating 26-3
  - hierarchical view in editor 26-16
  - naming 26-6
  - popup for field 9-6
  - popup for screen 7-28
  - popup title 26-6
  - properties of 26-6
  - saving in editor 26-19
  - separator styles 26-13
  - tear-off 26-6
  - testing 7-29
    - in menu bar editor 26-18
  - unused in file 26-17
- Menu bar
  - assigning in screen wizard 5-22
  - displaying items on 26-8
  - screen wizard template 4-4
- Menu bar editor 26-1
- Menu item
  - assigning control string to 26-8
  - attaching external help 26-11
  - attaching help screen 26-10
    - attaching Panther help screen 13-9
    - displaying on menu bar 26-8
    - displaying on toolbar 26-9
    - inactivating 26-10
    - including help 26-11
    - indicator symbol
      - reserving space for 26-8
    - invoking submenu 26-12
    - keyboard mnemonic 26-7
    - naming 26-7
    - platform-specific types 26-12
    - properties of 26-7
    - right justifying on menu bar 26-9
    - setting status in transaction style 24-9
    - text 26-7
    - transaction classes for 24-7
    - types 26-11
      - action 26-11
      - edit 26-12
      - separator 26-11
      - submenu 26-12
      - toggle 26-11
      - windows list 26-12
      - windows operations 26-12
- Menu list window 26-16
- Menu Name property
  - menus 26-6
  - screens 7-28
- Menu property
  - menu items 26-8
- Menu Script File property 7-28
- Menu Title property 26-6
- Message
  - bell 13-4
- Message queuing
  - in the JIF 25-15
- Methods
  - adding
    - for components 8-4
  - changing
    - for components 8-4

- defining
    - for components 8-4
  - executing
    - ActiveX methods 19-9
  - Middleware
    - connecting to 2-9
  - Millimeter
    - as unit of measurement 10-7
  - Min Size property 10-9
  - Minimize option
    - on screens 7-8
  - Minimum Decimals property 11-21
  - Minimum Digits property
    - check digit modulus 9-29
  - Minimum Horizontal Space property
    - box 22-7
    - screen 7-12
  - Minimum property
    - graph widget axes 14-19
  - Minimum Value property
    - scales 15-19
    - setting 15-13
  - Minimum Vertical Space property
    - box 22-7
    - screen 7-13
  - Minor Increment property
    - graph widget tick marks 14-19
  - Mnemonic
    - accessing from data entry widgets 9-5
    - assigning to widget 9-4
    - attaching to menu item 26-7
  - Mnemonic Position property
    - push button label 20-3
    - widgets 9-5
  - Model property
    - screen 23-4
    - set by screen wizard B-5
    - table view 23-4
  - Money
    - currency format 11-21
  - More button 3-29
    - in Properties window 2-15
  - Motif resource file
    - defining color scheme 12-2
    - defining pixmap location 22-12
    - text alignment 11-6
  - Mouse
    - right-button behavior 9-6
  - Mouse pointer
    - custom shape creation 7-21
    - shape options for Motif 7-19
    - shape options for Windows 7-20
    - specifying 7-19
  - Mouseless interface A-1
  - Mullion
    - in a frameset 18-1
  - Multiline text widget 3-20
    - 3D (in Windows) 11-24
    - and autotab behavior 9-19
    - assigning double-click event to 9-11
    - specifying word wrap on 15-17
  - Multiple Create mode 3-10
  - Multiple Select mode 3-9
  - Must Fill property 15-15
- ## N
- Name property
    - link 23-14
    - menu items 26-7
    - selection group 21-14
    - style widget 24-9
    - table view 23-3, 23-14
    - widgets 9-3
  - Naming conventions
    - in Panther 2-17
  - Navigating
    - in wizards 4-6
  - Next Tab Stop property 9-14
    - on dynamic labels 9-4

- No border
  - screens [7-22](#)
- No Validation property [9-9](#)
  - in styles editor [24-12](#)
- Null edit
  - on required data field [15-15](#)
  - results of [11-13](#)
  - with edit masks [15-7](#)
- Null Field property [11-12](#)
- Null Text property [11-12](#)
- Null value
  - default indicator [11-12](#)
- Number of rows
  - specifying for grid widget [16-8](#)
- Number of Selections property
  - radio button widget [21-8](#)
  - selection group [21-14](#)
  - toggle button widget [21-9](#)
- Numeric data [15-4](#)
  - specifying for scale widgets [15-19](#)
  - specifying range [15-13](#)
- Numeric format [11-20](#)
  - defining custom format [11-21](#)
  - examples of [11-21](#)
  - including punctuation [11-21](#)
  - rounding [11-22](#)
  - zero format [11-22](#)
- O**
- Occurrence
  - referencing [9-3](#)
  - specifying in tab order [9-15](#)
- Onscreen Columns property [16-5](#)
- Onscreen Rows property [11-14](#)
  - grid widget [16-8](#)
  - list box widget [21-5](#)
- Opening
  - application components (screens, reports, service components) [2-4](#)
  - repository [2-21](#)
    - on startup [2-21](#)
- Operating system
  - date/time
    - displaying [11-18](#)
- Option menu widget [3-21](#), [15-20](#)
  - 3D (in Windows) [11-25](#)
  - and autotab behavior [9-19](#)
  - controlling size of [15-22](#)
  - populating [15-21](#)
  - scrolling [15-22](#)
  - specifying initial text [15-24](#)
- Order property
  - toolbar items [26-9](#)
- Ordering
  - grid columns [16-6](#)
- Orientation property
  - graph widget [14-14](#)
- Other Style property [22-2](#)
- Overlapping
  - widgets [2-17](#)
- Overlapping Widgets menu option [2-17](#)
- P**
- Packed decimal [9-6](#)
- Pane widget [18-4](#)
  - in a frameset [18-1](#)
  - properties in web applications [18-16](#)
  - runtime properties [18-13](#)
- Panther
  - starting [2-2](#)
- Parent First specification [23-16](#)
- Parent object
  - finding [2-31](#)
  - finding children of [2-32](#)
- Parent property
  - table views [23-14](#)
- Password Char property [11-5](#)
- Password Field property [11-5](#)
- Percent Location property
  - graph widget

- pie chart 14-36
- Pie chart 14-31
  - creating 14-32
  - legend 14-10, 14-11
  - segments 14-35
  - specifying in report wizard 6-20
- Pixel
  - as unit of measurement 10-7
- Pixmap
  - creating 20-5
  - displaying on toolbar items 26-14
  - displaying on widgets 22-10
  - location on system 22-12
    - Motif 26-16
    - Windows 26-16
  - on push button widget 20-5
  - sizing
    - on toolbar items 26-15
    - on widgets 22-12
- Placement property 11-21
  - graph widget legend 14-10
- Point Marker property
  - graph widget data series 14-29
- Point Size property 11-9
- Pointer property 7-19
- Popup Menu property
  - screens 7-28
  - widgets 9-6
- Popup Script File property 9-6
- Portability
  - font name 11-7, 14-8
- Position Region property 22-5
- Positioning properties
  - box widgets 22-6
- Precision property
  - for C type specification 9-6
- Preview button 4-6
- Previous Tab Stop property 9-14
- Primary keys
  - specifying in screen wizard 5-8
- Primary Keys property 2-29
  - table view 23-5
- Procedures properties
  - service components 3-33
- prodev
  - starting 2-2
  - startup options 2-3
- Properties
  - controlling inheritance 2-29
  - defining
    - for components 8-6
  - screen property headings defined 3-32
  - setting 2-14
  - widget property headings defined 3-30
- Properties window 2-14
  - expanding/collapsing 3-29
  - in menu bar editor 26-3
  - in styles editor 24-2
- Property specifications
  - set by report wizard B-8
  - set by screen wizard
    - three-tier screens B-4
    - two-tier screens B-1
- Protected widgets 14-1
- Protection
  - and tabbing order 9-9
  - focus 9-8
  - for widgets 9-8
  - from clearing 9-10
  - from clearing data 15-16
  - from entering data 15-16
  - scrolling field 9-10
  - shifting field 9-10
  - tabbing into 9-8
  - validation 9-9
- Punctuation
  - in numeric fields 11-21
- Push button widget 3-22, 20-1
  - assigning mnemonic 20-3
  - attaching action to 20-7
  - cancel 20-6

- deactivating 20-6
- default 20-6
- displaying images on 20-4
- in repository entry 20-2
- label text alignment 11-6, 20-2
- setting initial activation status 20-6
- setting status in transaction style 24-9
- specifying in screen wizard 5-21
- transaction classes for 24-7

## Q

### Queue

- associated with services 25-15
- changing queuespace membership 25-19
- creating in JIF
  - independent 25-15
  - service 25-17
- defining in JIF 25-15
- deleting in JIF 25-20
- independent 25-15
- updating in JIF 25-15

### Queuespace

- JIF 25-15

## R

- Radio button widget 3-23, 21-7
  - 3D (in Windows) 11-25
  - displaying image on 22-10

### Range

- in regular expression 15-7
- specifying 15-13

- Record-by-record reports 6-3

### Redo

- actions 10-20

### Region Margin property

- box 22-7
- screen 7-13

### Regular expression 15-7

- character classes

- term 15-10

- character-level 15-8
- concatenating 15-11
- constructing 15-9
- examples of 15-8
- field-level 15-8
- properties in screen editor 15-8
- repeating 15-11
- simple 15-9
- special characters 15-9

### Relations property 23-16

### Remote library

- creating 2-9
- opening 2-7

### Repeating property 11-12

### Report data

- displaying only detail data 6-19
- displaying only summary data 6-19

### Report structure 3-9

### Report title

- specifying in screen wizard 6-21

### Report types 6-3

- address labels 6-11
- column 6-5
- graph 6-7
- matrix 6-9
- row 6-6
- specifying in report wizard 6-12

### Report wizard

- choosing database columns 6-14
- grouping data 6-16
- including graphs 6-19
- resulting definition 6-23
- specifying report type 6-12
- starting 6-1
- template screen
  - and the repository 6-2
- totaling data
  - specifying totals 6-19

### Reports

- creating 2-4

Repository

- adding objects to screen from 7-4
- and the report wizard 6-1, 6-2
- and the screen wizard 5-1, 5-2
- creating 2-20
- importing database to 2-25
- naming conventions 2-19
- opening 2-21
- opening on startup 2-21
- table of contents 3-8, 7-4
- viewing contents (TOC) 2-22

Repository entry

- and widget names 9-4
- from a database table 2-25
- opening 2-21
- saving 2-24

Repository TOC 3-8, 7-4

- displaying comments 7-30, 9-30

Required property 15-14

- in styles editor 24-11

Reservation

- prompt for release of 3-10

Resize Function property 7-7, 10-10

Resizable property 7-7, 10-9

Reverse display attribute

- setting 12-7

Revert menu option 3-5

Right justified text 11-6

Root property 23-11

Root table view

- specifying 23-11
- specifying in report wizard 6-13

Rounding property 11-22

Row Entry Func property 16-15

Row Exit Func property 16-15

Row reports 6-6

- wrapping text overflow 6-21

Row Separators property 16-10

Row Titles property 16-9

Rows property

- for splitter widgets 18-4

Rubberbanding 10-2

Runtime licensing

- ActiveX controls 19-4

**S**

Sample applications

- framesets 18-16

Save Pref menu option 3-5

Saving

- application components (screens, reports, service components) 2-16

- COM components 8-11

- repository entry 2-24

- service components 8-1

Scale property

- graph widget axes 14-19

Scale widget 3-22, 15-19

- setting initial value 15-20

Scheme

- defined 12-2

- specifying on Color palette 12-5

- using Motif resource file 12-2

- using Windows control panel 12-2

Screen

- 3D (in Windows) 11-23

- adding objects from repository 7-4

- border 7-22

- clock update and 11-18

- creating 2-4, 7-2

- decoration 3-24

- documenting 7-29

- embedding ActiveX controls 19-2

- font property 11-8

- iconifying 7-7, 7-8, 7-9

- location 7-11

- manipulating without mouse A-2

- maximizing/minimizing 7-7

- menu

  - attaching as menu bar 7-27

- modal 7-5



- mouse cursor
    - specifying shape 7-19
  - navigating without mouse A-1
  - root table view
    - specifying 23-11
  - sizing 7-6
    - maintaining 7-7
  - title bar 7-24
  - viewing names of 2-5
  - viewport 7-6
  - virtual 7-6
- Screen editor
- menu bar description 3-4
  - starting 2-2
  - toolbar 3-3
- Screen format
- specifying in screen wizard 5-4
- Screen properties
- set by screen wizard
    - service containers B-5
    - three-tier client screens B-4
    - two-tier screens B-1
- Screen title
- displaying a 7-23
  - specifying in screen wizard 5-21
- Screen wizard
- application model selection 5-17
  - layout selection 5-15
  - output for three-tier architecture 5-25
  - output for two-tier architecture 5-23
  - service definition 5-19
  - starting 5-1
  - template screens
    - and the repository 5-2
  - Web-friendly screens 5-4
- Scroll bars
- on list box widget 21-5
- Scroll Increment property 11-15
- grid widgets 16-9
  - synchronized group 9-23
- Scrolling array
- alternative scroll driver
    - specifying 9-25
  - circular 11-15
  - creating 11-15
  - input protection 9-10
  - isolating 9-24
  - scroll increment specification 11-15
  - synchronizing 9-20
    - setting circular scrolling for 9-23
    - setting scroll increment 9-23
  - tab order 9-17
- Scrolling property
- arrays 11-15
  - list box widget 21-5
- Select on Entry property 15-15
- Selecting
- application component (screen, report, service component) 2-15
  - widget 2-15
- Selecting widgets
- in character mode 10-2
  - in grid widget 16-3
  - rubberbanding 10-2
  - using Widget List 10-3
- Selection screen 13-10
- attaching 13-12
  - creating 13-10
  - displaying on entry 13-12
  - properties set by screen wizard
    - three-tier architecture B-7
    - two-tier architecture B-3
  - specifying in screen wizard 5-18
  - using with table lookup 13-13
- Selection Screen property 13-12
- set by screen wizard 5-25, 5-28
- Selection service container
- creating 13-12
- Selection widgets 3-22
- grouping 21-11
  - and autotab behavior 9-19

- specifying number of selections 21-14
- Separator menu items 26-11
  - styles 26-13
- Separator property
  - menu item 26-10
- Sequential link type 23-14
  - join specification 23-17
- Server link type 23-15
  - join relationship 23-18
  - synchronized scrolling 9-20
- Server View properties
  - widgets 3-32
- Service
  - cache service container options 25-10
  - call options 25-11
  - controlling behavior of 25-8
  - defining in JIF 25-5
  - deleting in JIF 25-12
  - reply options 25-10
  - transaction type options 25-9
  - transport methods in JIF editor 25-8
  - updating in JIF 25-5
- Service code
  - generating for Windows and Motif 25-21
- Service component
  - associated with service 25-7
- Service components
  - creating 2-4, 8-1, 8-2
  - JetNet/TUXEDO 8-15
  - saving 8-1
- Service definition
  - in screen wizard 5-19
- Service definition code
  - written to clipboard 25-23
  - written to file 25-23
- Service group
  - defining in JIF 25-12
  - deleting in JIF 25-14
  - updating in JIF 25-12
- Service name
  - default conventions for 5-20
  - editing in screen wizard 5-20
- Service properties
  - link widgets 23-16
  - set by screen wizard
    - link widgets B-6
    - table view widgets B-6
  - table view 23-7
  - widgets 3-32
- Service queue
  - creating in JIF 25-17
- Service request code
  - written to clipboard 25-23
  - written to file 25-23
- Service template
  - format for 25-23
- Setting field
  - in properties window 3-28
- Shift Increment property 11-16
- Shifting field 11-16
  - and input protection 9-10
  - horizontal scroll bar 11-16
  - shift increment specification 11-16
- Sign property
  - for decimal specification 9-6
- Single line text widget 3-20
  - and autotab behavior 9-19
  - assigning double-click event to 9-11
- Size to Contents property 10-7
- smwizard screen 4-4
- smwizis screen 4-4
- smwizrw screen 4-4
- smwizsrv screen 4-4
- smwizweb screen 4-4
- smwzmenu 4-4
- Snap to grid 10-15
  - using 10-16
- Sort Order Function property 16-12
- Sort Order property 9-25, 16-11
- Sort Widgets property
  - table view 23-6

- Source Mgmt
  - Source code management 3-5
- Space command 10-16
- Spacing property
  - vertical arrays 11-14
- Spacing widgets
  - custom 10-17
- Splitter widget 18-1
  - horizontal splitter 3-27, 18-2
  - properties in web applications 18-14
  - runtime properties 18-12
  - two-way splitter 3-27, 18-3
  - vertical splitter 3-27, 18-2
- SQL
  - automated
    - setting properties for 23-8
- Start Angle property
  - graph widget
    - pie chart 14-35
- Start Column property
  - grid position 10-15
  - specifying widget position 10-15
- Start Row property
  - grid position 10-15
  - specifying widget position 10-15
- Starting
  - Panther editor 2-2
- Startup property 7-8
- Startup state
  - for screens 7-8
- Static label widget 3-19, 14-1
  - displaying active pixmap on 22-10
  - resizing 14-2
- Status line
  - bell 13-4
  - in screen editor 3-3
  - text
    - displaying 13-2
    - menu item 26-10
- Status Text property
  - menu item 26-10
- Stripe Current Row property 16-10
- Style property
  - graph widget data series 14-26
  - graph widget tick marks 14-20
  - list box border 21-4
  - wallpaper 7-18
- Style Source property
  - graph widget
    - pie chart 14-37
- Styles editor 24-1
  - invoking 24-1
- styles.sty
  - saving 24-13
- Subdetail specification
  - in screen wizard 5-15
- Submenu
  - attaching to menu item 26-10, 26-12
  - creating 26-4
  - naming 26-10
  - nesting 26-4
- Submenu property
  - menu item 26-10
- Subtitle property
  - graph widget 14-10
- Summary-only report
  - creating with report wizard 6-19
- Synchronization property 9-21
- Synchronized arrays 9-20
  - creating 9-22
  - defining number of occurrences 9-23
  - identifying members of 9-23
  - isolating 9-24
  - modifying 9-22
  - setting circular scrolling behavior 9-23
  - setting scroll increment 9-23
  - specifying onscreen size of 9-23
  - updating group of 9-24
- System menu
  - displaying on screen border 7-25

System Menu property 7-25  
System Update property 11-18

## T

Tab Card widget 3-25, 17-1  
Tab Controls 17-1  
    using 17-4  
Tab Deck widget 3-25, 17-1  
Tab dialog screen 17-5  
Tab Entry Function property 17-9  
Tab Exit Function property 17-9  
Tab order  
    and cursor positioning keys 9-13  
    changing 9-13, 9-16  
    default 9-13  
    hidden widgets and 11-4  
    specifying 9-12  
Table Lookup property 13-14  
Table of contents  
    of library 2-5, 3-8  
    of repository 3-8, 7-4  
Table property 2-29  
    table view 23-4  
Table views 23-1  
    accessing properties for 23-2  
    adding members to 23-10  
    connecting to database 23-9  
    creating 23-9  
    determining relationships 5-9  
    displaying members 23-10  
    identifying as root 23-11  
    identifying database table 2-29  
    linking two 23-16  
    properties for SQL generation 23-8  
    selecting 23-2  
    specifying additional in screen wizard 5-9  
        attaching selection screen 5-18  
    specifying master in screen wizard 5-6  
    synchronized scrolling 9-20

Tear-off menu 26-6  
Tear-Off property  
    Tear-Off property  
        menus 26-6  
Terminal  
    bell 13-4  
Text  
    entry widgets 3-20  
    format and display properties 11-3  
    graph widgets 14-7  
    hiding 11-5  
    justification 11-5  
Text Size property  
    graph widget 14-9  
    graph widget labels 14-9, 14-13  
    graph widget legend 14-9, 14-11  
    graph widget pie chart labels 14-37  
    graph widget subtitle 14-9, 14-10  
    graph widget title 14-9, 14-10  
Text widget  
    3D (in Windows) 11-24  
Thousands Separator property 11-21  
Threshold property  
    specifying size of select set 23-7  
Tick marks  
    graph widget axes 14-18  
Title  
    on grid columns 16-6  
    on grid rows 16-9  
Title Bar property 7-24  
Title property  
    graph widget 14-9  
    graph widget legend 14-11  
    list box widget 21-4  
    on borders 22-9  
    on wizard output 4-6  
    screens 7-23  
    specifying in screen wizard 5-21  
TM Class property  
    menu items 24-7, 26-11

- Toggle button widget 3-23, 21-9
    - displaying image on 22-10
    - label text alignment 11-6
  - Toggle indicator
    - on menu item 26-11
  - Toggle menu item 26-11
  - Tool box
    - multiple create mode 3-10
  - Toolbar
    - adding items 26-9
    - assigning images to items 26-9, 26-14
    - assigning in screen wizard 5-22
    - assigning tooltip text to item 26-9
    - configuring
      - in editor 3-18
    - ordering items in 26-9
    - screen wizard template 4-4
    - sizing images 26-15
  - Toolbar property
    - menu items 26-9
  - Tooltip
    - assigning to toolbar item 26-9
    - testing 26-18
  - Tooltip property
    - toolbar items 26-9
  - Totaled report data
    - specifying in report wizard 6-19
    - summary-only reports 6-19
  - Transaction
    - unspecified 23-12
  - Transaction classes 24-5
    - defaults 24-6
    - style assignments by mode 24-7
  - Transaction manager
    - guidelines for using 23-1
  - Transaction mode 24-3
  - Transaction model
    - specifying 23-4
  - Transaction properties
    - link widgets 23-14
    - screens 3-33
    - table view 23-3
    - widgets 3-31
  - Transaction styles 24-5
    - assigned to default classes 24-7
    - creating 24-8
    - defaults 24-6
    - saving 24-13
    - testing 24-13
  - Turning on/off inheritance 2-29
  - Tuxedo
    - connecting to middleware 2-9
    - System /Q support
      - JIF 25-15
  - Type property
    - menu items 26-7
    - sequential 23-14
    - server 23-14
- ## U
- Underlined property 11-10
  - Undo
    - actions 10-20
    - Undo levels 3-11
  - Unfiltered data 15-2
  - Updatable property 23-3
  - Update group
    - table view 23-11
  - Update Order property 23-16
  - User interface components
    - overview 1-1
  - Utilities
    - AxView 19-13
- ## V
- Validation
    - automatic help 13-5
    - field 9-9
    - using check digit 9-29
    - using table lookup 13-13

Validation Link property 23-21  
    set by screen wizard B-3, B-6

Validation properties  
    widgets 3-31

Validation Protection property 9-9

Validation Service property 23-16

Value Location property  
    graph widget  
        pie chart 14-35

Value Source property  
    graph widget data series 14-23

Vert Rotation property  
    graph widget 14-22  
        bar/line graph 14-43  
        high/low chart 14-51  
        pie chart 14-33  
        XY plot 14-47

Vertical Anchor property  
    widgets 10-19

Vertical lines  
    in grid widgets 16-8

Vertical Scroll Bar property  
    grid widgets 16-8  
    list box widget 21-6  
    scrolling arrays 11-15

Vertical Shrinking property  
    screen 7-14, 22-8

Viewport 7-6  
    changing 7-6

Virtual screen 7-6

**W**

Wallpaper 7-17  
    file types supported for 7-17

Wallpaper Pixmap property 7-18  
    in framesets 18-6

Warning property  
    specifying size of select set 23-7

Web applications  
    using ActiveX controls

    transferring data 19-8  
    using framesets 18-14

Web Options properties  
    screens 3-33  
    widgets 3-32

Widget List  
    using 10-3

Widget name  
    assigning 9-3  
    in tab properties 9-14  
    using in Relations dialog box 23-17

Widget properties set by screen wizard  
    three-tier client screens B-5  
    two-tier screens B-2

Widget Type property 11-3

Widgets  
    3D (in Windows) 11-24  
    accessing with mnemonic 9-4  
    arranging 10-13  
        insufficient space 10-17  
    automatic vs. custom spacing 10-16  
    centering in screen 10-18  
    changing types 11-3  
    command execution with 3-22  
    copying  
        in screen editor 10-11  
    creating 2-13  
    creating with keyboard A-3  
    database type 3-24  
    decoration type 3-24  
    deleting  
        in screen editor 10-13  
    displaying images on 22-10  
    documenting 9-30  
    dominant selection 10-1  
        setting 10-2  
    entering data automatically with 3-21  
    font property 11-8  
    identifying 9-2  
    overlapping 2-17  
    positioning 10-13

- aligning with other widgets 10-14
- moving 10-11
- spacing evenly 10-16
- units of measurement 10-7
- protecting 9-8
- resizing 10-5
  - at runtime 10-9
  - units of measurement 10-7
- selecting 10-1
  - on tab cards 17-4
- selecting with keyboard A-4
- selection type 3-22
- size
  - changing 10-5
  - specifying 10-6
  - unifying 10-8
- sizing to fit content 10-7
- specifying initial data for 11-2
- Width property
  - defined 10-6
  - for widgets 10-6
  - graph widget tick marks 14-20
- Windows initialization file
  - defining bitmap location 22-12
  - defining pixmap location 26-16
- Windows list menu item 26-12
- Windows operations menu item 26-12
- WinHelp 13-14
- Wizards
  - generated JPL modules 4-5
  - navigating in 4-6
  - templates 4-2
- Word wrap function
  - specifying 15-17
- Word Wrap property 15-18
- Word wrapped text
  - special logical keys 15-18

## X

- X Anchor property
  - graph widget legend 14-13
- X axis
  - Axis, graph widget 14-4
- X Location property
  - graph widget legend 14-13
- X Position property
  - graph widget
    - pie chart 14-34
  - graph widget legend 14-13
- X Value Source property
  - graph widget
    - XY plot 14-23
- XBM files 22-10
- XML
  - generating
    - for screens 7-31
    - for widgets 9-31
- XML properties
  - screens 3-33
  - widgets 3-32
- XPM files 22-10
- XY plot
  - creating 14-45
  - data series style 14-27
  - legend 14-10, 14-11, 14-30

## Y

- Y Anchor property
  - graph widget legend 14-13
- Y axis property
  - graph widget data series 14-29
- Y Location property
  - graph widget legend 14-13
- Y Position property
  - graph widget
    - pie chart 14-34
  - graph widget legend 14-13

Y Value Source property  
graph widget

XY plot [14-23](#)

Y1 axis

Axis, graph widget [14-4](#)

Y2 axis

Axis, graph widget [14-4](#)

Yes/No

data entry filter [15-3](#)

## **Z**

Zero Format property [11-22](#)

Zoned decimal [9-6](#)

ZOOM logical key

action-type list box [21-6](#)

list-box in selection group [21-15](#)

not supported for multiline text widget [15-18](#)