

Contents:

About This Document

Introducing Panther

Introducing the Tutorial

Module 1—Preparing the Server and the Client

- 1. Setting Up the Server**
- 2. Configuring the Servers**
- 3. Setting Up the Client**
- 4. Defining a Test Service**
- 5. Setting Up the Web Application Server**

Module 2—Creating and Testing Screens

- 6. Creating a Repository**
- 7. Using the Screen Wizard**
- 8. Defining Services**
- 9. Testing the Screens**
- 10. Setting Properties to Query the Database**

Module 3—Connecting the Screens

- 11. Enhancing the Screen**

12. Inheriting from the Repository

13. Writing and Executing JPL

14. Customizing Screen Behavior

Module 4—Extending the Application

15. Implementing Selection Screens

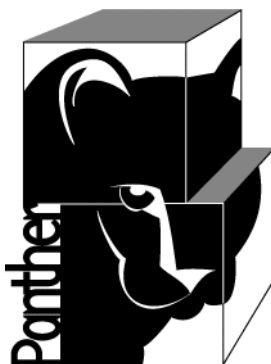
16. Calculating Data from Database Values

17. The Finale

A. Setting Up the Tutorial

B. Troubleshooting

Index



Panther

Getting Started

JetNet/Oracle Tuxedo

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

(800) 458-3313

© 1996-2017 Prolifics, Inc.

All rights reserved.

Contents:

About This Document

Documentation Website	xiii
How to Print the Document.....	xiv
Documentation Conventions	xiv
Contact Us!	xvi

Introducing Panther

About Panther	1
Solutions and Application Scalability	2
Simple Applications Use a Two-Tier Solution	2
Enterprise-wide Applications Use a Three-Tier Solution	3
Web Applications	5
Product Components	6
Visual Object Development	8
Editor	9
Screen Wizard Development.....	9
Repository	9
Menu Bar Editor.....	10
Styles Editor	10
JIF Editor.....	11
Debugger	11
Development Tools	11
Source Control Support.....	12
Programming Interfaces	12
Built-in SQL Database	13

Database Connectivity.....	14
Behind the Screens	14
Transaction Manager.....	14
Middleware Support.....	16
JetNet.....	16
Oracle Tuxedo.....	17
COM/MTS	17
IBM WebSphere.....	18

Introducing the Tutorial

About this Tutorial.....	1
Accessing Lessons in the Tutorial.....	3
About Each Module.....	3
Module 1—Preparing the Server and the Client	3
Module 2—Creating and Testing Screens	5
Module 3—Connecting the Screens.....	7
Module 4—Extending the Application	8
Hints for Completing the Tutorial	9
Before You Start.....	10
For More Information.....	10

Module 1—Preparing the Server and the Client

1. Setting Up the Server

UNIX Application Server.....	1-2
Create an application directory.....	1-2
Edit the environment setup file.....	1-3
Get new application components.....	1-4
Get tutorial components	1-5
Define the server environment	1-5
Link to the server executables	1-5
Create a middleware configuration file	1-7
Name the application.....	1-8
Get the machine's name and configuration file.....	1-9
Get the machine's port number	1-10
Provide the configuration file's location	1-11

Boot the application	1-11
Connect to the application.....	1-13
If you take a break.....	1-15
To resume the tutorial	1-15
To continue the tutorial	1-15
Windows Application Server	1-16
Create an application directory	1-16
Get new application components	1-16
Get tutorial components	1-17
Define the server environment	1-17
Copy the server executables	1-18
Start the JetNet manager	1-19
Create a middleware configuration file	1-20
Name the application.....	1-20
Get the machine's name and configuration file	1-21
Get the machine's port number	1-23
Provide the configuration file's location.....	1-24
Boot the application	1-25
Connect to the application.....	1-26
If you take a break.....	1-27
To resume the tutorial	1-28
What did you do?	1-28
What did you learn?	1-29

2. Configuring the Servers

Add servers to the application	2-2
Configure the standard server	2-2
Define server properties	2-3
Configure the file access server	2-5
Activate servers	2-6
If you take a break.....	2-7
What did you do?	2-7
What did you learn?	2-7

3. Setting Up the Client

UNIX Client	3-2
Apply environment settings.....	3-2
Connect to the middleware.....	3-3
Open a library and access library members.....	3-4
Save members to appropriate libraries	3-7
If you take a break	3-7
To continue the tutorial	3-8
Windows Client	3-8
Edit initialization files	3-8
Connect to the middleware.....	3-9
Open a library and access library members.....	3-11
Save members to appropriate libraries	3-13
If you take a break	3-14
What did you do?.....	3-14
What did you learn?.....	3-14

4. Defining a Test Service

Invoke the JIF editor.....	4-2
Define a service by connecting to the middleware.....	4-3
Are you connected?.....	4-6
For the next lesson.....	4-7
What did you do?.....	4-7
What did you learn?.....	4-7

5. Setting Up the Web Application Server

Before starting this lesson	5-2
Start the Web Setup Manager.....	5-3
Enter the program locations.....	5-4
Check the settings for your Web Application Server.....	5-6
General environment settings	5-10
3-Tier Configuration.....	5-14
Specify database settings and workstation jserver	5-16
Add JPL routines to the client library.....	5-18
Change permissions of shared files (UNIX only)	5-21

Test the connection.....	5-22
Stop and restart server after making any changes	5-23
Shut down the server	5-23
What did you do?	5-24
What did you learn?	5-24

Module 2—Creating and Testing Screens

6. Creating a Repository

Connect to the middleware.....	6-2
Create a repository	6-5
Connect to the database.....	6-6
Import database tables	6-7
View repository contents.....	6-9
What did you do?	6-10
What did you learn?	6-10

7. Using the Screen Wizard

Open the repository	7-2
Create screens with the screen wizard.....	7-2
Specify the contents of the master section	7-5
Define the detail columns.....	7-5
Specify application architecture	7-8
Determine service operations	7-9
Customize the output screen	7-10
Save the screens	7-11
What did you do?	7-15
What did you learn?	7-16

8. Defining Services

Invoke the JIF Editor.....	8-2
Define a service.....	8-4
What did you do?	8-6
What did you learn?	8-6

9. Testing the Screens

Access test mode	9-3
View data.....	9-4
Edit the data.....	9-5
Save the changes.....	9-6
Add a new record.....	9-6
What did you do?.....	9-8
What did you learn?.....	9-9

10. Setting Properties to Query the Database

Using the Properties window.....	10-3
Change properties locally	10-5
Edit the service component	10-8
View specific records	10-8
What did you do?.....	10-11
What did you learn?.....	10-11

Module 3—Connecting the Screens

11. Enhancing the Screen

Access table view properties	11-5
Update the JIF.....	11-8
Resize the screen	11-9
Move widgets	11-10
Open a repository entry	11-10
Copy widgets.....	11-11
Name the widgets	11-14
Define the query fields	11-14
Synchronize the service component	11-16
Query the database	11-17
What did you do?.....	11-18
What did you learn?.....	11-18

12. Inheriting from the Repository

Define user input	12-2
-------------------------	------

Define what the user sees	12-4
Propagate changes to screens and service components.....	12-5
Edit inherited property values	12-6
Create a push button widget	12-8
Define push button behavior	12-10
Create a buttons repository entry	12-11
What did you do?	12-12
What did you learn?	12-13

13. Writing and Executing JPL

Write a procedure to access a distributor's orders	13-3
Write a procedure to receive data.....	13-6
Generate a unique ID number	13-11
Insert the ID in the Database	13-13
Invoke the hook function on the server.....	13-14
Write a hook function for the client event.....	13-16
Invoke the hook function for the client event	13-17
Add a new database record.....	13-18
View orders	13-21
What did you do?	13-22
What did you learn?	13-22

14. Customizing Screen Behavior

Add double-click functionality.....	14-1
Write a screen entry function that executes only on screen exposure.....	14-3
Test the JPL.....	14-5
What did you do?	14-7
What did you learn?	14-7

Module 4—Extending the Application

15. Implementing Selection Screens

Join multiple tables.....	15-2
Add details from another table.....	15-3
Generate selection screens	15-5
Save the wizard output	15-8

Define link and validation services	15-9
Test the selection screen.....	15-9
Validate the data	15-12
What did you do?.....	15-13
What did you learn?.....	15-14

16. Calculating Data from Database Values

Add a column to the grid widget	16-3
Define a currency format.....	16-4
Define a math expression (for server processing)	16-4
Add the widget to a table view	16-6
Calculate results on the server	16-10
Calculate results on the client.....	16-11
Update totals on transaction manager events	16-12
Delete a detail record.....	16-12
Validate client data	16-15
Clearing data in a virtual field	16-17
Update a detail record.....	16-18
Connect two screens	16-20
What did you do?.....	16-21
What did you learn?.....	16-21

17. The Finale

A. Setting Up the Tutorial

B. Troubleshooting

Error Files	B-1
Setup and Connection Problems.....	B-2
Starting the application or servers	B-2
Booting the application	B-2
Activating a server	B-2
Setting up the Web application server.....	B-3
Starting the client.....	B-3
Starting a UNIX client.....	B-3
Connecting the client.....	B-3

Connecting to the server remotely	B-3
Connecting to the server locally.....	B-4
Accessing remote libraries	B-4

Index



About This Document

Getting Started serves as an introduction to Panther JetNet/Tuxedo Edition, offering an introduction to the Panther software and development process. It describes the features of Panther as an enterprise-wide development tool and gives step-by-step instructions for building a three-tier Panther JetNet application. This includes directions for setting up servers and clients to provide all Panther users with the concepts of administering an enhanced client/server environment.

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/index.htm>.

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.

Convention	Item
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
boldface text	Indicates terms defined in the glossary.

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

Introducing Panther

About Panther

Panther is a framework for component-based development that gives you a powerful tool for leveraging a hybrid application development approach—for increased speed-to-market, flexibility, integration, portability, reuse and enhanced responsiveness to business needs. Key features include:

- Industry standard component models—Panther supports industry-standard component models, EJBs and COM+, and simple conversion from one model to another. Panther also makes it easy for developers to use off-the-shelf COM components, ActiveX controls and JavaBeans enabling shortened development cycles, easier application maintenance and faster time-to-market.
- OTMs—Panther makes it easy for developers to build server components for use with OTMs, the component-based counterpart to TP middleware. Panther includes adapters for IBM WebSphere Application Server and Microsoft Transaction Server (MTS). That makes Prolifics the first and (at least for now) the only vendor to offer a solution that provides seamless integration with multiple industry-standard OTMs. So now developers can build and deploy reusable components for the most complex transaction processing applications faster than ever without being restricted to a proprietary development solution.
- Web Development—Panther's robust development capabilities and powerful application server mean that developers can construct web applications quickly using prebuilt components and Panther objects encapsulated in their own custom HTML. Leapfrogging over application servers that enable only web development, Panther offers a complete environment featuring all the tools

necessary for development, application integration and full-scale deployment. With Panther's integrated application server, developers can dynamically create HTML and build business logic completely in Java for enterprise-scale web applications.

The Panther framework contains a series of Panther software components, packaged in the following editions, to help you build enterprise-wide and web-based applications using the database of your choice:

- 2-Tier—Contains support for building applications using a two-tier architecture. Windows applications can also use COM components in their applications and deploy them using COM, COM+, DCOM, or MTS.
- 3-Tier JetNet—Contains support for JetNet, Panther's middleware product.
- 3-Tier Oracle Tuxedo—Contains support for Oracle Tuxedo, the leading TP monitor middleware from Oracle systems.
- 3-Tier WebSphere—Contains support for building and deploying EJB components for IBM's WebSphere Application Server.

Solutions and Application Scalability

With Panther software, you can build small, departmental-sized applications using traditional client/server principles as well as larger, high-demand enterprise-wide applications that require a more sophisticated, three-tier client/server architecture. In addition, you can build database applications that run on an intranet or the Internet. Panther's editor and screen wizard provide a visual environment in which to create your application's interface and business logic.

Simple Applications Use a Two-Tier Solution

The two-tier client/server model typically separates data from the logic of an application. The database server stores the application data while the client screens contain the business and programming logic and process user input.

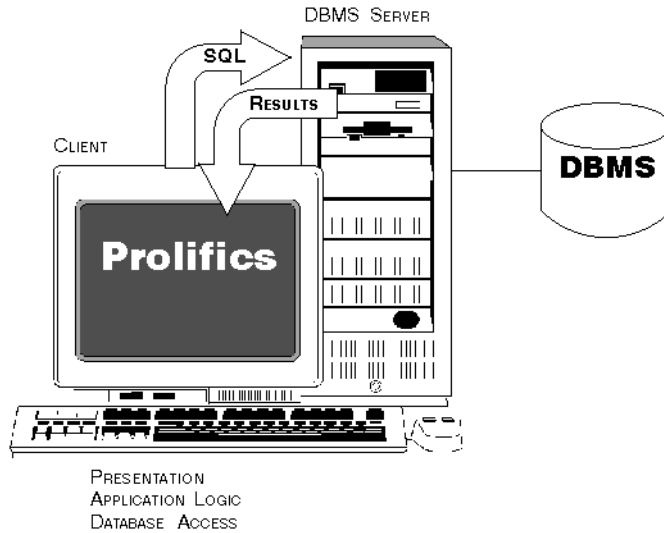


Figure 1 In two-tier architecture, each client has direct connection to the database server.

For small and departmental-sized applications, a two-tier solution can be the best alternative. With Panther software, you can build such applications and quickly test the interface and database connectivity. As the application requirements grow or the number of users grows, you can convert simple client/server processing to a more enhanced and enterprise-wide application.

Enterprise-wide Applications Use a Three-Tier Solution

Larger, enterprise-wide applications can be built quickly and easily with Panther. The interface you create is defined, in Panther terms, as the client. Essentially, the clients are processes which directly interact with the user. A client takes user input, packages it into a request for the middleware, and sends the request off. The middleware forwards the request to the Panther application server process which then implements that business logic. A client also receives replies from services and then presents the data to the user.

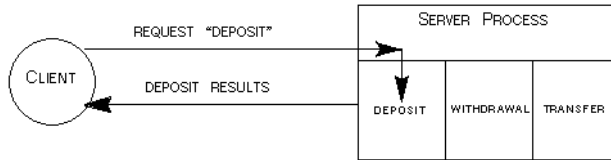


Figure 2 The client requests a service and the appropriate server responds.

In the three-tier or enhanced client/server model, the backend server is known as the resource manager, and is most often a database. The layer between client and backend server is the application server. This server handles the business logic of the application and doesn't need to reside at the client end. Hence, the client is responsible for user interactions, and the application server is responsible for providing business-level services and interacting with the resource manager as needed.

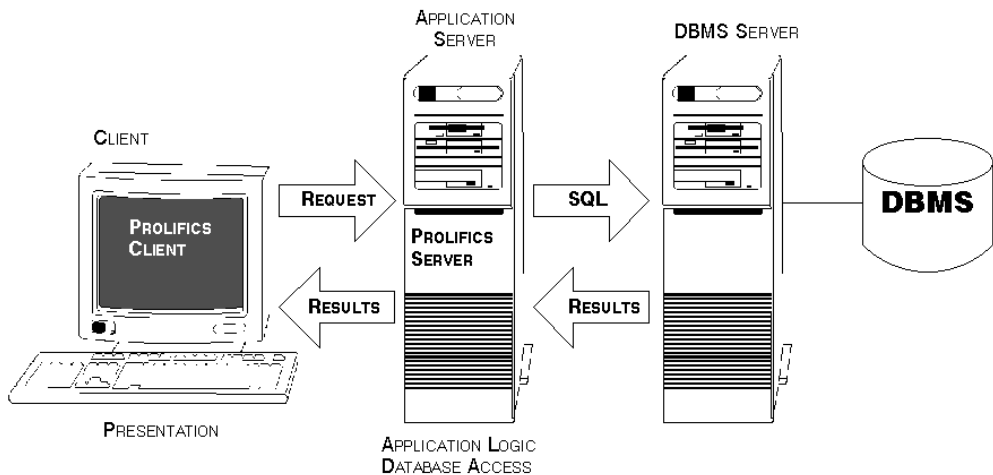


Figure 3 Three-tier clients have a connection to the database by way of the Panther application server.

Three-tier solutions address the needs of large-database users supporting many access points, usually in an open systems, client/server computing environment. Such transaction-processing applications are characterized by:

- High throughput, volume, and performance.

- Continuous real-time processing.
- A need to provide highly secured access to data and detailed control over its availability.
- Requirements for mechanisms that preserve transactional integrity and provide fast, reliable recovery.

The central component of a three-tier system is the middleware that manages communication among the components. Panther provides the tools you need to design and define the services that enable a transaction processing system to function in accordance with the application's requirements.

Web Applications

Your application can be deployed on the Internet or on an intranet. In three-tier applications, the web application server acts as a Panther client, submitting service requests for any data to the application server. In two-tier applications, the web application server has a direct connection to the database.

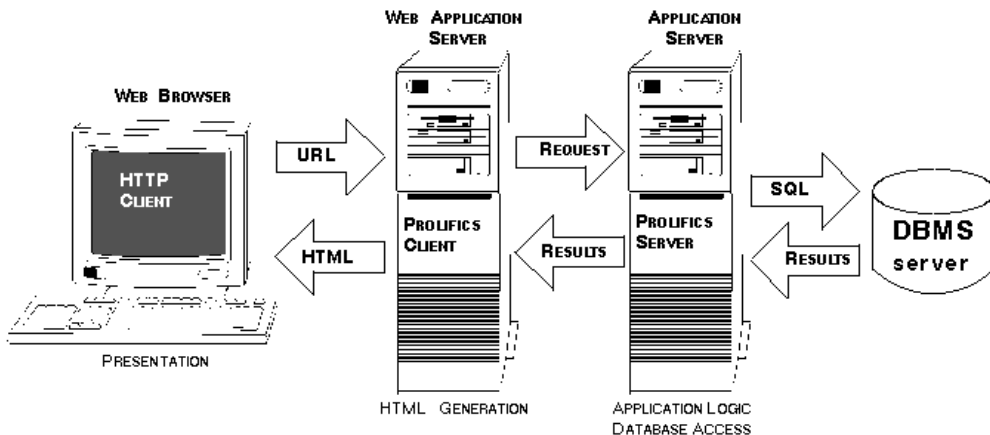


Figure 4 In Panther web applications, the web application server generates HTML for your web client screens.

Product Components

Panther is a framework providing you with everything you need for building n-tier client/server applications:

- Panther development tools:
 - Editor—A graphical environment for creating screens, reports, and service components, using widgets such as push buttons and data entry fields. Wizards are available to guide you through the process of creating screens or reports that access database information.
 - Visual object repository—A central library for creating, storing, and accessing objects used in building your application, allowing you to control and reuse them. In addition to screens and their objects, the repository also stores the properties associated with each object.
 - Libraries—A facility for storing all the objects used in an application. To be visible to the development team, an object must reside in a shared-access library.

Depending on the product, there can be up to three standard application libraries, divided according to where their members are accessed: client libraries contain application components that make up the user interface; server libraries contain server functions and service components required in three-tier processing; and common libraries contain components needed application-wide.

- Menu bar editor—For designing menu bars and toolbars.
 - Testing environment and built-in debugger.
 - Networked library and repository access for cohesive and controlled software development. This includes repository-stored application objects, interfaces to third-party source control (PVCS; SCCS and those with MSSCCI support), and, for some products, access to libraries on remote machines.
- Panther deployment tools:

- Programming options—You can use JPL (Panther's scripting language with a C-like syntax), Java or C to add programming logic to your screens, service components, and reports. All Panther objects and their properties can be accessed and modified programmatically through JPL modules, Java methods, or C function calls.
- Styles editor—For assigning styles that define an object's color and protection based on the current transaction.
- Database access and support, including:
 - Database drivers for your specific database engines.
 - Ability to import database definitions into a repository.
 - JDB database—A single-user SQL database. JDB can be used as a prototyping tool to test and refine multi-user database applications without requiring an external database.
 - Transaction Manager—A runtime component that performs the processing needed to view and update database information. The transaction manager automatically generates SQL statements from settings stored as screen and object properties.
- Middleware access and support, including:
 - Middleware adapter—A facility to connect to the middleware software which manages communication between the client and server so that data can be passed from client to middleware, middleware to server, and back again.
 - Ability to create service components to communicate with your chosen middleware, JetNet, Oracle Tuxedo, MTS, or WebSphere.
 - Runtime support for integration with your chosen middleware product.

Visual Object Development

The Panther development environment lets you build, test, and debug your application without having to recompile, relink, or leave the development workspace.

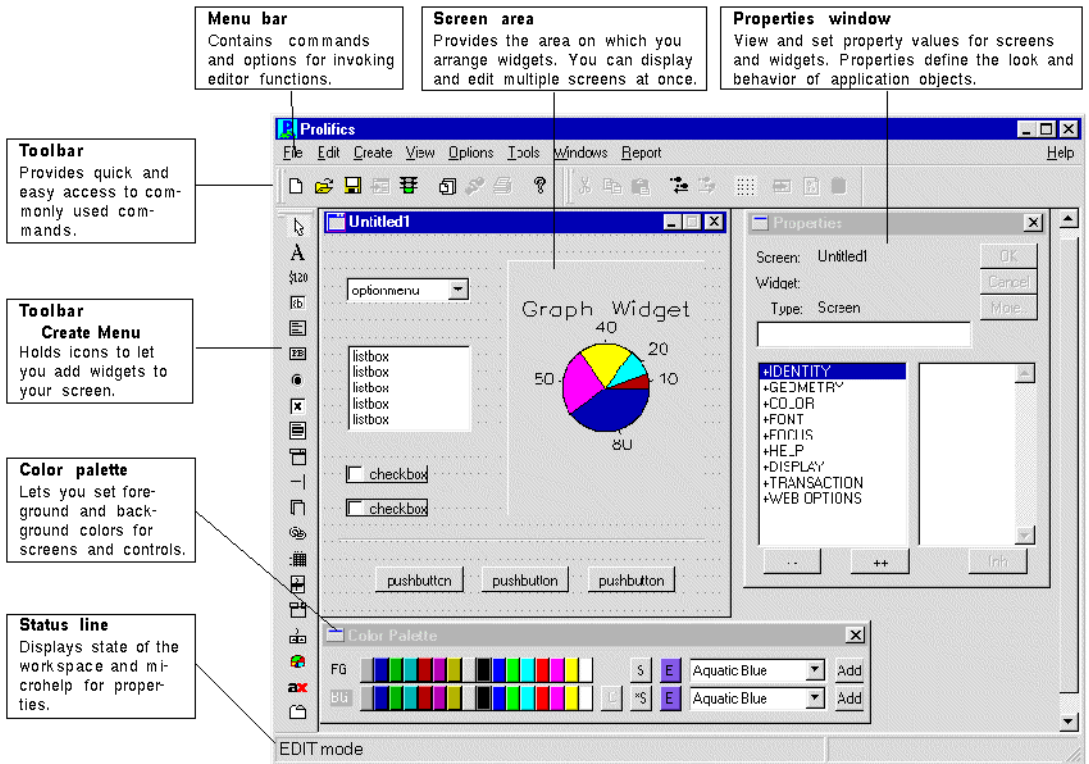


Figure 5 The editor workspace in Panther.

Editor

The editor is a powerful and fully graphical environment for creating and refining screens, reports, and service components. It lets you build client/server and distributed applications simply by dragging and dropping application objects onto Panther screens.

For information on accessing and using the editor, refer to Chapter 2, “Editor Basics,” in *Using the Editors*.

Screen Wizard Development

The screen wizard guides you through the process of building client screens for two-tier and three-tier applications which incorporate database tables and columns you import from your database. With the JetNet middleware adapter, the screen wizard can also build the corresponding service components.

Complex database-oriented screens with full master-detail-subdetail capabilities are easy to build and can be used immediately—because all the background processing needed to manage complex database transactions is built-in. The screens can be used as-is, or serve as a basis for further screen development.

For information on using the screen wizard, refer to Chapter 5, “Screen Wizard,” in *Using the Editors*.

Repository

Panther's visual object repository with multi-level inheritance provides an excellent single point of control over the data elements imported from your database and objects in a large application. A repository is a development tool that helps you establish a controlled environment and simplifies application maintenance. In a repository, you can create, store, and gain access to collections of refined and reusable application objects, each equipped with a discrete set of display and behavioral attributes called properties.

When you build client screens and service components from repository objects, you are provided with a comprehensive inheritance mechanism. Panther automatically sets up inheritance links between the objects you use. Changes to the repository entries are automatically reflected in screens and service components. Alternatively, inheritance can be overridden on a property-by-property basis.

For more information, refer to Chapter 11, “Creating and Using a Repository,” in *Application Development Guide*.

Menu Bar Editor

Panther's integrated menu bar editor lets you create menus (with pull-downs and submenus) which can be attached to your screens as menu bars and/or toolbars. Pull-down menus and their submenus can be nested as deep as you wish. You can associate menu bars with specific screens or widgets. You can also install a menu bar as the application-wide default to appear when no other menu bar has been specified. Menus can also be invoked as popups (by using the right mouse button) from a screen or field. In addition, Panther's library functions allow you to change a menu bar dynamically at runtime.

For more information, refer to Chapter 26, “Menu Bar Editor,” in *Using the Editors*.

Styles Editor

A style is a collection of properties that can be applied to a widget or menu item. The transaction manager determines, based on the current transaction, what makes the most sense and what style to apply to application objects. As a user runs your application, the appearance and behavior changes can provide visual cues, such as graying or ungraying a push button, to indicate a field's protection or availability. For the most part, styles can eliminate the need for you to code such property changes. The styles editor lets you fully customize styles that are automatically applied as needed.

For more information, refer to Chapter 24, “Styles Editor,” in *Using the Editors*.

JIF Editor

JetNet and Oracle Tuxedo applications use a JIF, or interface file, to act as the central facility for service and queue information used in enhanced client/server processing. The JIF editor lets you define the behavior of your application's services. It provides an environment for maintaining consistency between services and their invocation by clients. Via the JIF editor, you can group services for easier assignment to server instances and better manage your application. Features of the JIF editor also include automatic generation of service and service call invocation code.

For Oracle Tuxedo applications, the JIF editor also helps you define Oracle Tuxedo queues for use with enqueue and dequeue operations.

For more information, refer to Chapter 25, "JIF Editor," in *Using the Editors*.

Debugger

Panther's built-in debugger lets you visually step through events and scripts, while setting breakpoints and examining variables. The debugger is linked to the screen editor so you can easily switch between editing, testing, and debugging sessions. The debugger is available both on the client and on application servers, providing full three-tier debugging capabilities.

For more information, refer to Chapter 39, "Using the Debugger," in *Application Development Guide*.

Development Tools

The Panther development environment is equipped with numerous utilities and built-in capabilities to help eliminate or minimize tedious maintenance tasks. Most of what you need is completely accessible from within the editor environment. You can:

- Connect to your distributed application.

- Connect directly to your database.
- Use Panther's prototyping database, JDB, to quickly build and test your database applications.
- Take advantage of Panther's built-in controls for monitoring multi-user access of shared libraries and their contents as well as use features of your own source control management system.
- Use Panther's simple, but powerful scripting language to handle almost all of your programming needs.

Source Control Support

To ensure that all members of your development team have access to the same information and sets of standards, you want to allow multi-user access with assurances that write-access to files is controlled and monitored. The editor provides an interface to your source code management system, specifically SCCS; PVCS and those systems supporting MSSCCI, to help you maintain libraries and repositories.

In a distributed development environment, you can set up source control archives which can be accessed remotely; for example, you can use UNIX's SCCS to archive files which are accessed from a Windows client.

In addition, if you do not use or have a source code management system, the editor provides a default warning system for controlling concurrent access to shared application objects during the development process.

For more information about implementing configuration management, refer to Chapter 10, "Accessing Libraries," in *Application Development Guide*.

Programming Interfaces

With Panther, you have a choice of programming options in Panther software components. You can use JPL (Panther's scripting language with a C-like syntax), Java, or C to add programming logic to your screens, service components, and reports.

JPL is a powerful scripting language that provides a procedural component to Panther's event-driven environment. You can write JPL directly in the editor environment using your preferred text editor.

In addition to the built-in JPL functions, you can invoke Panther C library functions and your custom C functions from JPL procedures. Under Windows, you can also make calls to DLLs directly from your JPL code.

For more information, refer to Chapter 2, “JPL Command Reference,” and Chapter 5, “Library Functions,” in *Programming Guide*.

All Panther objects and their properties can be accessed and modified programmatically through JPL, C, or Java. With the properties API, you can identify any application object, including the application itself, and get or set its properties at runtime.

For a list of all Panther properties, refer to Chapter 1, “Runtime Properties,” in *Quick Reference*.

Programmers who are skilled in Java will find they can write application business logic in Java regardless of deployment environment. Panther provides a complete Java-based object framework and class factory as well as access to many Panther specific methods for interacting with an application.

For more information, refer to Chapter 21, “Java Event Handlers and Objects,” in *Application Development Guide*.

Built-in SQL Database

JDB is a fully integrated, single-user SQL database—a powerful prototyping tool that lets you test and refine multi-user database applications without the need for an external database. Use it on your servers, or use on your clients for local storage. If you have not chosen the database engine for your application or the production database is not immediately available, you can use JDB. Development can proceed while work continues on creating a production database.

For more information, start with Chapter 1, “Introduction to JDB,” in *JDB SQL Reference*.

Database Connectivity

From within the editor you can connect to your database and quickly begin developing database applications. You can import database table definitions into your application's repository at the outset of development—and then again whenever the database schema changes. If your database engine supports views and synonyms, you can import those as well.

Panther's transaction manager can automatically generate SQL statements thereby making your application database-independent. However, you can also write your own SQL. Panther provides the DBMS statements that let you take advantage of your database's unique features, such as executing stored procedures.

For more information, refer to Chapter 11, “DBMS Statements and Commands,” in *Programming Guide*.

Behind the Screens

The most powerful and useful tools in Panther are those that can't be seen. These runtime features make developing an application, be it two-tier or three-tier, easy and quick.

Transaction Manager

The transaction manager simplifies the process of building database applications by letting you invoke database operations and apply transaction-specific control attributes—without coding.

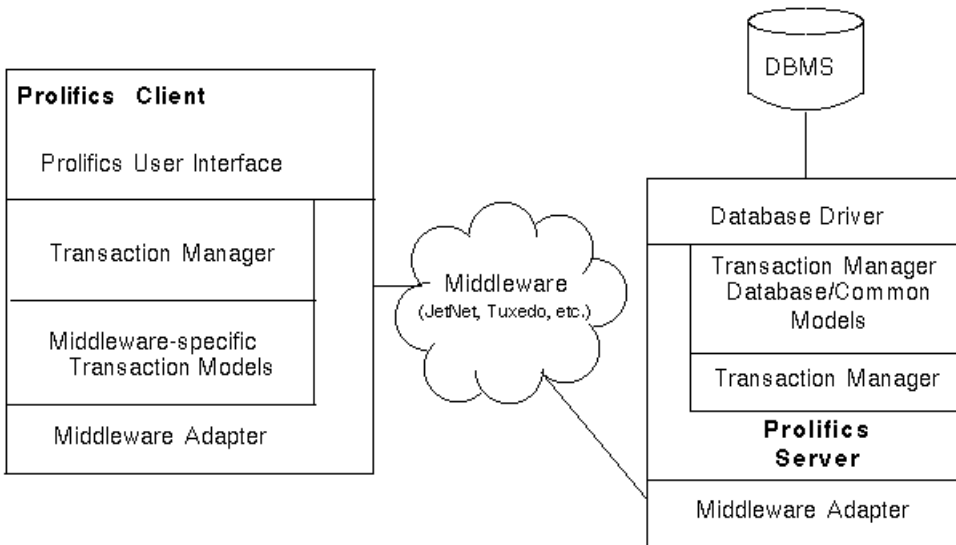


Figure 6 Application built for three-tier architecture with the screen wizard takes advantage of the transaction manager to generate service requests.

The transaction manager processes high-level commands related to operations requested by the end-user. It receives such requests—like view, save, and new—directly from a client, and from a server (in a three-tier application).

The request is sent to a database-specific transaction model, optimized for your target database, and a common transaction model. Typically, the models cause Panther to generate and execute the appropriate SQL statements, pass that to the database and then carry the results, by way of the application server in three-tier processing, back to the client, or user.

For more information on the transaction manager, its commands, and how to maximize its use, start with Chapter 31, “Building a Transaction Manager Screen,” in *Application Development Guide*.

Middleware Support

In a three-tier architecture, the communication between clients and servers across a network is managed by middleware software. The middleware adapter is the mediator between client and middleware and between server and middleware in Panther three-tier products.

JetNet

Panther's built-in middleware, JetNet, supports:

- Service requests in both synchronous (blocking) and asynchronous (non-blocking) modes—Multiple outstanding asynchronous requests are possible, and you can choose to wait for them in several highly flexible ways or allow Panther to handle service completion implicitly.
- Event handling—At critical points during execution, events are generated. Panther provides built-in handlers or you can write your own JPL or C routines to handle these events. Events include:
 - Exceptions (informational, warning, and error).
 - Receipt of unsolicited messages.
 - Initiation and termination of service requests.
 - Termination of the Panther application server.
- Message broadcasting and notification—Unsolicited messages from the current server or other clients can be handled by your own handler routines written in JPL or C.

The JetNet manager (`jetman`) provides you with the ability to configure, activate, and maintain Panther applications in three-tier architecture. The JetNet manager provides an easy-to-use interface for defining how your application will run—the structure of your application servers and the communication between the middleware and JetNet.

In addition, the JetNet manager lets you start and stop your application or individual servers running within the application. It gives you a view into your application—showing you what clients and services are connected and what they are doing.

Additional command-line utilities are provided:

- `rboot` is used to start JetNet and boot up your application servers.

- `rbshutdown` shuts down JetNet and your application servers.
- `rbconfig` provides an alternative method for creating a JetNet configuration file.
- `rblisten` allows application servers to run on multiple machines.

Oracle Tuxedo

Panther's Oracle Tuxedo version is completely compatible with Oracle Tuxedo and supports its features. In addition to the JetNet features, the Oracle Tuxedo version supports:

- Transaction control—Transactions can be demarcated with `BEGIN...END` blocks, allowing for the automatic generation of `ROLLBACK` and `COMMIT` commands.
- Stable-storage queuing—Takes advantage of the reliable queue management feature of Oracle Tuxedo System /Q.
- Event brokering—Clients and servers can subscribe to and post events. Event notification can be made by unsolicited message to clients, or by initiating a service call or queueing within Oracle Tuxedo's System /Q feature.
- Oracle Tuxedo-specific data transport buffers—`FML`, `FML32`, and `STRING` buffer types in addition to Panther's own buffer format (`JAMFLEX`).

COM/MTS

In Windows applications, you can build COM components in the Panther editor and deploy those components using `COM`, `COM+`, `DCOM`, or `MTS`. Those COM components can be used in a Panther application or be called from other COM-based applications.

Using MTS to deploy your components allows you to take advantage of the database connection pooling and transactional support that are built into MTS.

For more information on building and deploying Enterprise JavaBeans in Panther, start with Chapter 1, "Overview," in *COM/MTS Guide*.

IBM WebSphere

For IBM WebSphere applications, you can build EJB components in the Panther editor and deploy them using WebSphere Application Server.

For more information on building and deploying Enterprise JavaBeans in Panther, start with Chapter 1, “Overview,” in *Panther for IBM WebSphere Developer’s Studio*.

Introducing the Tutorial

The tutorial is designed to give you an overview of three-tier application development for JetNet and Oracle Tuxedo applications—providing you with a general methodology for enhanced client/server application development.

Whether you are new to Panther software or are already a user, a novice programmer or an experienced software engineer, you can build this mini-application to learn about Panther's application development process. In addition, you can use the tutorial's examples, principles, code, and concepts as a template for your own applications.

About this Tutorial

You are now ready for the feature presentation—get ready to begin the tutorial!

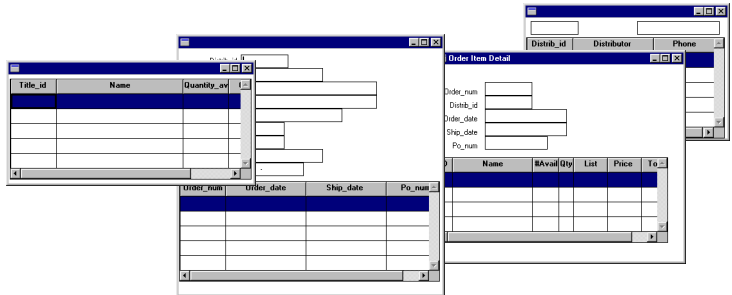
The tutorial is organized into four modules—preparing the client and the server, creating and testing screens, connecting the screens, and extending the application—plus a wrap-up, designed to be completed in sequence. An introduction precedes each module, describing the basic concepts you'll learn and what you can expect to accomplish as a result of completing the module. Each module includes from one to five lessons—for a total of 16 lessons.

You will start by setting up the three-tier environment. In practice, a project leader would probably set this up for the development team—the intent here is for you to become familiar with the process of setting up a three-tier development environment.

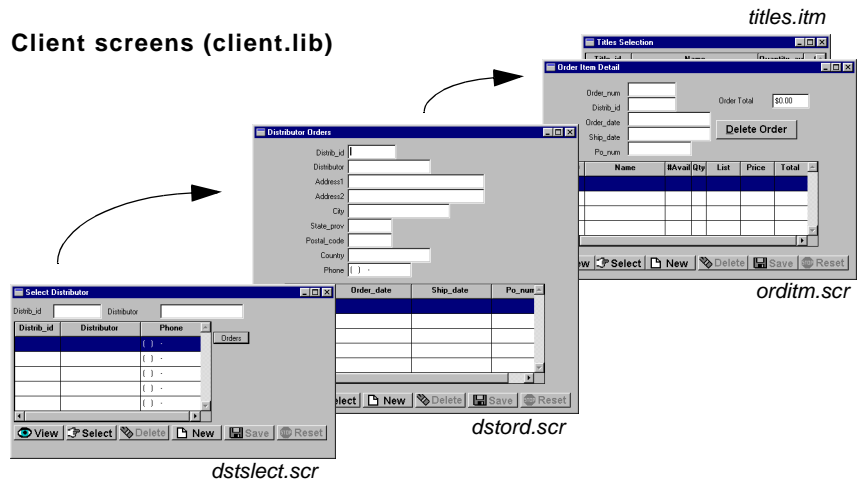
Once you have the environment set up, you can begin building your application—creating three screens that implement some of the functions used to view and update video distributors and their orders.

The tutorial provides you with a JDB database containing distributor and order data. It also includes a tutorial library, which contains the client screens, service components, and JPL modules you'll use in the lessons.

Service containers (server.lib)



Client screens (client.lib)



Accessing Lessons in the Tutorial

You can click on any of the lessons in the following figure to access that lesson.



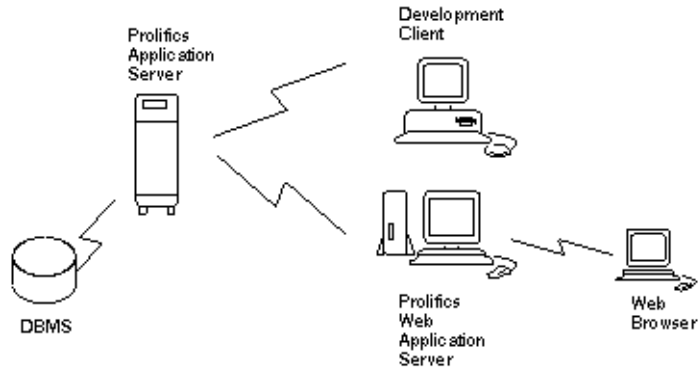
Figure 0-1 Lessons in the JetNet Tutorial

About Each Module

Module 1—Preparing the Server and the Client

Module 1 consists of five lessons that guide you through the process of setting up the Panther software environment to run a three-tier JetNet or Oracle Tuxedo application. In practice, a project leader would probably set this up for the development team—the intent here is for you to become familiar with the process of setting up a three-tier development environment.

In the first four lessons, you start up the application and its servers, start up a client, save screens to their appropriate client and server libraries, define a service in the JIF, and then try out the client screen to test the connection between client and server. In the fifth lesson, you set up a Panther web application server and test the same client screen using a web browser.



On completing this module, you will have a client screen that issues service requests to a Panther application server and to a Panther web application server.

In the process of setting up the environment, you are introduced to the following concepts:

Three types of servers are supported by Panther:

- standard servers, which advertise services defined in the JIF (an interface file containing service information used in client/server processing) and perform remote processing of reports during runtime
- file access servers, which provide the development team shared access to libraries and repositories across the network, and also provide file transfer services during runtime
- conversion servers, which service three-tier applications that have been converted from a two-tier architecture.

All Panther components, such as screens, menu bars, and so forth, reside in libraries. The three standard libraries, `client.lib`, `server.lib`, and `common.lib`, contain objects used by clients, servers, and both, respectively.

Panther applications depend on a number of environment and/or setup variables. These describe the operating environment—for example, the layout of your system and the terminal you are using. They also point to the Panther software installation and specific setup files, and other files required by the application such as libraries. Setup files can also store variables that control the behavior of a Panther application and how users interact with it—for example, message display, cursor behavior, and numeric format.

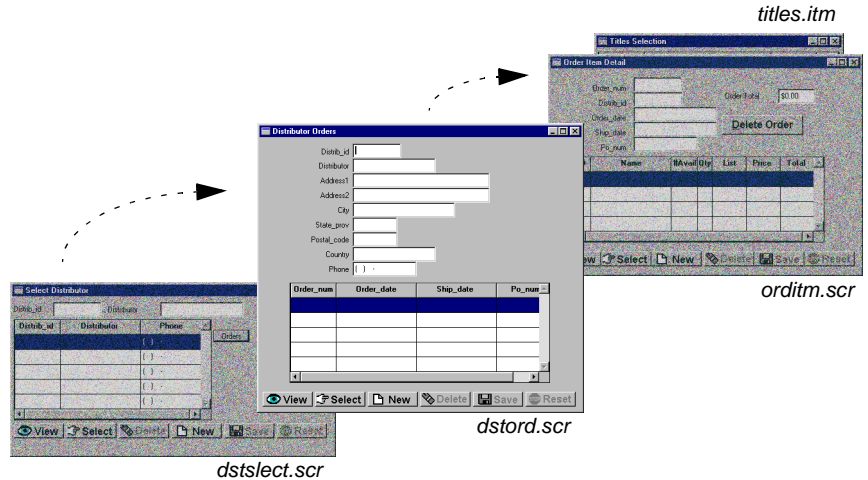
- JetMan, the graphical JetNet manager, provides all the facilities you need to configure and manage Panther's JetNet middleware. The utilities `rbboot` and `rbshutdown` also provide the ability to quickly start up and shut down a Panther application.
- All services must be defined in the JIF—an interface file that stores information about services used by an application.

Panther accesses the JIF whenever it needs to determine the requirements and specifications of a service. You create and edit JIF files using the JIF editor.

Module 2—Creating and Testing Screens

Module 2 consists of five lessons that start you on the process of building your application—one that will implement the functions used to view and update a list of video distributors and their orders. In these lessons, you:

- Create a repository and populate it with imported database objects from the `vidsales` database provided with the tutorial.
- Use the screen wizard to create a screen that displays information about the distributors and their orders.
- Define the services needed to view, update, and add order and distributor information in the database.
- Test the behavior of your screen.
- Assign properties to the screen objects, and enhance the screen to query the database using those properties.



When you complete the lessons in this module, you will have a client screen that lets you query the database for specific distributors. In the process of creating the screen, you are introduced to the following concepts:

- The use of a remote repository for storing database-derived objects used in your application allows the screen wizard to build screens.
- The screen wizard guides you through the process of creating screens, prompting you for basic design information that it uses to build fully functional screens.
- Whenever you create a service component that has service routines, you need to define the service it will use in the JIF—an interface file that stores information about services used by an application.

Panther accesses the JIF whenever it needs to determine the requirements and specifications of a service. You create and edit JIF files using the JIF editor.

- Test mode allows you to test screen attributes and logic, including data validation, database interactions, and client/server connections. In test mode, your client screen appears and behaves as it would in the final application.
- The transaction manager knows about the interaction between database tables and columns from information retrieved from the database during the import process. It automatically builds the appropriate SQL statements, keeps track of

data changes, and knows when to activate or deactivate specific widgets on the client screen.

- Each object in your repository has a set of properties that define its visual and behavioral attributes, as well as database definitions if the object was derived from a database. These properties can be used as the basis of a query on the database.

Module 3—Connecting the Screens

Module 3 consists of four lessons that show you how to improve the appearance, capabilities, and flow of your application. In these lessons, you:

- Enhance a client screen and its service component by allowing users to search for distributors by name or ID.
- Apply global and local changes to application objects.
- Write several JPL procedures to send and receive data between screens and to and from the database.
- Add other capabilities, such as double-click events, to the user interface.

When you have completed the lessons in this module, you will have a second screen, which you will use to query the database for distributors using either their name or ID, and to invoke the screen you created in Module 2.

In Lesson 13, you add JPL procedures to your screens. These procedures are provided in `tutorial.lib`, and their contents are described in the lesson. If you have additional questions about the code, you can refer to the comments in the actual JPL procedure, since the comments are not included in the lesson text.

In the process of creating the query screen, you will be introduced to the following concepts:

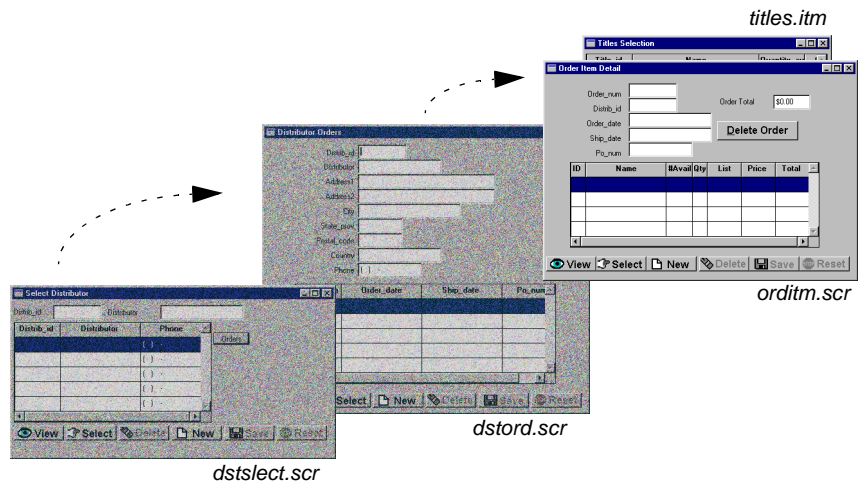
- The editor provides a graphical environment for enhancing screens that were previously created with the screen wizard, allowing you to alter the size, position, properties, and so forth, of both the screens and the objects contained in them.
- The inheritance link between a repository and application objects allows you to use the repository to update the client screens or service components by

propagating changes from parent objects in the repository to objects in your application screens.

- JPL procedures can be attached to screens and widgets to perform such functions as sending or receiving data between screens, performing error handling, and providing the transaction manager with additional instructions for handling database transactions.
- A variety of ways exist to enhance the usability of your application, such as implementing double-click events, having commands or procedures executed only under certain conditions, and so forth.

Module 4—Extending the Application

Module 4 consists of two lessons that guide you through the process of adding a third screen to your application. The screen and its service component, an order entry interface, are created using the screen wizard. It allows the user to add a new order and update existing ones for a selected video distributor.



In the process of developing this portion of the application, you will be introduced to the following concepts:

- Using the screen wizard to create selection screens—pick lists that display acceptable database values for a field. Selection screens are helpful when you need to add a new record to the database.

- Calculating data from database values. You will extend the use of the database by adding a virtual column to a table view and thereby include the widget in the automated SQL generation.
- Including a virtual widget in a table view also allows the transaction manager to apply the same styles to the widget as to the other widgets belonging to the table view.
- Using three-tier processing to calculate values on the server using transaction manager hook functions.
- Deleting a single detail record on a master-detail screen, instead of deleting the master and all of its detail records.
- Implementing a validation procedure on the client screen that recalculates totals when an update, delete, or insert operation is indicated.

Hints for Completing the Tutorial

- This tutorial is meant to be completed sequentially—in each lesson, you'll build on the results of the previous lesson. If you jump ahead, you might not have the data or screens you need to perform the tasks being covered in that lesson.
- Whenever you are done using the application—at the end of the day or if you want to continue the tutorial later—remember to shut down the application servers. This is just good practice, since it frees up system resources that may be required by others. Whenever you want to resume, make sure you're in the appropriate directory, set up the environment (by applying the setup file for UNIX users), and boot the application. This procedure is described at the end of Lesson 1.

Before You Start

Before you begin the tutorial, you should be aware of the following requirements:

- You need to know the directory and path name where Panther software is installed.
- The client portion of Panther runs either on a UNIX workstation or under Windows, and instructions are given for both platforms. The illustrations in each lesson represent the Window version.
- If you are running the client under Windows, make sure the Panther `Samples\Tutorial` directory has been installed on your PC.
- For UNIX, it is assumed for this tutorial that you are running under either the Korn or Bourne shell.
- If you plan to set up a Web application server (in Lesson 5; optional), you need to have a Web browser and HTTP server available. On the HTTP server, you need to know the location of its CGI directory and the location of the Panther installation.

For more information on configuring systems to run the tutorial, refer to Appendix A, “Setting Up the Tutorial.”

For More Information...

As you're progressing through the tutorial, you might find that you have questions about a specific area of Panther; or you may just want to learn more about a topic after completing the tutorial. Listed below are places you can look for more information online.

For Information on	Refer to This Manual
Configuring your application server and your environment	<i>JetNet /Oracle Tuxedo Guide</i>
Developing your application; using Panther software components; using two- and three-tier architectures, using the JPL scripting language	<i>Application Development Guide</i>
Developing your application for the web	<i>Web Developerment Guide</i>
Using the editor, screen wizard, and JIF editor	<i>Using the Editors</i>
Using a command or function	<i>Programming Guide</i>

For More Information...

Module 1— Preparing the Server and the Client

OVERVIEW

1 Setting Up the Server

The first step in preparing the three-tier Panther software environment for JetNet and Oracle Tuxedo applications is to set up the application server. This includes creating the directory where you run the tutorial, populating it with the appropriate files, setting some environment variables, copying or linking to the server executables, and then starting the application.

In this lesson you learn how to:

- Create an application directory and copy to it the libraries, environment files, and other files needed for the tutorial.
- Edit the environment files used by the server, client, and middleware.
- Copy or link to the server executables.
- Create and boot a Panther application with JetMan, the JetNet manager utility.

A Panther application server can run either on UNIX or Windows; instructions are given for both.

- [UNIX Application Server \(page 1-2\)](#)
- [Windows Application Server \(page 1-16\)](#)

UNIX Application Server

If you are setting up the server on UNIX, it is assumed that you are running under either the Korn or Bourne shell.

Before beginning this lesson, you need to know the following:

- The location of your Panther application server installation. (The default location is `/usr/prolifics`.)

Panther is installed at:_____

- The location of your license file. (The default location is `/usr/prolifics/licenses/license.dat`.)

The license location is:_____

Create an application directory

For the tutorial, it is recommended that you create the application directory as a local directory under your home directory. You will work in that directory, copying all the necessary files for the tutorial application to that location. In practice, the application directory is located in a central location that is accessible to the development team.

- 1 Log onto your application server machine.
- 2 Create an application directory. For the purposes of this tutorial, create the directory in your home directory and call it `proltut`. At the command line, type:

```
mkdir proltut
```

- 3 Change to the `proltut` directory:

```
cd proltut
```

- 4 Copy the file `setup.sh` from the Panther server installation's `config` directory to the `proltut` directory. This file contains environment setup information that is required by Panther, such as path names and terminal information. Usually, a copy of this file resides in the application directory and in each developer's working directory.

```
cp ProInstallDir/config/setup.sh .
```

where *ProInstallDir* is the full pathname to the Panther application server installation (which you noted on [page 1-2](#)). It is `/usr/prolifics` by default.

Edit the environment setup file

You set the Panther environment on a UNIX server with the `setup.sh` file. Edit your copy of this file to include information needed by Panther such as location of the installation and license file.

- 5 Open up the `setup.sh` file using an editor. Make the following changes:
 - Note that `SMBASE` is set to `/usr/prolifics` by default. If Panther is installed at another location, then set `SMBASE` to the full path name of the correct directory (as noted on [page 1-2](#)).
 - Check the location of the `license.dat` file. If it is not in the default directory (`$SMBASE/licenses/license.dat`), set the `LM_LICENSE_FILE` variable to the license file's full path name (as noted on [page 1-2](#)). Uncomment this setting and its export line.
 - Uncomment the `SMTERM` line and set it to the appropriate terminal type. This variable tells Panther what console type and model you are using. For example, if you are running under Motif, set `SMTERM=X`. If you are running under UNIX in character mode, refer to the Configuration for information on using video files, which tell Panther how to drive the terminal display.
 - Uncomment the shared library path variable setting that is appropriate to your UNIX system. Also uncomment the corresponding export line. For example, for Solaris systems, uncomment these lines:

```
LD_LIBRARY_PATH=$SMBASE/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

- 6 Save and close the `setup.sh` file.
- 7 At the command line, type:

```
.. /setup.sh
```

This applies the settings in `setup.sh`.

Get new application components

The Panther `newapp` directory contains three standard application libraries and three server environment files; all must be copied to the `proltut` directory. The libraries will include all components used by the application such as screens and menus. The three environment files provide environment settings for Panther servers.

- 8 Copy all files from the Panther `newapp` directory to the application directory (`proltut`):

```
cp $SMBASE/samples/newapp/* .
```

This copies the libraries (`client.lib`, `server.lib`, `common.lib`) and the server environment files (`progserv.env`, `proserv.env`, `machine.env`) to the `proltut` directory.

More About Libraries

All Panther application components such as screens and menus reside in libraries. The three standard application libraries listed below are installed with Panther software; you can add other libraries to your application as needed.

- `client.lib`—client library that contains components of the user interface, such as screens, menus, toolbars, JPL modules (Panther’s scripting language) used by the client, a styles file, and images that illustrate push buttons and toolbar items.
- `server.lib`—server library that contains server components (graphical representations of server functions) and routines; this library is required only for developing and deploying three-tier applications.
- `common.lib`—common library that contains components needed application-wide by both client and server, such as the JIF (JetNet Interface file) and JPL code.

Get tutorial components

The Panther tutorial directory includes a tutorial-specific library and the JDB database `vidsales`. (JDB is a single-user SQL database provided with Panther; it is described in Lesson 6.) You need to copy these to the application directory.

- 9 Copy all files from the tutorial directory to your `proltut` directory.

```
cp $SMBASE/samples/tutorial/* .
```

This copies the tutorial library (`tutorial.lib`) and the JDB database (`vidsales`) to the `proltut` directory.

Define the server environment

In the tutorial, you run executables that are associated with two server types: `proserv` for standard servers, and `devserv` for file access servers. When these servers are activated, they read the environment files `proserv.env` and/or `machine.env`.

Edit the environment files (use any ASCII text editor such as `vi`) so that each server can find the Panther installation and knows which libraries to open.

- 10 Open up the `machine.env` file and edit the following:

- Uncomment `SMBASE` (remove the `#` from the beginning of the line) and provide the full path name of the Panther installation (as noted on [page 1-2](#)) if it is not `/usr/prolifics`.

```
SMBASE=ProInstallDir
```

- Uncomment the shared library path variable setting that is appropriate to your UNIX system.
- Update the license file location if it is not in the default location.

- 11 Save and close the file.

Link to the server executables

You need to create local links in the `proltut` directory to the server executables provided with Panther. These executables are located in the distributed `util` directory.

- 12** At the command line, type:

```
ln -s $SMBASE/util/proserv .
```

This creates a symbolic link to the server executable `proserv`, which runs a standard server.

- 13** At the command line, type:

```
ln -s $SMBASE/util/devserv .
```

This creates a symbolic link to the server executable `devserv`, which runs a file access server.

More About Panther Servers

Panther supports three types of application servers to handle requests from clients:

- Standard servers, which advertise services defined in the JIF (JetNet Interface File, which contains service information used in client/server processing) and perform remote processing of report during runtime.
- Conversion servers, which provide services for three-tier applications that have been converted from a two-tier architecture.
- File access servers, which provide the development team shared access to libraries and repositories across the network, and also provide file transfer services at runtime.

For more information about servers, refer to the *JetNet/Oracle Tuxedo Guide*.

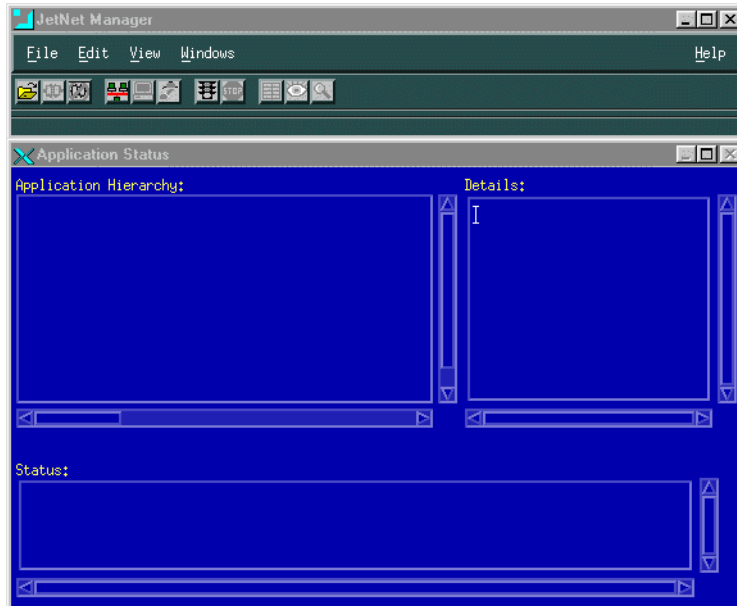
A configuration file is required for Panther's middleware, JetNet, which manages communication between clients and servers. You create this file through JetMan, the JetNet manager.

Note: In the Oracle Tuxedo version, JetNet configuration files can be used with—and are accessible to—all Oracle Tuxedo utilities, including `xtuxadm`. Thus, you can use the JetNet manager to create a configuration file, then edit and enhance it for use with Oracle Tuxedo later on. The alternative option is to use Oracle Tuxedo's own configuration files; in which case you would not use JetNet at all.

- 14** Start up the JetNet manager in the same window where you set the tutorial environment:

```
jetman
```

The opening screen of the JetNet manager displays:

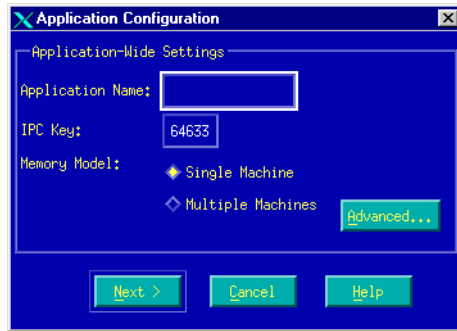


Create a middleware configuration file

JetMan integrates all the facilities you need to configure and manage the middleware component of a Panther application. With it, you create and edit a binary Jet Net configuration file—by default, `broker.bin`—in the current directory.

- 15 Choose File→New→Application.

JetMan displays the Application Configuration dialog:



Name the application

All components of a Panther application—servers, clients, and services—are identified to your system through the name that you enter in the configuration file's Application Name property.

- 16** In the Application Name property, enter Tutorial as the name of your application. Leave all other properties unchanged.
- 17** Choose Next. The JetNet manager displays the Machine Configuration dialog, where you can configure the server machine properties as desired.



Get the machine's name and configuration file

The Machine Configuration dialog contains two properties that you use later to set the environment variables `SMRBHOST` and `SMRBCONFIG`:

- 18 The Name property is set to the host machine—that is, the machine that is running the JetNet manager. Make a note of the Name property (listed under General Information) for later reference:

Host Machine Name: _____
(SMRBHOST)

- 19 The Local JetNet Configuration File property contains the name and location of the new application's configuration file:

`appDirectory/proltut/broker.bin.`

Make a note of this property setting for later reference:

Local JetNet Config File: _____
(SMRBCONFIG)

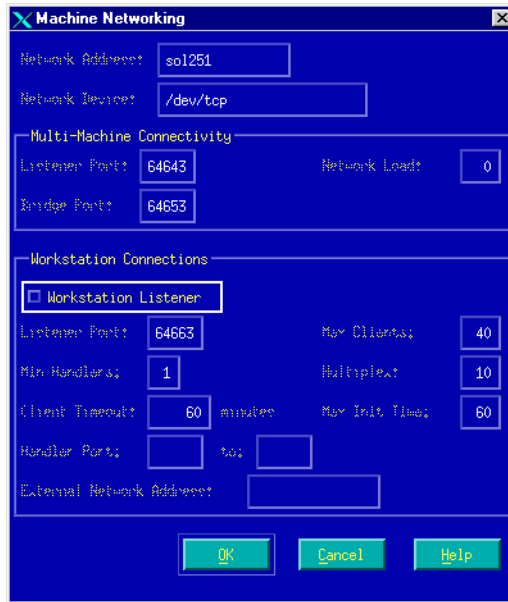
Get the machine's port number

It is likely that you use your own workstation to develop applications and to run JetNet manager for most application configuration tasks. In both cases, a workstation establishes a client connection to the server machine via the machine's name and workstation listener port.

You obtained the host machine's name in the previous section; now get the value of its workstation listener port, which you use later to set the client environment variable SMRBPORTR.

20 From the Machine Configuration dialog, choose the Networking push button.

The Machine Networking dialog displays:



21 Select the Workstation Listener check box on page 1-11

22 Copy the Listener Port number (under the Workstation Connections section).

Listener Port: _____
(SMRBPORTR)

23 Choose OK to return to the Machine Configuration dialog.

- 24** Choose Done to save the new configuration file.

It takes a few moments as the JetNet manager creates a configuration file for the Tutorial application.

- 25** When it completes, choose File→Exit, and choose Yes to the closing message.

Provide the configuration file's location

The name and location of the configuration file `broker.bin` must be defined in the server environment through the `SMRBCONFIG` variable:

- 26** Set the `SMRBCONFIG` variable:

Edit `setup.sh` to include and export the location of `broker.bin`. Add the variable and the value for `SMRBCONFIG` exactly as it is displayed in the JetNet manager (as you recorded in step 22).

```
SMRBCONFIG=appDirectory/proltut/broker.bin
export SMRBCONFIG
```

Save and close the file.

Note: The value in `SMRBCONFIG` must match the value in JetMan. If your home directory is a symbolic link to another disk location, the disk location that appears in JetMan must be entered, not the value displayed with the UNIX command `pwd`.

Boot the application

Now you can start your application with the JetNet manager. Run the following utility each time you want to start up your Panther application. Make sure you are in the Tutorial application directory (`proltut`) and follow these steps: on page 1-15

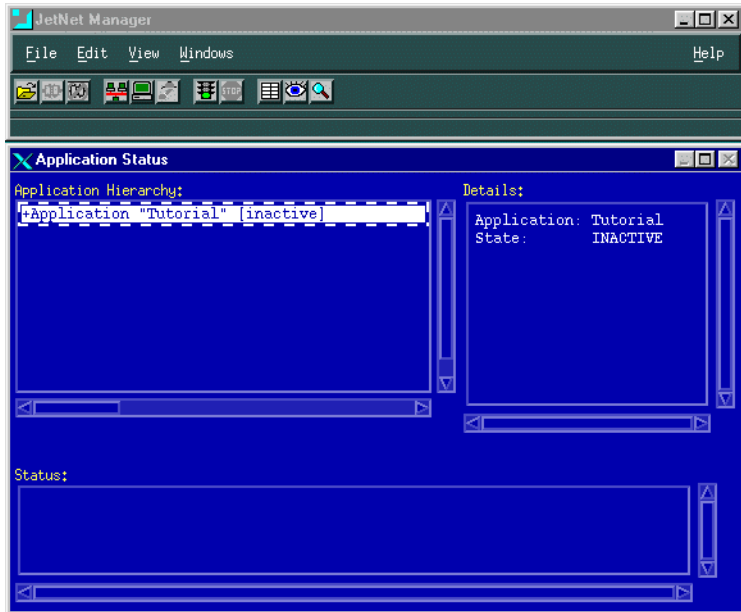
- 27** From the server command line, reestablish the environment that the Tutorial application requires by entering this command:

```
. setup.sh
```

- 28** Enter this command:

```
jetman
```

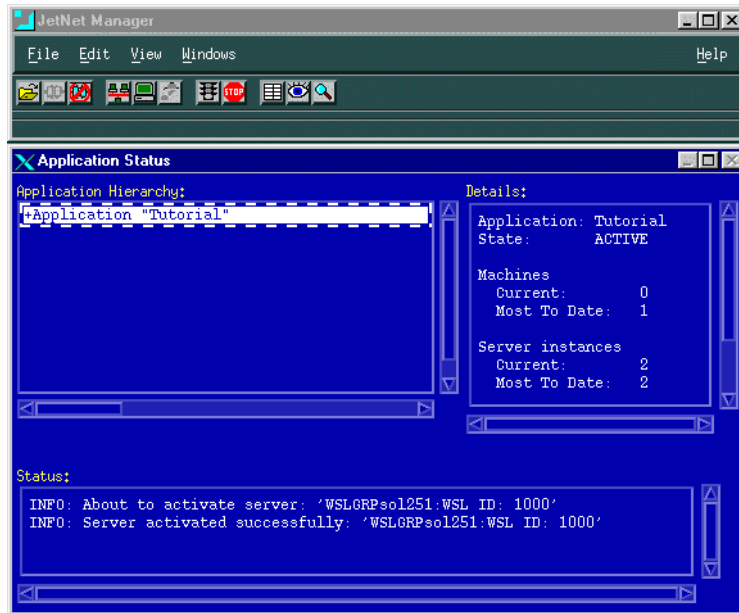
The Application Status dialog opens.



29 If Application “Tutorial” is not already highlighted, select it by choosing File→Select Application.

30 Choose Edit→Activate from the menu bar.

This starts the application. The status window indicates success or failure for each attempt to activate an application server. A successful boot of all servers starts your Panther application.

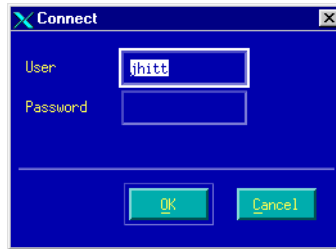


Note: If you have problems starting the application, refer to Appendix B on [page B-1](#). Before trying to restart the application, first shut it down by selecting the Application item from the hierarchy list and choosing Edit→Deactivate. This shuts down all components of the Panther application, including any active servers.

Connect to the application

To view all components of an active application, you must connect to the JetNet manager as a client. If your user name is set in your environment, the connection window displays it.

31 Enter your user name, if it is not already entered, and choose OK.



The information required by the Connect dialog differs for native and workstation clients:

- The dialog for workstation clients asks for the server machine's host name and port.
- The dialog for a native client (shown above) asks only for login information.

Note: In this tutorial, user name and password are not needed; however, a deployed application typically requires, at a minimum, this level of security. You can set a password in the JetNet manager through the application's Application Password property (refer to [page 3-12](#) in *JetNet/Oracle Tuxedo Guide*).

All components of the tutorial application now display in the Application Status window.

More About Panther Utilities

Instead of using the Jet Net manager to create and manage an application, you can use one of several utilities to perform basic tasks:

- `rbconfig` creates a minimal middleware configuration file (`broker.bin`) that you can use as a starting point for further development.
- `rbboot` activates a Panther application.
- `rbshutdown` deactivates a Panther application.

If you take a break

Whenever you are done using the application—at the end of the day or for an extended break—remember to shut down the application. This is good practice, since it frees up system resources that others might require.

- 32** Select the Application item from the Application Hierarchy list and choose Edit→Deactivate from the menu bar on the Deactivate button from the toolbar.

This shuts down all components of the Panther application, including any active servers.

To resume the tutorial

When you want to resume the tutorial (or run your application), repeat the following procedures (described in Steps 27 through 31 above):

- Establish the environment by running `setup.sh`.
- Start `jetman`.
- Select the application, if it is not already highlighted. (If the application is not listed, you need to open the `broker.bin` file. Choose File→Select Application.)
- Choose Edit→Activate.
- Open a middleware connection.

To continue the tutorial

To skip to the Lesson 1 summary, refer to [page 1-29](#).

Windows Application Server

Before beginning this lesson, you need to know the following:

- The location of your Panther application server installation. (The default location is `C:\Prolifics\Panther`; this is the location referenced in the steps in this tutorial.)

Panther is installed at: _____

- The location of your license file. (The default location is `C:\Prolifics\Panther\licenses\license.dat`.)

The license location is: _____

Create an application directory

For the tutorial, it is recommended that you create the application directory as a new folder under your root directory. You will work in that directory, copying all the necessary files for the tutorial application to that location. In practice, the application directory is located in a central location that is accessible to the development team.

- 1 Create an application directory. For the purposes of this tutorial, create a new folder in the root directory (for example `C:\`) and name it `proltut`.

Get new application components

The Panther `newapp` directory contains three standard application libraries and three server environment files; all must be copied to the `proltut` directory. The libraries will include all components used by the application such as screens and menus. The three environment files provide environment settings for Panther servers.

- 2 Copy all files from the `C:\Prolifics\Panther\Samples\newapp` directory to the application directory (`C:\proltut`). This copies the standard application libraries (`client.lib`, `server.lib`, `common.lib`) and the server environment files (`progserv.env`, `proserv.env`, `machine.env`) to the `proltut` directory.

More About Libraries

All Panther application components such as screens and menus reside in libraries. The three standard application libraries listed below are installed with Panther software; you can add other libraries to your application as needed.

- `client.lib`—client library that contains components of the user interface, such as screens, menus, toolbars, JPL modules (Panther's scripting language) used by the client, a styles file, and images that illustrate push buttons and toolbar items.
- `server.lib`—server library that contains server components (graphical representations of server functions) and routines; this library is required only for developing and deploying three-tier applications.
- `common.lib`—common library that contains components needed application-wide by both client and server, such as the JIF (JetNet Interface file) and JPL code.

Get tutorial components

The Panther tutorial directory includes a tutorial-specific library and the JDB data base `vidsales`. (JDB is a single-user SQL database provided with Panther; it is described in Lesson 6.) You need to copy these to the application directory.

- 3 Copy all files from the `C:\Prolifics\Panther\Samples\Tutorial` directory to your `C:\proltut` directory.

This copies the tutorial library (`tutorial.lib`), the JDB database (`vidsales`), and two initialization files to the `proltut` directory.

Define the server environment

In the tutorial, you run executables that are associated with two server types: `proserv.exe` for standard servers, and `devserv.exe` for file access servers. When these servers are activated, they read the environment files `proserv.env` and/or `machine.env`.

Edit the environment files (use any ASCII text editor such as Notepad) so that each server can find the Panther installation and knows which libraries to open.

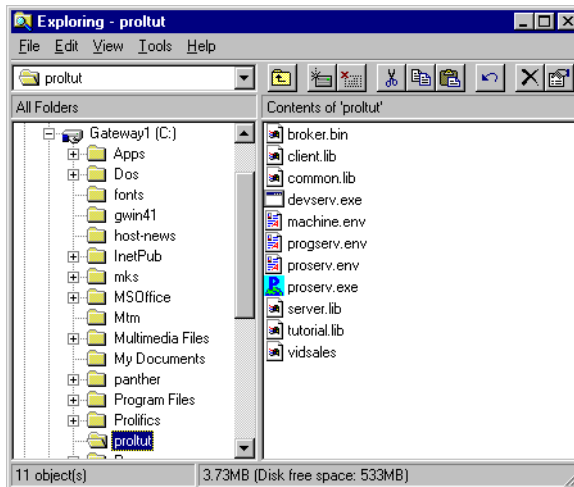
- 4 Edit the `SMBASE` setting in the `machine.env` file. It should be set to the Panther installation directory (as noted on [page 1-16](#)).

- 5 Check the location of the license.dat file. If it is not in the default directory (\$SMBASE\licenses\license.dat), set LM_LICENSE_FILE to the license file's full path name (as noted on [page 1-16](#)).

Copy the server executables

In order for your server executables to be able to read the application's settings in the environment files, you need to copy the server executables from Panther's util directory to the tutorial's application directory.

- 6 Copy proserv.exe and devserv.exe from the C:\Prolifics\Panther\util directory to the application directory (C:\proltut).



More About Panther Servers

Panther supports three types of application servers to handle requests from clients:

- Standard servers, which advertise services defined in the JIF (JetNet Interface File, which contains service information used in client/server processing) and perform remote processing of report during runtime.
- Conversion servers, which provide services for three-tier applications that have been converted from a two-tier architecture.

- File access servers, which provide the development team shared access to libraries and repositories across the network, and also provide file transfer services at runtime.
- For more information about servers, refer to *JetNet/Oracle Tuxedo Guide*.

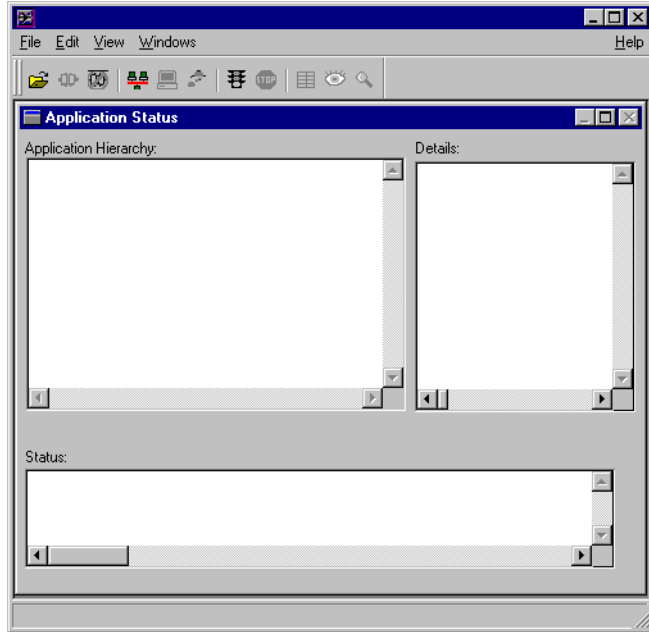
Start the JetNet manager

A configuration file is required for Panther's middleware, JetNet, which manages communication between clients and servers. You create this file through JetMan, the JetNet manager.

Note: In the Oracle Tuxedo version, JetNet configuration files can be used with—and are accessible to—all Oracle Tuxedo utilities, including `xtuxadm`. Thus, you can use the JetNet manager to create a configuration file, then edit and enhance it for use with Oracle Tuxedo later on. The alternative option is to use Oracle Tuxedo's own configuration files; in which case you would not use JetNet at all.

- 7 Start up the JetNet manager from the Start Menu by choosing Programs→Panther Application Server→Utilities→Local JetMan.

The opening screen of the JetNet manager displays:



Create a middleware configuration file

JetMan integrates all the facilities you need to configure and manage the middleware component of a Panther application. With it, you create and edit a binary Jet Net configuration file—by default, `broker.bin`, in the current directory.

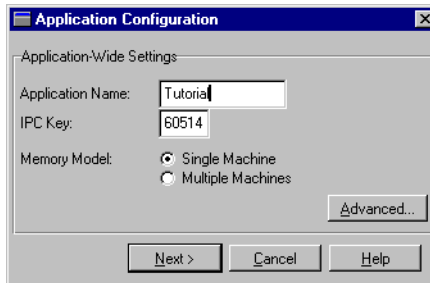
- 8 Choose File→New→Application.

JetMan displays the Application Configuration dialog.

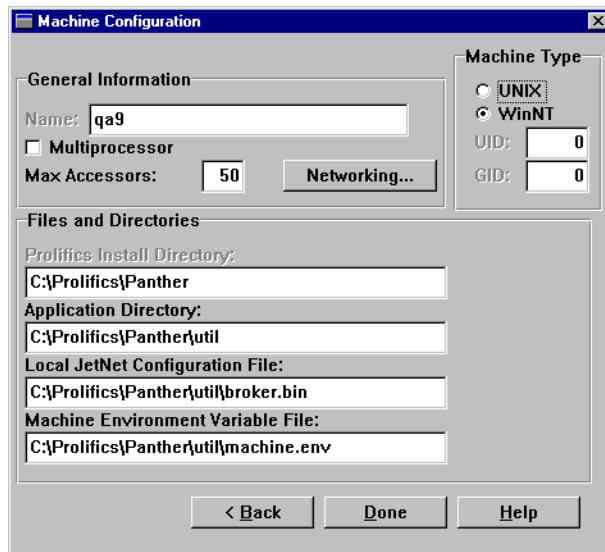
Name the application

All components of a Panther application—servers, clients, and services—are identified to your system through the name that you enter in the configuration file's Application Name property:

- 9 In the Application Name property, enter Tutorial as the name of your application. Leave all other properties unchanged.



- 10 Choose Next. The JetNet manager displays the Machine Configuration dialog, where you can configure the server machine properties as desired.



Get the machine's name and configuration file

The Machine Configuration dialog contains two properties that you use later to set environment variables `SMRBHOST` and `SMRBCONFIG`:

- 11 The Name property is set to the host machine—that is, the machine that is running the JetNet manager. Make a note of the Name property (listed under General Information) for later reference:

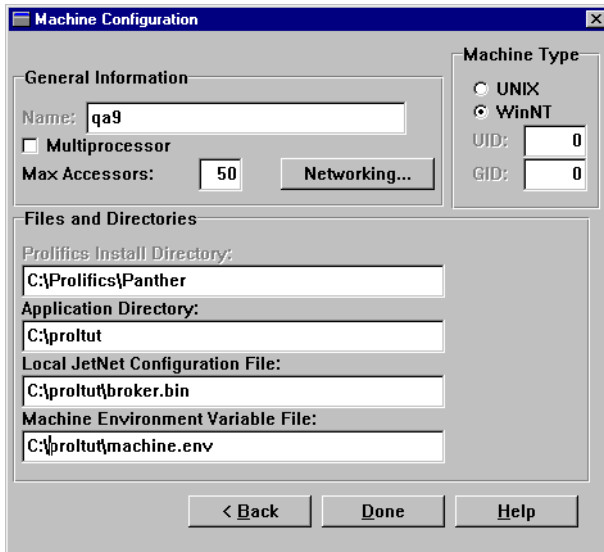
Name : _____
(SMRBHOST)

- 12 Change the Application Directory setting to the tutorial's application directory (C:\proltut).

- 13 The Local JetNet Configuration File property contains the name and location of the new application's configuration file. Change this line to specify a path to broker.bin in the proltut directory (for example, C:\proltut\broker.bin). Make a note of this property setting for later reference:

Local JetNet Configuration File: _____
(SMRBCONFIG)

- 14 The Machine Environment Variable File contains the name and location of the machine configuration file to use for the application. Change this line to specify a path to machine.env in the proltut directory (for example, C:\proltut\machine.env).



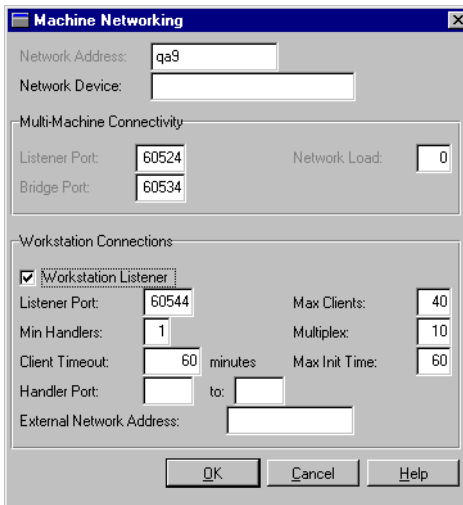
Get the machine's port number

It is likely that you use your own workstation, not the server machine, to develop applications and to run JetNet manager for most application configuration tasks. In both cases, a workstation establishes a client connection to the server machine via the machine's name and *workstation listener port*.

You obtained the host machine's name in the previous section; now get the value of its workstation listener port, which you use later to set the client environment variable SMRBPORTR.

- 15 From the Machine Configuration dialog, choose the Networking push button.

The Machine Networking dialog displays:

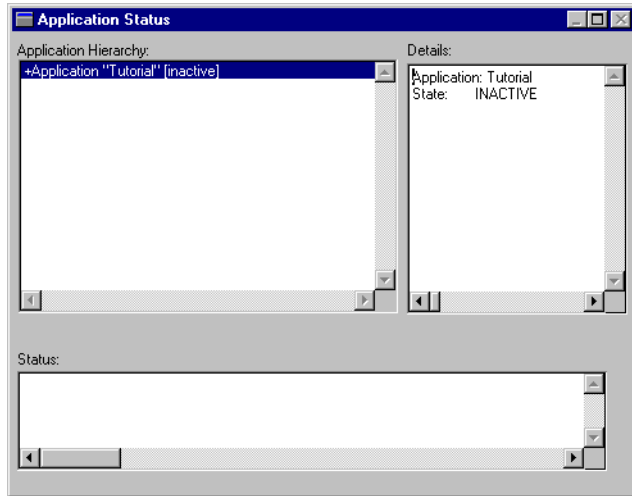


- 16 Select the Workstation Listener check box.
- 17 Copy the Port number (under the Workstation Connections section).

Port : _____
(SMRBPORTR)

- 18 Return to the Machine Configuration dialog by choosing OK.
- 19 Save the new configuration file by pressing Done.

It takes a few moments as the JetNet manager creates a configuration file for the Tutorial application and displays its components in the Application Status dialog:



20 When it completes, choose File→Exit.

Provide the configuration file's location

The name and location of the configuration file `broker.bin` must be defined in the server environment through the `SMRBCONFIG` variable.

21 Set the `SMRBCONFIG` variable:

Open up the file `jetman32.ini` in the Windows directory (`C:\WINDOWS`). Add the variable and the value for `SMRBCONFIG` exactly as it is shown in the JetNet manager (as you recorded above in Step 17).

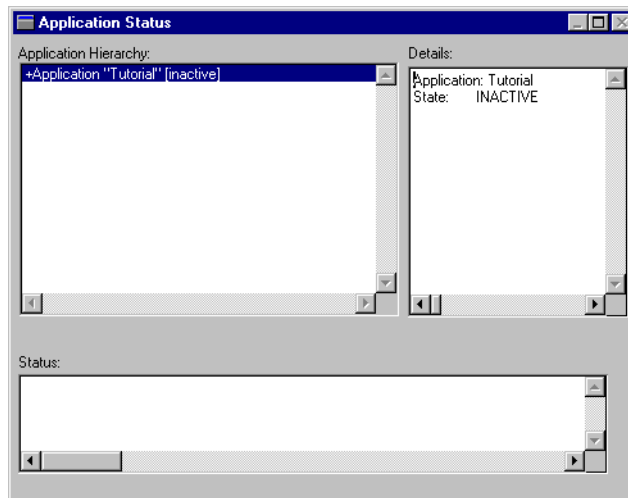
```
SMRBCONFIG=C:\proltut\broker.bin
```

Note: If your Windows server is running a production application and you cannot change `SMRBCONFIG` in `jetman32.ini`, you can use the Select Application option within JetMan.

Boot the application

Now you can start your application with the JetNet manager. Run the following utility each time you want to start up your Panther application.

- 22 Start JetMan.
- 23 If Application “Tutorial” is not displayed, choose File→Select Application. Browse to the `\proltut\` directory, highlight `broker.bin`, and choose Open.
- 24 If Application “Tutorial” is not already highlighted, select it in the Application Hierarchy list.



More About JetNet Manager

If you run the JetNet manager from a native client (one that runs on the same machine as the server), the dialog displays the application but shows incomplete information about its components, as shown above. If you run the utility from a workstation client (one that runs on a different machine from the server), the dialog is initially empty.

- 25 Choose Edit→Activate from the menu bar on the Activate button from the toolbar.

This starts the application. The status window indicates success or failure for each attempt to activate an application server. A successful boot of all servers starts your Panther application.

Note: If you have problems starting the application, refer to Appendix B on page B-1. Before trying to restart the application, first shut it down by selecting the Application item from the hierarchy list and choosing Edit→Deactivate. This shuts down all components of the Panther application, including any active servers.

More About Panther Utilities

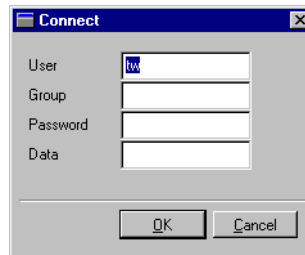
Instead of using the Jet Net manager to create and manage an application, you can use one of several utilities to perform basic tasks:

- `rbconfig` creates a minimal middleware configuration file (`broker.bin`) that you can use as a starting point for further development.
- `rbboot` activates a Panther application.
- `rbshutdown` deactivates a Panther application.

Connect to the application

To view all components of an active application, you must connect to the JetNet manager as a client. If your user name is set in your environment, the connection window displays it.

26 Enter your user name, if it is not already entered, and choose OK.



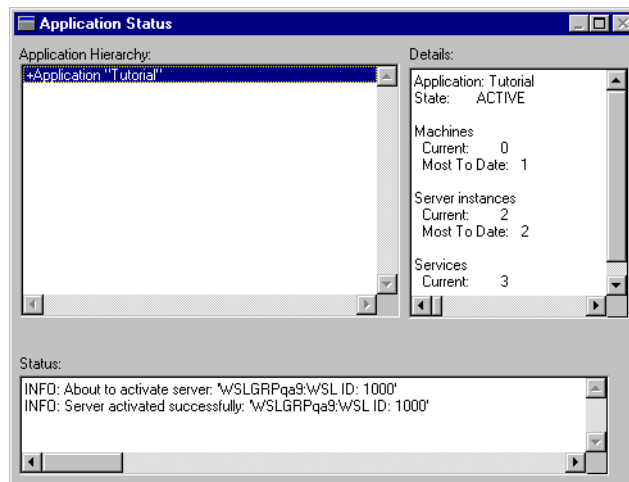
The information required by the Connect dialog differs for native and workstation clients:

- The dialog for workstation clients asks for the server machine's host name and port.

- The dialog for a native client (shown above) asks only for login information.

Note: In this tutorial, user name and password are not needed; however, a deployed application typically requires, at a minimum, this level of security. You can set a password in the JetNet manager through the application's Application Password property (refer to [page 3-12](#) in *JetNet Guide/Oracle TuxedoGuide*).

All components of the tutorial application now display in the Application Status window.



If you take a break

Whenever you are done using the application—at the end of the day or for an extended break—remember to shut down the application. This is good practice, since it frees up system resources that others might require.

- 27 Select the Application item from the Application Hierarchy list and choose Edit→Deactivate from the menu bar on the Deactivate button from the toolbar.

This shuts down all components of the Panther application, including any active servers.

To resume the tutorial

When you want to resume the tutorial (or run your application), repeat the following procedures (described in Steps 22 through 26 above):

- Start up the JetNet manager by choosing Programs→Panther Application Server→Utilities→Local JetMan.
- Select the application, if it is not already highlighted. (If the application is not listed, you need to open the broker.bin file. Choose File→Select Application.)
- Choose Edit→Activate.
- Open a middleware connection.

What did you do?

In this lesson, you performed these tasks:

- Created a Panther application directory, `proltut`, where, for this tutorial, you store all necessary files, perform all server-related tasks and boot your application. In practice, this directory should be in a central location that is accessible to the entire development team.
- Set the server machine environment—path names, location of the Panther installation, libraries, and so forth. The extent of this work depended on whether your server runs on Windows or UNIX.
- Copied the standard application libraries, environment files, tutorial library, and `vidsales` database to the `proltut` directory, and edited the server environment files for your application.
- Provided access to the Panther server executables by creating links in the application directory (in UNIX) or by copying those executables to the application directory (in Windows).
- Created the binary middleware configuration file, `broker.bin`, with the JetNet manager. This also created your application.

- Activated your application with the JetNet manager.

What did you learn?

You learned:

- The application directory is the area where a server machine runs Panther. In general, it contains files that need to be accessed by all development team members, such as server executables, server environment files, and application libraries. Also, each developer has a working directory with local files such as client environment files and local copies of libraries.
- On UNIX, a setup file `setup.sh` contains variable settings, path names, and other information that are required for running Panther software. Each time you want to start up the application, you need to apply these settings before booting.
- The environment variable `SMBASE` is referred to frequently during installation and configuration tasks. It refers to the Panther installation directory and is set in several environment and configuration files. The default location is:

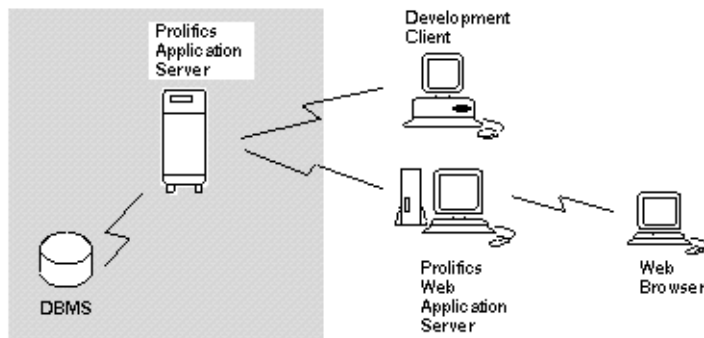
UNIX: `/usr/prolifics`

Windows: `C:\Program Files\Prolifics\Panther`

- The JetNet manager lets you define an application by creating the middleware configuration file `broker.bin`, and view and edit configuration file settings. Some of these settings were applied elsewhere in the setup procedure.
- All Panther components reside in libraries. Three standard libraries (`client.lib`, `server.lib`, and `common.lib`) are provided with Panther; an application component must reside in a shared library to be available to other developers.
- When you want to start your application, then reactivate the application with the JetNet manager (or the `rbboot` utility). For UNIX, make sure you are in the application directory, and set up the UNIX environment with the setup file `setup.sh` before running JetMan.
- When you are done using your application—at the end of the day or if you want to continue the tutorial later—you should shut it down, along with any active servers, with the JetNet manager (or the `rbshutdown` utility).

2 Configuring the Servers

After you activate your Panther application, you need to define and configure its servers. The information you supply to the JetNet manager is added to the binary middleware configuration file, `broker.bin` that you created in Lesson 1.



In this lesson you learn how to:

- Define different types of servers.
- Set server properties.

-
- 1 If you shut down the application and exited the JetNet manager since the previous lesson, reactivate it now and connect to the middleware from the server machine. To review those steps in Lesson 1:

-
- [UNIX \(page 1-15\)](#)
 - [Windows \(page 1-28\)](#)

All components of the tutorial application now display in the Application Status window.

Add servers to the application

You need to add two servers to the tutorial application: a standard server and a file access server.

- 2 Double-click on the application Tutorial to expand the heading.
- 3 Select Machine (this should show the name of your server machine).

Configure the standard server

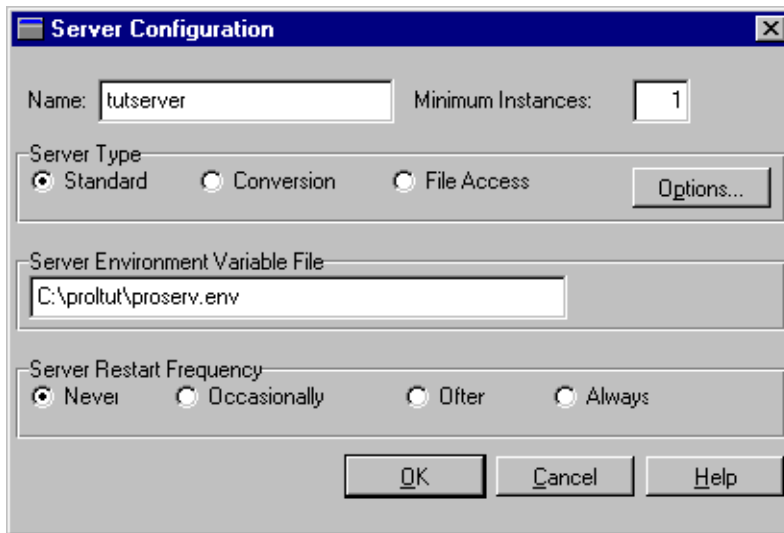
First, add and configure a *standard server*—a server that advertises and executes JIF-defined services, and performs remote processing of reports. After adding this server, you configure it by setting properties that define its functionality.

More About the JIF

The JIF is a file that stores information about services used by your application—that is, functions performed by a server at the request of a client or another server. The JIF tells clients and servers what parameters and information to use when they process service calls. When you create a service component that has service routines, you must also define in the JIF the service that it provides. You learn more about the JIF in lessons 3 and 6.

- 4 Choose File→New→Server.

The Server Configuration dialog box opens.



- 5 Enter `tutserver` as the name of the server.
- 6 Set Minimum Instances to 1. This number determines how many instantiations of this server are created when it is activated. Because you are using Panther's single-user database JDB, in this tutorial this number must be 1; in practice, Minimum Instances is set to a number that is appropriate to the application.
- 7 Choose Standard as the server type (the default selection).
- 8 Choose the Options button.

The Standard Server Details dialog box opens.

Define server properties

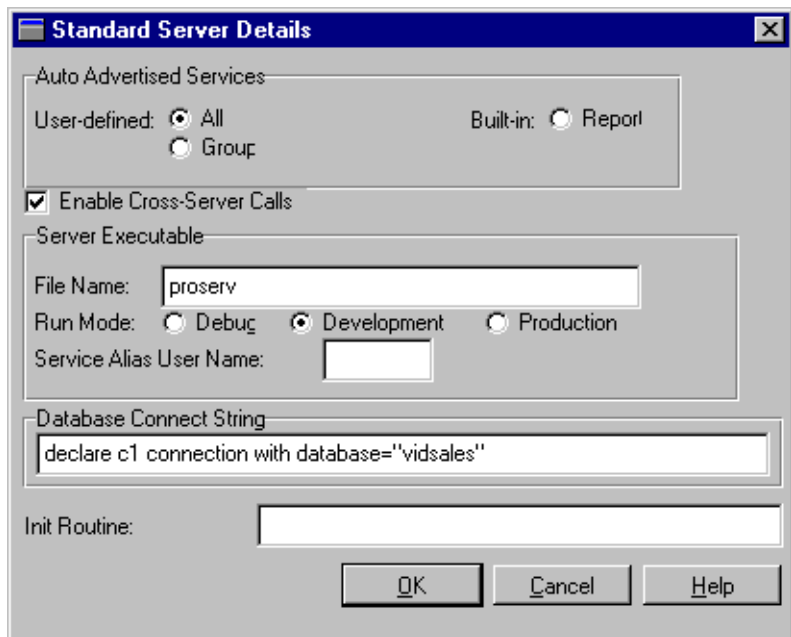
In the Standard Server Details dialog, you set properties that are specific to a standard server, such as which JIF-defined services it advertises, whether it runs in a development or production environment, and which database it connects to.

- 9 Under Auto Advertised Services, choose All. This tells the server to advertise all services defined in the JIF. This choice is useful during development because it makes all services available to the application from any given server.

-
- Under Server Executable, Run Mode, choose *Development* to indicate the mode for the server executable. (When you activate the server, in Step 20, the File Name field is populated with the full path name of the `proserv` executable in the server machine's `pro\lntut` directory.)
 - Enter the database connection string:

```
DECLARE c1 CONNECTION WITH DATABASE="vidsales"
```

When you activate the server, it automatically connects to the specified database `vidsales`.



- Choose *OK*.
The Server Configuration dialog box redisplay.
- Choose *OK*.
The Application Status window redisplay.
- Expand the *Machine* heading in the Application Hierarchy list. It should now include the server `tutserver`.

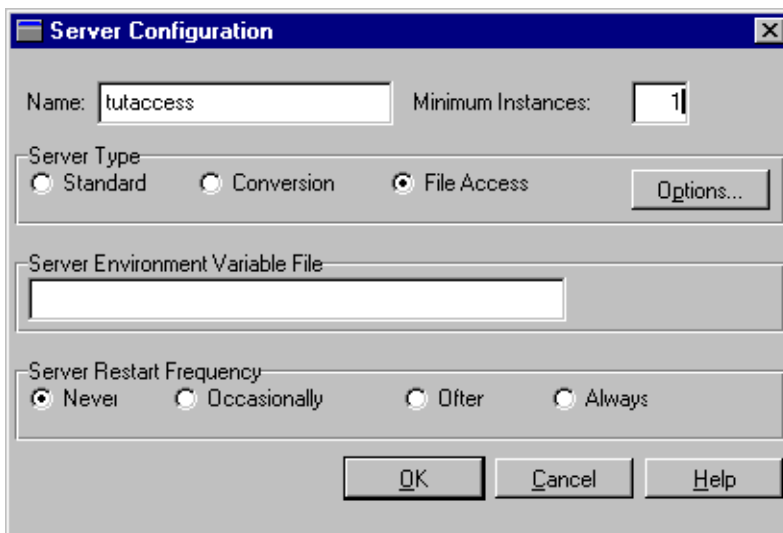
Configure the file access server

Next, you need to add a file access server to the application. This server offers clients shared access to libraries and repositories across the network and provides file transfer services at runtime; it also facilitates remote processing of reports that are transferred back to the client.

- 15 With Machine selected in the Application Hierarchy list, add another server by choosing File→New→Server.

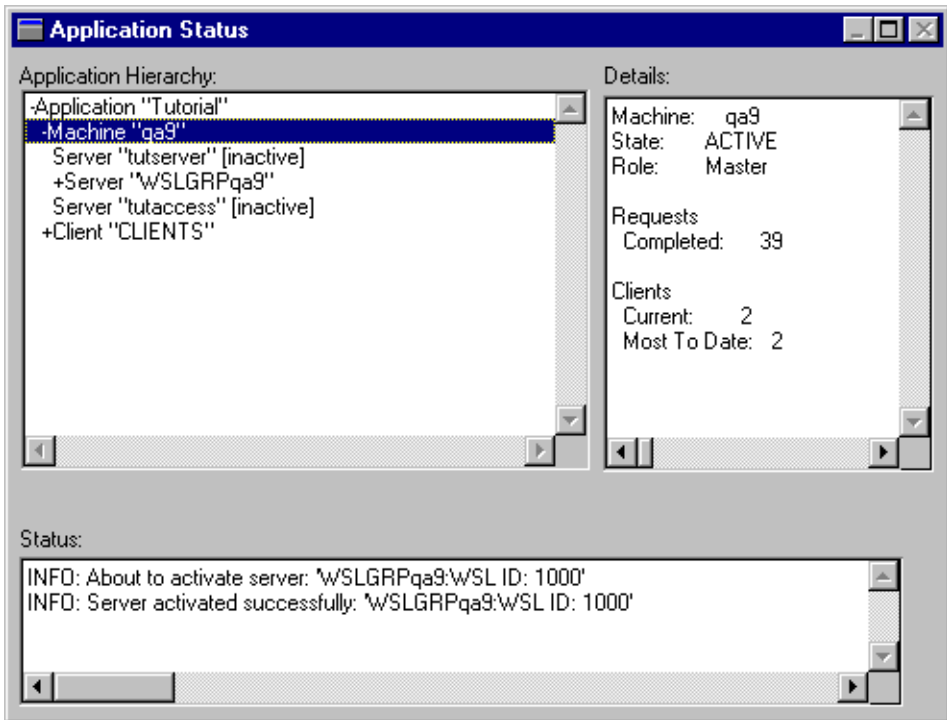
The Server Configuration dialog opens.

- 16 Enter `tutaccess` as the name of the server.
- 17 Set Minimum Instances to 1. This number determines how many instantiations of this server are created when it is activated. Because you are using Panther's single-user database JDB, in this tutorial this number must be 1; in practice, Minimum Instances is set to a number that is appropriate to the application.
- 18 Choose File Access as the Server Type.



- 19 Choose OK.

In the Application Status window, both the `tutaccess` and `tutserver` servers are included in the Application Hierarchy list.



Activate servers

Now start both servers.

20 Select the `tutserver` server in the Application Hierarchy list.

21 Choose Edit→Activate.

The status window indicates the server's successful activation.

22 Select the `tutaccess` server and repeat step 21.

If you have problems activating servers, refer to Appendix B on [page B-1](#).

23 Exit the JetNet manager by choosing File→Exit.

A dialog box opens, prompting you to confirm that you wish to terminate the session.

If you take a break

If you take a break, remember to shut down the application to free up the system resources.

- 24 Select the Application item from the Application Hierarchy list and choose Edit→Deactivate from the menu bar (or the Deactivate button from the toolbar).

What did you do?

In this lesson, you performed these tasks:

- Invoked the JetNet manager to define and configure the servers associated with your application.
- Added and configured a standard server, `tutserver`, which advertises and executes JIF-defined services. During development, all services normally are advertised on a single server and available to developers; in a deployed application, a server usually advertises only a subset of all possible services.
- Added and configured a file access server, `tutaccess`, so developers have shared access to libraries and repositories across the network.
- Activated the standard server and file access server.

What did you learn?

You learned:

- The JetNet manager lets you configure and manage the middleware component of a Panther application. With it, you create and edit the binary middleware configuration file (`broker.bin`), which specifies how to set up an application's clients and servers and configure their interaction—for example, whether

multiple workstations can attach to the application, the maximum number of machines, servers, and services that the application can support, how many servers to activate and on which machine, and so on.

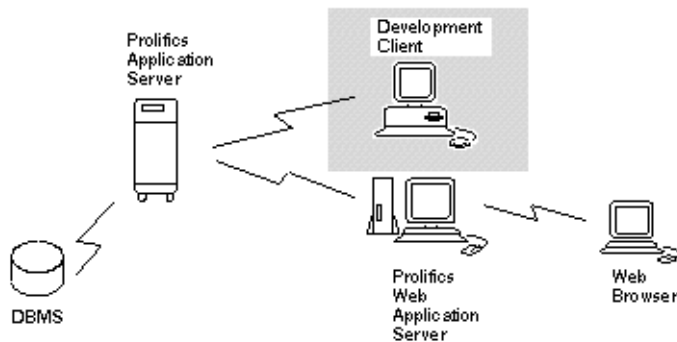
- A Panther standard server advertises and executes JIF-defined services, and also performs remote processing of reports during runtime.

Among the properties you specify for a standard server are:

- The JIF-defined services to advertise.
 - Whether to allow the server to request services from another server.
 - Whether to configure for development or production environments and if it is enabled for debugging.
 - The server's database connection.
 - Any JPL or C Functions to be executed when the server is initialized.
- A Panther file access server offers clients shared access to libraries and repositories across the network. It also provides file transfer services during runtime. File access servers have a single property, which identifies the server by name.

3 Setting Up the Client

The server is configured and the development team is ready to start. Now you need to set up the clients.



In this lesson you learn how to:

- Define the server to your workstation (remote clients).
- Define the libraries (remote and/or local) that should open by default when you start up your Panther client.
- Connect to the middleware which monitors and manages the requests and events that occur in a three-tier environment.
- Save application components to their appropriate libraries.

A Panther client can run either on UNIX or Windows; instructions are given for a native (local) UNIX client and a remote Windows client.

- UNIX Client ([page 3-2](#))
- Windows Client ([page 3-8](#))

For Windows, instructions are given for editing the necessary files for workstation clients to automatically connect to the Panther servers and access shared libraries.

UNIX Client

- 1 If your application server is not running, reactivate it now from the server machine (refer to [page 1-15](#), “To resume the tutorial”).

Apply environment settings

The local UNIX environment must know where and what to access on startup of the Panther client, such as which libraries to open and the location of the server configuration file.

- 2 From the tutorial's application directory (`homeDir/proltut`), open up the `setup.sh` file using an editor. Make the following changes:

- Add `SMFLIBS` to open the libraries needed in the editor and export the setting:

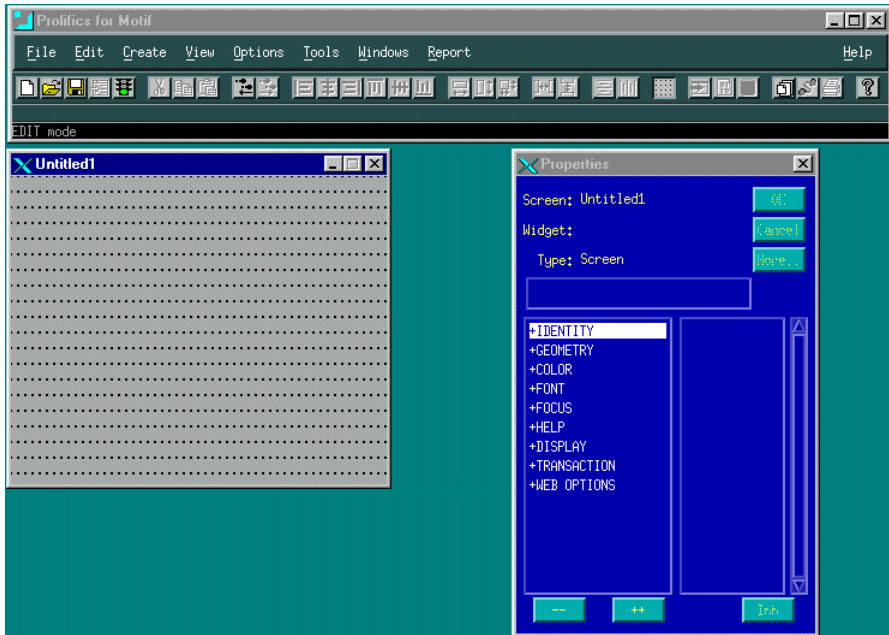
```
SMFLIBS=client.lib:common.lib:server.lib
export SMFLIBS
```

- 3 Apply the settings in `setup.sh`:

```
. setup.sh
```

- 4 At the command line, type `prodev`.

The editor workspace opens.



Connect to the middleware

A Panther client connects to servers through the middleware, which enables client connections to all active servers, including the file access server for access to remote libraries and repositories, and the standard server for database interaction. Since a UNIX local client opens local libraries, you can request a middleware connection from within the editor.

- 5 Choose File→Open→Middleware Session.

The Connect dialog box opens. The configuration file is provided if the entries in your setup file were properly set. If your user name is set in your environment, it will be provided as well.

- 6 Choose OK. You are now connected to the middleware.

More About the Middleware

In a three-tier architecture, communication between clients and servers, across a network is managed by middleware software. Panther facilitates all interaction between these agents, passing information from client to middleware, middleware to server, and back again.

Open a library and access library members

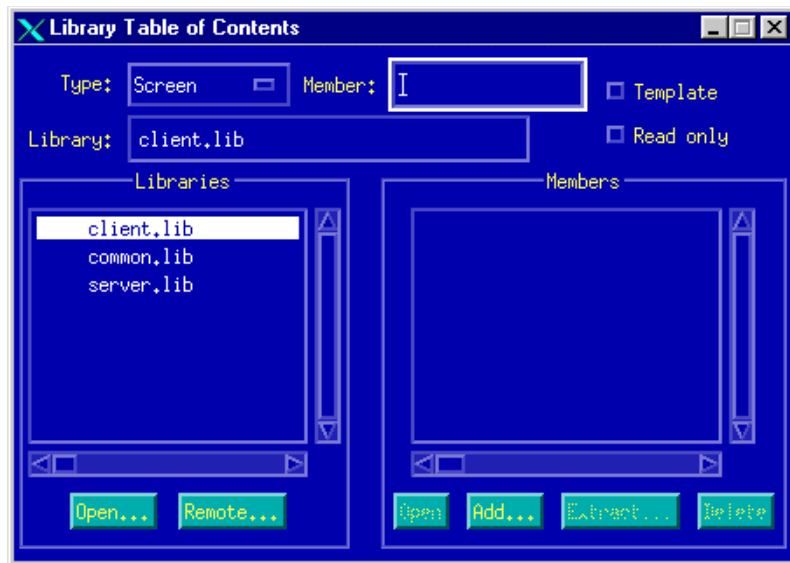
A tutorial library is provided with the Panther installation and was copied to the tutorial's application directory. This library contains an example of a client screen and its corresponding service component. First, open the library to access its members.

More About Client Screens and Service Components

Client screens and service components each serve a specific purpose in Panther applications. They look very similar, but client screens reside on the client and represent the user interface, while service components reside on the server and are invisible to the user at runtime. A service component, which is a graphical representation of a service, provides a physical means of sending, receiving, and processing data between a client screen and a service.

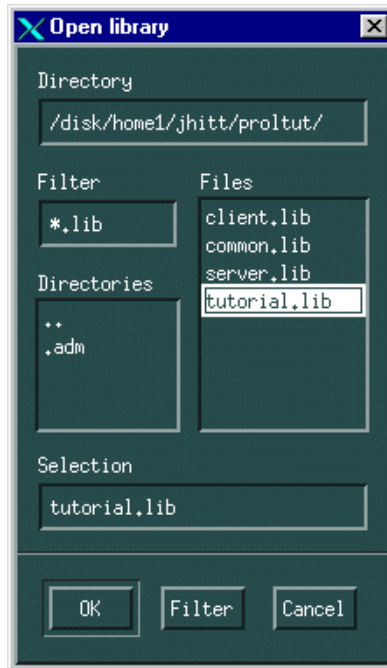
- 7** Choose View→Library TOC.

The Library Table of Contents opens.



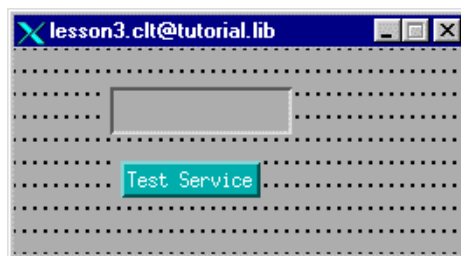
- 8 Under Libraries, choose the Open button to gain access to libraries stored on your system.

The Open Library dialog box opens:



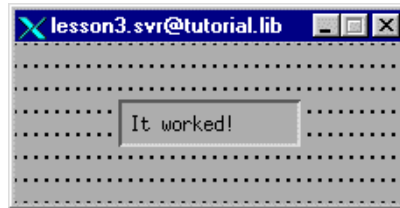
- 9 From the `/prolifics/samples/tutorial` directory, select `tutorial.lib` and choose OK.
- The Library Table of Contents dialog reappears, and `tutorial.lib` is added to the list of open libraries.
- 10 Select `tutorial.lib`, and double-click on `lesson3.clt` in the Members list to open it.

The client screen `lesson3.clt` opens in the editor workspace.



- 11 In the Library Table of Contents dialog, double-click on `lesson3.svr` in the Members list to open it.

The service component `lesson3.svr` opens in the editor workspace.



Save members to appropriate libraries

Now save both the client screen and its corresponding service component to the appropriate remote libraries, under the same name. You must save the service component to the server library to make it available to the server, and to test the connection between client and server later on.

- 12 With focus on `lesson3.svr`, choose File→Save As→Library Member.

- 13 Enter `test` as the Member name, and save it to `server.lib`.

The service component is now available to the server and visible to other developers.

- 14 Close the service component by choosing File→Close→`test`.

- 15 Change focus to `lesson3.clt` and choose File→Save as→Library Member.

- 16 Enter `test` as the Member name, and save it to `client.lib`.

- 17 Close the screen by choosing File→Close→`test`.

If you take a break

If you take a break, you can close the editor by choosing File→Exit and shut down the application in JetMan by choosing Edit→Deactivate (once the application is selected).

To continue the tutorial

To skip to the Lesson 3 summary, refer to [page 3-14](#).

Windows Client

- 1 If your application server is not running, reactivate it now from the server machine ([UNIX-page 1-15](#); [Windows-page 1-28](#)).

Edit initialization files

In order to start a Panther client running on Windows, you must provide setup information in the Panther initialization files, which are included with the client installation. The Panther client must know the name and address of the application server so it can access remote libraries and repositories, and automatically open application libraries on startup.

Two tutorial-specific initialization files are in the Windows directory (typically `C:\WINDOWS`):

- `prtut.ini` for the Panther client process and editor
- `jifedtut.ini` for the JIF editor

In both initialization files, variables need to be set as follows.

- 2 In both `prtut.ini` and `jifedtut.ini` files, verify that `SMBASE` is set to the path name of your Panther installation on Windows. For example:

```
SMBASE=C:\Prolifics\Panther
```

- 3 In both initialization files, set the server's host name and port, as obtained in Lesson 1:

```
SMRBHOST=host  
SMRBPORt=hostAddress
```


The variable `SMFLIBS` identifies the libraries to open when you start Panther components. Because you want only remote libraries to open for the tutorial, all `SMFLIBS` settings should include your server's host name:

- 4 In `prtut.ini`, set `SMFLIBS` as follows:

```
SMFLIBS=host!client.lib;host!common.lib;host!server.lib
```

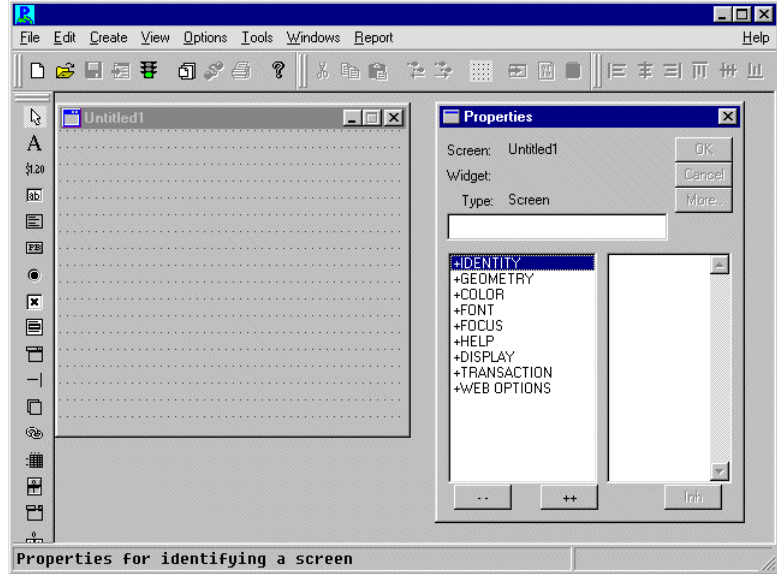
- 5 In `jifedtut.ini`, set `SMFLIBS` as follows:

```
SMFLIBS=host!common.lib
```

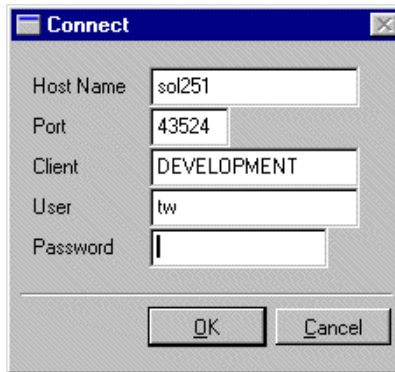
Connect to the middleware

A Panther client connects to servers through the middleware, which enables client connections to all active servers, including the file access server for access to remote libraries and repositories, and the standard server for database interaction. During development, middleware connections can be established in one of two ways, depending on the client setup:

- If a client's `SMFLIBS` environment variable is set to open remote libraries (as in step 4), a dialog automatically displays when you invoke one of the Panther editors. This dialog asks whether to connect to the middleware.
 - If `SMFLIBS` is not set or only opens local libraries, you can request a middleware connection from within the editor.
- 6 From the Start menu, choose Programs→Panther Client→Tutorial (folder)→Tutorial.



- 7 If the Connect dialog does not appear, choose File→Open→Middleware Session. Assuming that the .ini settings in Step 3 are set correctly, the Connect dialog box opens. If remote libraries are specified in the .ini file, the Connect dialog opens automatically.



More About the Middleware

In a three-tier architecture, communication between clients and servers across a network is managed by middleware software. Panther facilitates all interaction

between these agents, passing information from client to middleware, middleware to server, and back again.

- 8 The configuration file and your user name are provided if the entries in your setup file were properly set. Choose OK.

If the host and port settings are correct—that is, they match the entries in the configuration file `broker.bin`—and the server is active, then you are connected to the middleware. The editor workspace opens.

Open a library and access library members

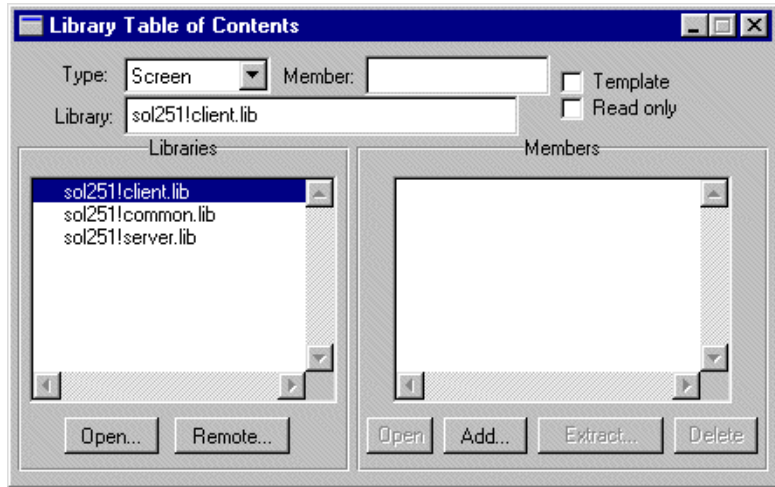
A tutorial library is provided with the Panther installation and is stored locally on your workstation or PC. This library contains an example of a client screen and its corresponding service component. First, open the library to access its members.

More About Client Screens and Service Components

Client screens and service components each serve a specific purpose in Panther applications. They look very similar, but client screens reside on the client and represent the user interface, while service components reside on the server and are invisible to the user at runtime. A service component, which is a graphical representation of a service, provides a physical means of sending, receiving, and processing data between a client screen and a service.

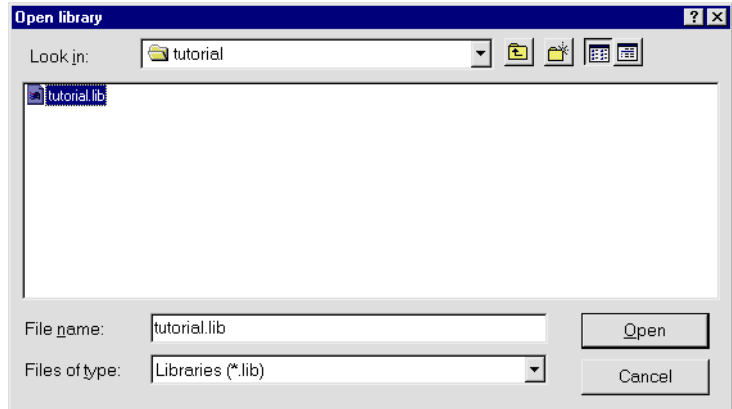
- 9 Choose View→Library TOC.

The Library Table of Contents opens.



- 10 Under Libraries, choose the Open button to gain access to libraries stored on your system.

The Open Library dialog box opens.

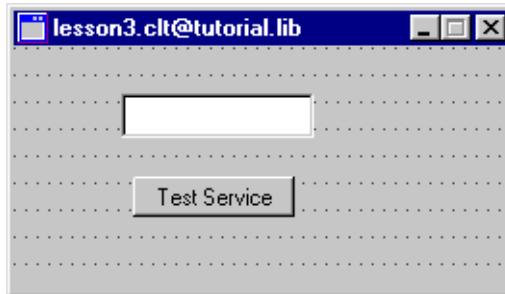


- 11 From the tutorial directory (*pantherInstallDir*\Samples\Tutorial), select `tutorial.lib` and choose Open.

The Library Table of Contents dialog reappears, and `tutorial.lib` is added to the list of open libraries.

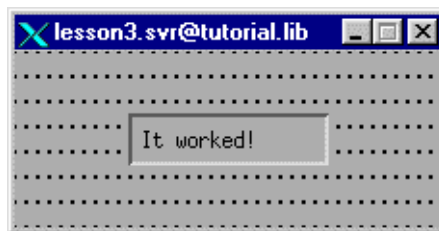
- 12 Select `tutorial.lib`, and double-click on `lesson3.clt` in the Members list to open it.

The client screen `lesson3.clt` opens in the editor workspace.



- 13** In the Library Table of Contents dialog, double-click on `lesson3.svr` in the Members list to open it.

The service component `lesson3.svr` opens in the editor workspace.



Save members to appropriate libraries

Now save both the client screen and its corresponding service component to the appropriate remote libraries, under the same name. You must save the service component to the server library to make it available to the server, and to test the connection between client and server later on.

- 14** With focus on `lesson3.svr`, choose File→Save As→Library Member.
- 15** Enter `test` as the Member name, click on the (remote) `server.lib` to select it, and choose OK.

The service component is now available to the server and visible to other developers.

- 16** Close the service component by choosing File→Close→`test`.

- 17 Change focus to `lesson3.clt` and choose `File→Save As→Library Member`.
- 18 Enter `test` as the Member name, and click on the (remote) `client.lib` to select it, and choose `OK`.
- 19 Close the screen by choosing `File→Close→test`.

If you take a break

If you take a break, you can close the editor by choosing `File→Exit` and shut down the application in JetMan by choosing `Edit→Deactivate` (once the application is selected).

What did you do?

In this lesson, you performed these tasks:

- On Windows, edited the Panther initialization files `prtut.ini` and `jifedtut.ini` (for the JIF editor).
On UNIX, you applied the settings in the `setup.sh` file.
- Connected to the middleware, which gives you access to the application servers and the services that they provide, including access to remote libraries.
- From the editor, opened the tutorial library, and from there opened a client screen and its corresponding service component. You saved these to the appropriate shared libraries.

What did you learn?

You learned:

- When a Panther application starts up it uses an initialization file (Windows) or setup and resource files (Motif) to determine values for a variety of attributes affecting application behavior. These attributes include the name and address of application servers so that Panther can access the services that they offer,

including remote library access and remote processing, which libraries to open on startup, and so on.

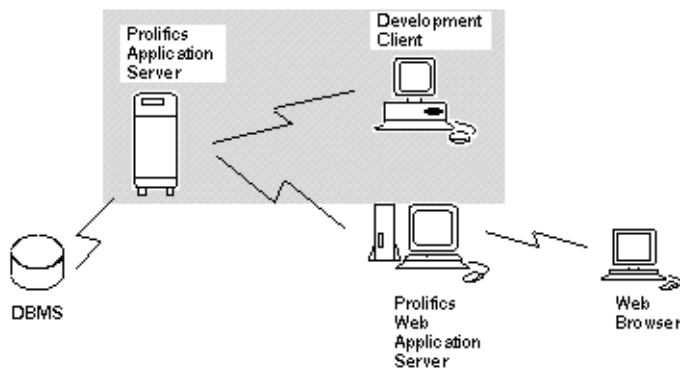
- In three-tier environments, the middleware controls communication between client and server components. The middleware is the interface between client and middleware, and between middleware and server. Depending on your client setup, you can connect to the middleware on startup or from within the editor.
- During development, you will probably save client screens and service components locally. However, to make them available to the server and to other developers, you must save them to the appropriate client or server shared library.

4 Defining a Test Service

The service component that you saved in Lesson 3 contains a service routine that populates the field on the client screen. This and other service routines are invoked on the server at runtime to handle service requests, which are typically issued by a client.

All services must be defined for your application in the JIF, an interface file that maps each service name to a service routine and container. The JIF also defines other attributes of a service, such as its input and output requirements.

There are a variety of other functions related to services that the JIF provides. For more information on the JIF, refer to [page 25-1](#) in the online *Using the Editors*.



In this lesson you learn how to:

- Invoke the JIF editor.
- Define a service that is used to test client/server connections.

-
- Save the JIF file.
 - Test all connections.
-

- 1 If necessary, reactivate the application from the server machine ([UNIX-page 1-15](#); [Windows-page 1-28](#)).

Invoke the JIF editor

Because the JIF resides in a common library (and can even be remote), the JIF editor must be connected to the middleware. As before, your client setup determines whether you are prompted for a middleware connection on startup of the JIF editor, or you establish the connection through the editor's menu.

- 2 To start the JIF Editor:

UNIX: Start the editor (`prodev`), and choose Tools→JIF Editor.

Windows: From the Start Menu, choose Programs→Panther Client→Tutorial→Tutorial JIF Editor.

The JIF editor opens and the View Services dialogue box appears.



Define a service by connecting to the middleware


The View Services dialog box, which shows all defined services in the JIF, is currently empty. Now define a service that tests your client/server connection.

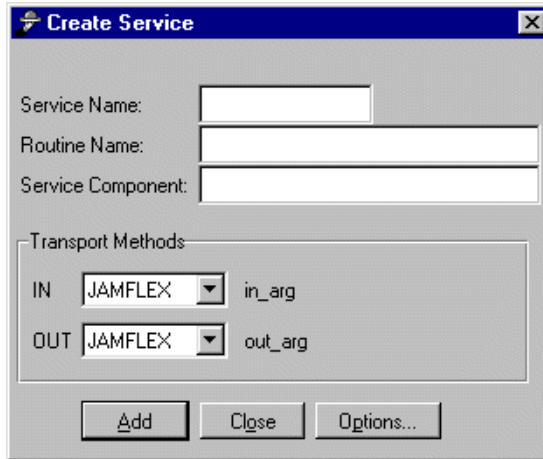
- 3 First, connect to the middleware if you were not prompted to do so when you started. Choose File→Open→Middleware Session. Assuming that the `.ini` files (Windows) or `setup.sh` (UNIX) are set correctly, the Connect dialog box opens.



For Windows, the dialog box displays the host and port information provided in the initialization file. If the Host Name and Port number are blank, type in the settings which you recorded in Lesson 1.

- 4 Choose OK to accept the settings. The View Services screen reappears.
Note: If the connection to the middleware is invalid or is not made, the View Services screen reappears, but you do not have access to the remote `common.lib` on the server. Also, updates to the JIF cannot take effect and new services are not advertised because the server is unaware of the changes.

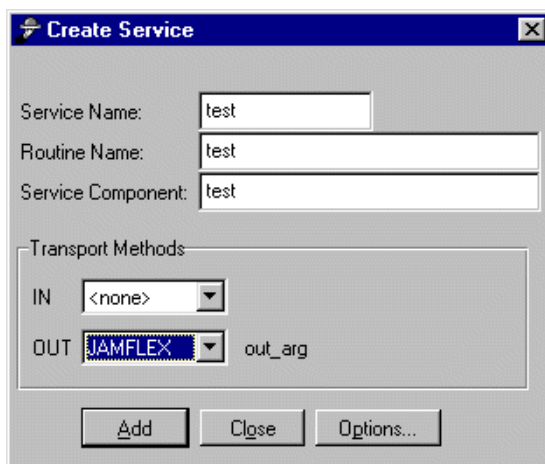
- 5 From the top menu, choose Service→Create or .
The Create Service dialog box opens.



- 6 Enter test as the Service Name, and press TAB.

The JIF editor uses the service name test for the initial setting of the Routine Name and Service Component fields. Panther needs this information to look up the service routine and service component that are associated with this service. Because the names of this service and its service component and service routine are identical, leave these settings unchanged.

- 7 Under Transport Methods, change the IN option to NONE. This means that the service expects no input data. (Leave the OUT option set to JAMFLEX.)

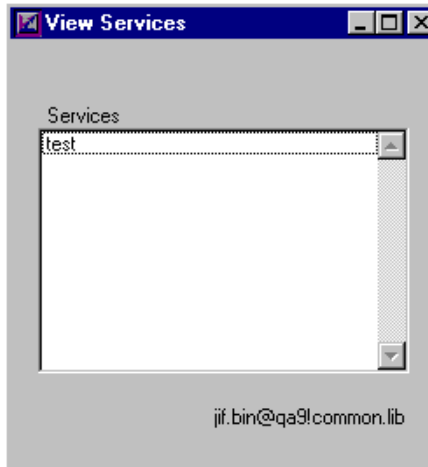


- 8 Choose Add.

The service test is defined in the JIF. The fields clear so you can create another service.

- 9 Choose Close to close the Create Service dialog box.

The View Services dialog displays the service that you created:



- 10 Save the service test to the JIF by choosing File→Save or .

Because you are connected to the middleware, servers are notified immediately when changes are saved to the JIF. Also, new services (depending on the advertise specification and if the server is configured to advertise all services) are advertised and immediately available to your application.

The service test is now available for testing.

- 11 Choose File→Exit.

You are prompted to release the reservation on the JIF because it is stored in a remote library and shared with others.



- 12 Choose Yes to release the reservation.

The JIF editor closes.

Are you connected?

The client screen and service component are now saved to the appropriate libraries, and defined the corresponding service in the JIF. You can now test the client/server connection. This is done by using the push button on the client screen test to call the service routine in the corresponding service component. If the connection is set up correctly, the service returns the appropriate value to the client.

Windows: Start the Tutorial editor.

(UNIX clients should be displaying the editor after leaving the JIF editor.)

- 13 Choose View→Library TOC and open the screen test in `client.lib`.

- 14 Access Test Mode by pressing F2 or File→Test Mode.

The test client screen opens in Test mode.

- 15 Choose Test Service.

This invokes the test service routine (which resides on the test service component). If a connection to the server exists, the service component returns the string "It worked!" to the client, which displays it in the text field.

- 16 Choose Options→Editor.

You return to the editor workspace.

- 17 Choose File→Exit to end your session in the editor.

For the next lesson

For the next lesson, leave the application server running so you can test the screen in a web browser.

What did you do?

In this lesson, you performed these tasks:

- Invoked the JIF editor.
- Used the JIF editor to define a service, test, to the JIF and associated it with a service routine and service component.
- Tested the connection between the client and the server by using the client screen to call the newly defined service.

What did you learn?

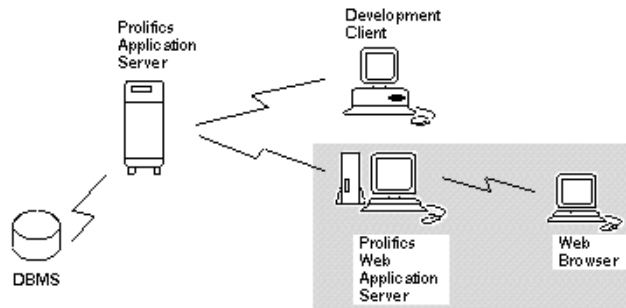
You learned:

- After you determine which services are required by your application, you need to define those services in the JIF. The JIF is the central storage file of information about your application's services. It specifies all the services and service groups that are available for a server to advertise. It also provides the mapping between a service and its service component and routine. The JIF is stored in the distributed common library and is therefore accessible to the development team. You create and edit the JIF with the JIF Editor.



5 Setting Up the Web Application Server

In Lessons 1 through 4 you set up the Panther application server, started the client, saved a service component and client screen to the correct libraries, updated the JIF, and tested the client screen. At the end of Lesson 4, a client made service requests to a server. In this lesson, you will set up a Panther web application and test the same client screen from a web browser.



A Panther web application server is installed on your HTTP server and can work with the following web architectures:

- CGI (Common Gateway Interface)
- ISAPI (Internet Server API for Microsoft's Internet Information Server)
- NSAPI (Netscape Server API)

When a browser request comes in for a Panther screen, the HTTP server passes the screen name to Panther. Panther opens the screen, performs any processing on the screen including service calls, generates the HTML to display the screen, at which point the HTTP server transmits the HTML back to the browser that requested it. For more information on web applications, refer to the *Web Developer's Guide*.

This lesson shows how to set up a Panther web application and is independent of all subsequent lessons in this tutorial.

In this lesson you learn how to:

- Run the Panther Web Setup Manager.
- Hook in a JPL module to handle the Panther web application server startup and shutdown procedures.
- Start up the Panther web application server and test the connection.

More About Web Application Server Processes

The Web Application Server consists of three executable processes for each application: requester, dispatcher, and jserver.

- Requester - accepts the CI/ISAP/NSAPI request from the HTTP server, asks the dispatcher for an available jserver, passes the request to the jserver, waits for the response, and transmits it back to the HTTP server.
- Dispatcher - Acts as the interface between the requester and the jserver. The dispatcher keeps track of available and busy jservers, and provides an available jserver to the requester.
- Jserver - Processes the CGI/ISAPI/NSAPI request from the requester, performing all application processing.

Before starting this lesson

Before beginning this lesson, you need to know the following:

- The location of your Panther web application server installation:

-
- UNIX: The default location is `/usr/prolifics`.
 - Windows: The default location is `C:\Prolifics\Panther`.

-
- The location of your HTTP server's program directory:

Common names for the program directory include `cgi-bin` and `scripts`.

- The URL for the Panther web setup manager on your HTTP server, as in:

`http://hostMachineName/ProgramDirectory/websetup`

This tutorial assumes that the Panther application server and the Panther web application server are installed on the same machine or accessible via the network. To test the screen on the web, your HTTP server must be able to access your application directory.

Start the Web Setup Manager

Each Panther web application must have an initialization file, which contains configuration settings for the application. The Panther Web Setup Manager helps you create a new initialization file (or update an existing one).

- 1 Start up your web browser (for example, Mozilla Firefox or Microsoft Internet Explorer).
- 2 In the URL window, type the location of the Panther Web Setup Manager.

UNIX HTTP server:

`http://hostMachineName/ProgramDirectory/websetup`

Windows HTTP server:

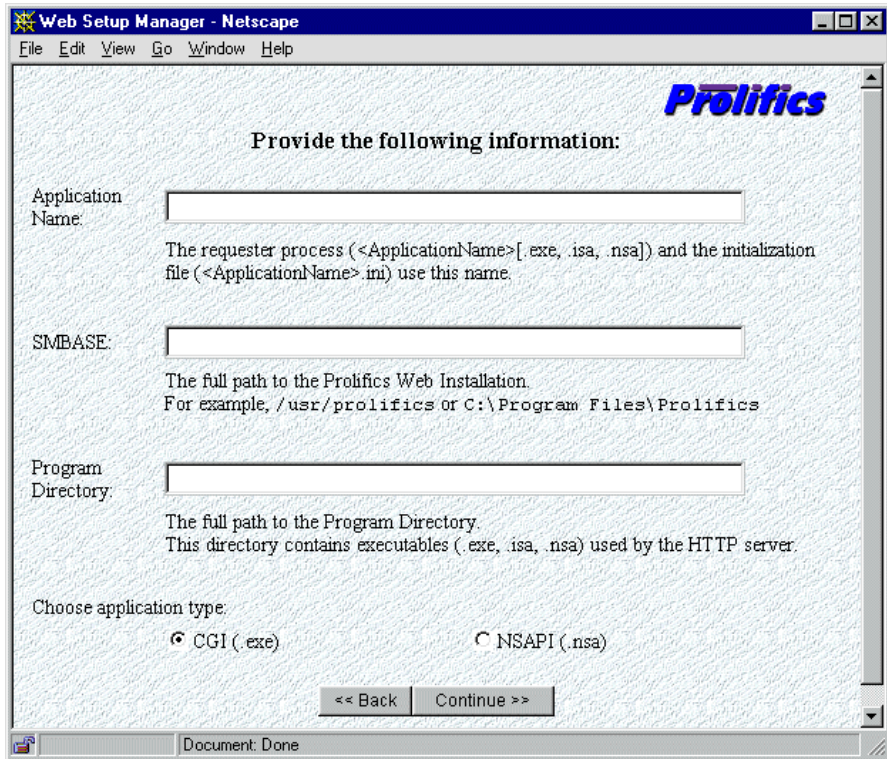
`http://hostMachineName/ProgramDirectory/websetup.exe`



- 3 Check that the Create an application radio button is selected.
- 4 Choose Continue.

Enter the program locations

You assign each web application a unique name which is used to name the application's requester program and the application's initialization (.ini) file.



- 5 For Application Name, enter a unique name to identify this lesson's web application, like `tut_webapp`.
- 6 For SMBASE, type the full path pointing to the Panther web application server installation (which you recorded on [page 5-2](#)).
- 7 For Program Directory, type the path to your HTTP server's program directory (which you recorded on [page 5-2](#)).

Note: This should be a directory path, not a browser URL path.

- 8 For application type, choose CGI.
(There are three types of requester programs. Depending on your platform, your options can include CGI, NSAPI, and ISAPI.)
- 9 Choose Continue.

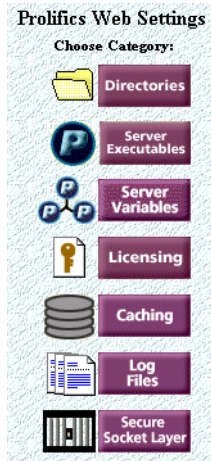
The program will now create both an .ini file and an executable specific to your application (for example, `tut_webapp.ini` and the executable `tut_webapp`). The next screen will tell you that both files have been created; the following steps will be to update the initialization file with the correct settings.



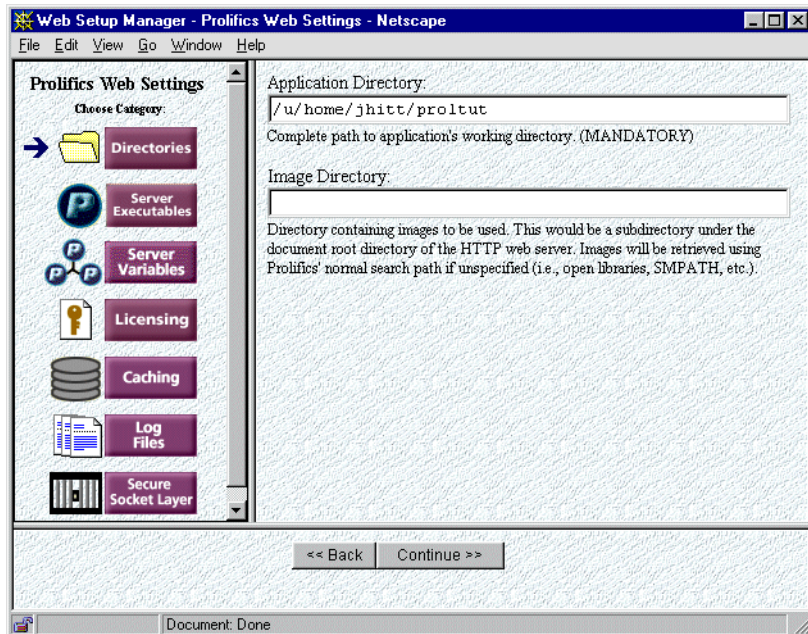
10 Choose OK.

Check the settings for your Web Application Server

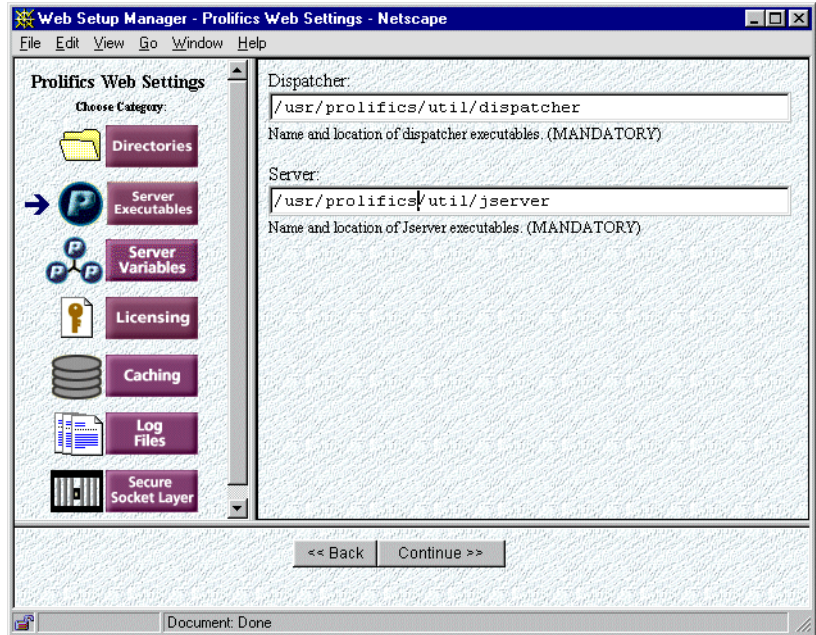
In the left-hand frame, you will see icons for the initialization settings specific to the Panther web application server. When you click on each setting, there will be an accompanying explanation of how to set that variable. Most settings can be left at the default for now; however, the following steps lead you through settings which you need to edit or double-check:



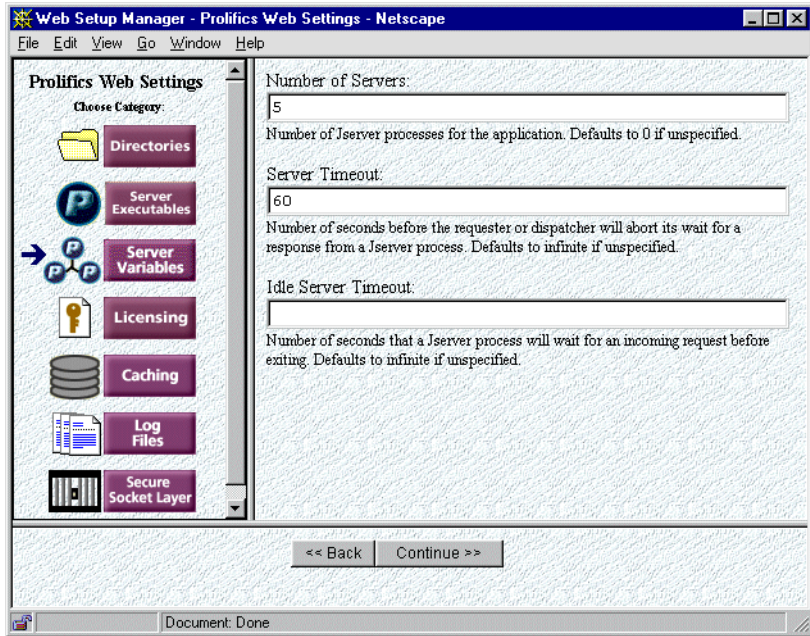
- 11 In Application Directory, enter the path where you created your `proltut` directory in Lesson 1.



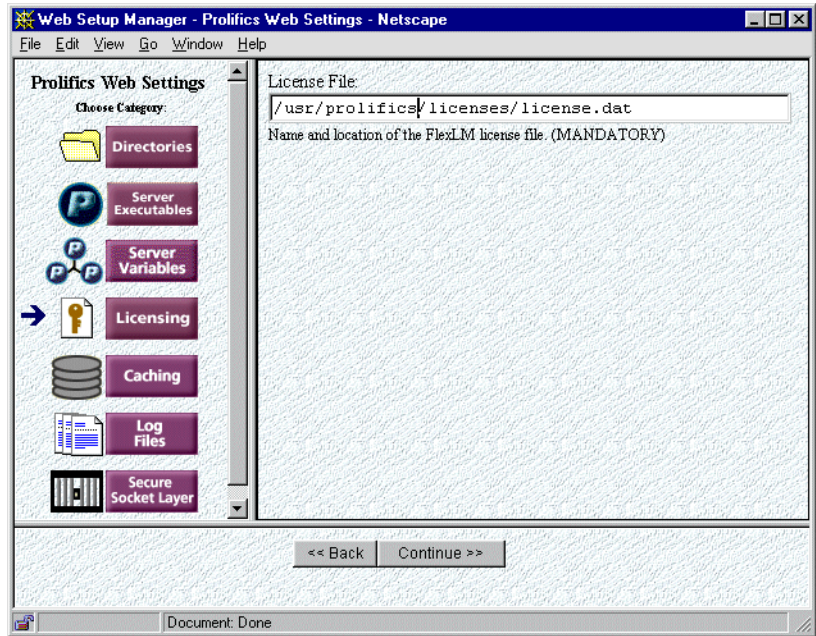
- 12 Choose Server Executables in order to specify the location of the dispatcher and jserver executables. Generally, the path location is to the `util` directory of the web application server installation.



- 13 In Dispatcher, set or check the full path name of the dispatcher executable.
- 14 In Server, set or check the full path name of the jserver executable.
- 15 Choose Server Variables.



- 16 In Number of Servers, set it to 5. This is the number of concurrent users. Since this setting greatly affects performance, you must set it for each web application.
- 17 Choose Licensing.



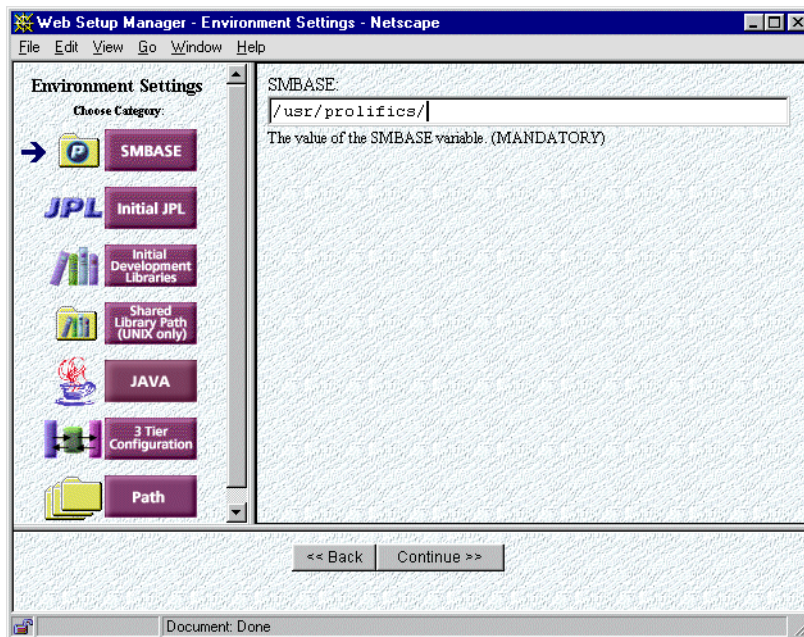
- 18 In License File, set or check the path of your license.dat file.
- 19 Choose Continue.

General environment settings

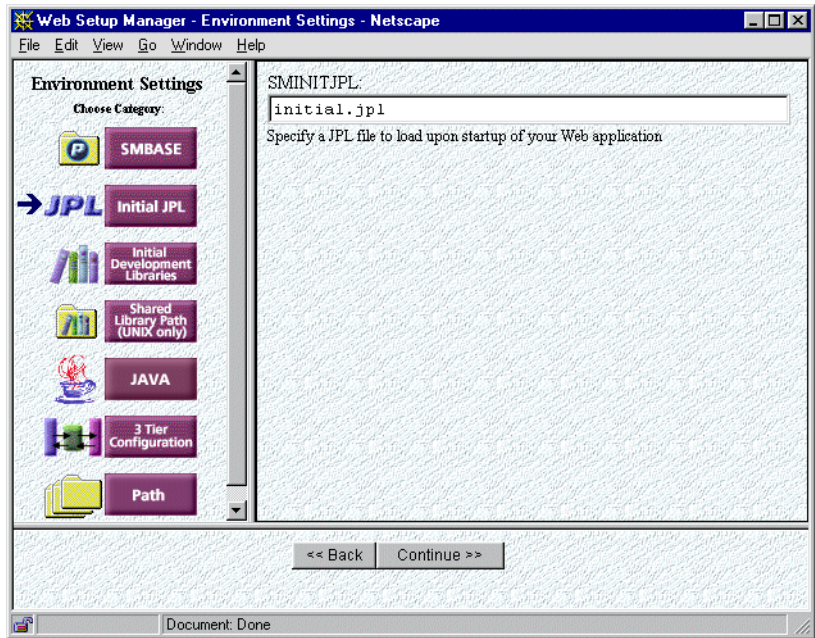
In the next section, you specify the environment settings needed by the web application server. You will see a new set of icons in the left-hand frame. Step through each category, editing or double-checking settings as necessary.



20 In SMBASE, set or check the setting of the web application server installation.



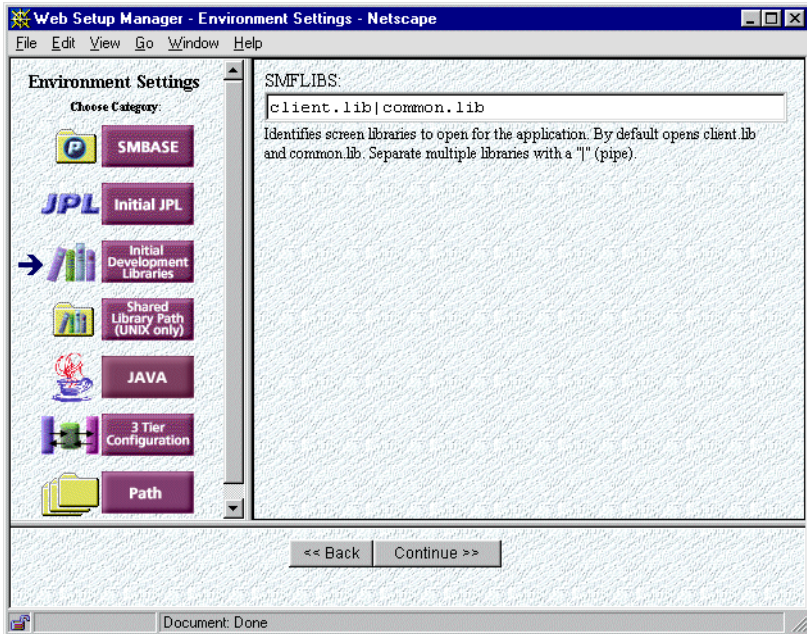
21 Choose Initial JPL.



22 In SMINITJPL, enter `initial.jpl`.

This is the name of the JPL file that will be run when the web application server starts. (Later in this lesson you will add this JPL file to the client library.)

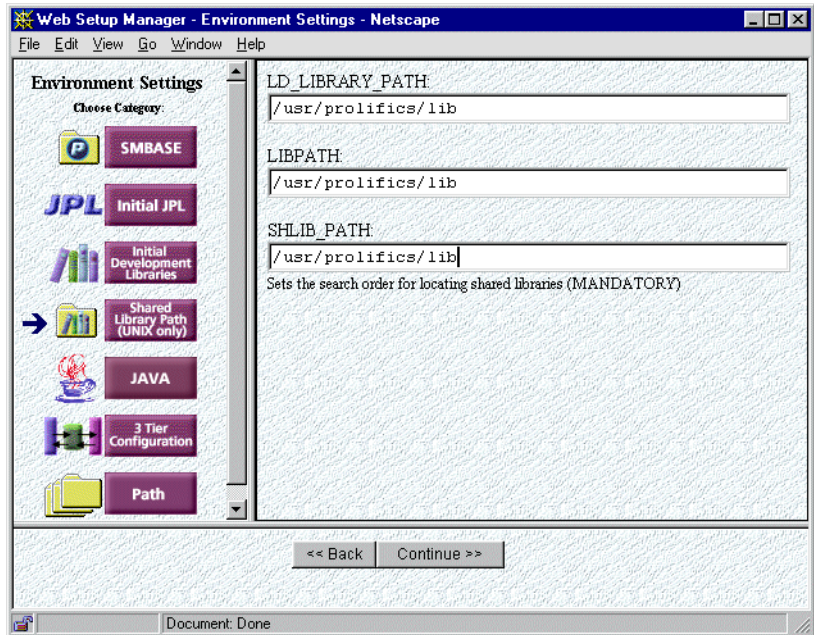
23 Choose Initial Development Libraries.



24 In SMFLIBS, set or check the settings for client libraries:

`client.lib|common.lib.`

25 (*UNIX only*) Choose Shared Library Path.



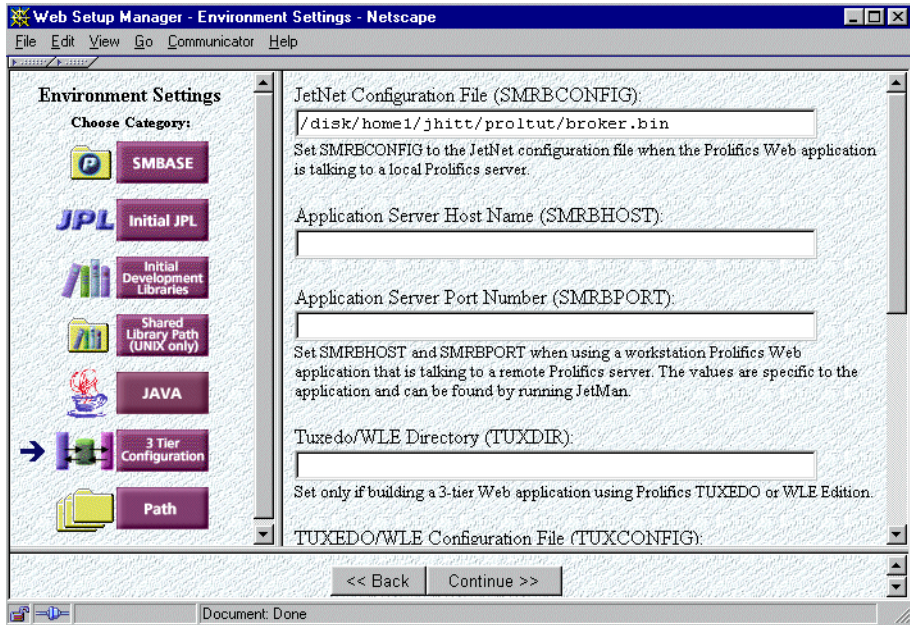
- 26 (UNIX only) Depending on the operating system, enter one of the following:
- Set `LD_LIBRARY_PATH` to the full path name of `WebInstallDir/lib`.
 - (AIX) Set `LIBPATH` to the full path name of `WebInstallDir/lib`.
 - (HPUX) Set `SHLIB_PATH` to the full path name of `WebInstallDir/lib`.

3-Tier Configuration

Depending on whether you are running the Panther web application server on a separate machine from your Panther application server, you will need to set this parameter in one of two ways, as outlined below.

- 27 Choose 3 Tier Configuration from the left-hand menu.
- (If running a Panther application server and a Panther web application server on same machine): In JetNet Configuration File (`SMRBCONFIG`), type the directory path name where you created your `broker.bin` file in Lesson 2. Leave the lines for `SMRBHOST` and `SMRBPOR` blank.

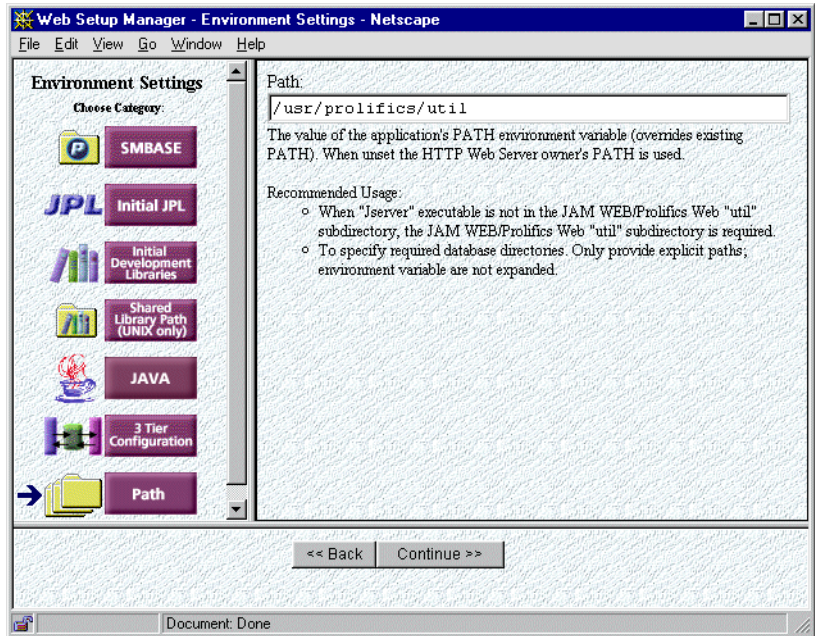
- (If running servers on different machines): Leave the line for JetNet Configuration File (SMRBCONFIG) blank. Enter the information for SMRBHOST (refer to Step 18 on page 1-9 in Lesson 1 for UNIX and to Step 11 on page 1-21 for Windows) and for SMRBPORT (refer to Step 22 on page 1-10 in Lesson 1 for UNIX and to Step 17 on page 1-23 for Windows).



28 If you are running the Panther web application server on a different machine from the Panther application server, you will need to set a value for certain platforms. It is not needed if you are using JetNet or Oracle Tuxedo 6.4 or later. Choose one of the values below and enter it under Work Station Device:

- For Oracle Solaris platforms, enter:
/dev/tcp.
- For SCO platform, enter:
/dev/inet/tcp.

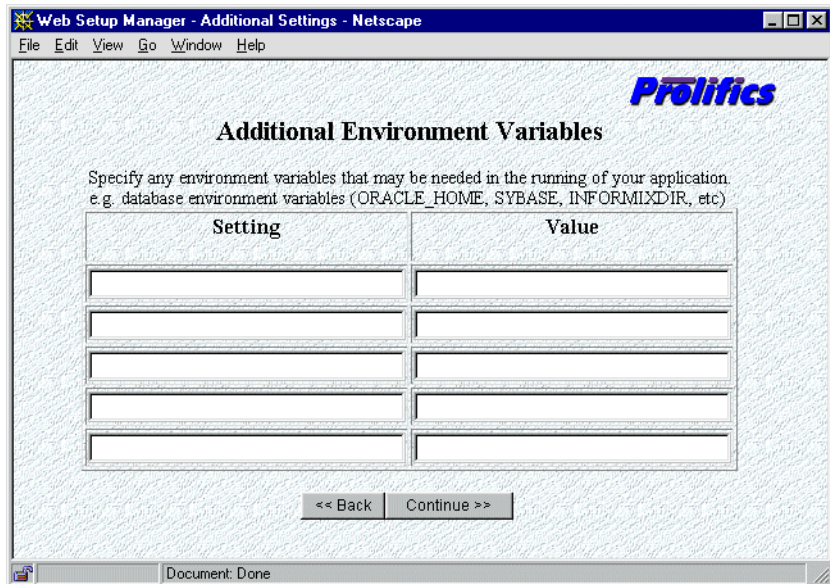
29 Choose Path.



- 30 Check that the path location is set to the `util` directory of the web application server installation. If the `jserver` executable is not in the Panther `webutil` directory, you must set the correct path to the `jserver` directory here.
- 31 Choose Continue.

Specify database settings and workstation jserver

The next screen allows you to enter Additional Environment Variables for such applications as databases. If you were setting variables for an Oracle database, for instance, type the variable under “Settings” (e.g., `ORACLE_HOME`) and the path name under “Value” (e.g., `/usr/oracle`). You will also be setting access to the JIF file.



- 32** Under Setting, enter `SMTPJIF`, the Panther variable that identifies the JIF file.
Note: If the `EnableWebId` option displays, note that it will not be used in the tutorial.
- 33** Under Value, enter `jif.bin`.
- 34** Choose Continue. A screen informs you that your Panther web application has been created.



35 Choose Start Application.

Add JPL routines to the client library

When a Panther web application server starts up, it looks for a JPL `web_startup` procedure to handle any necessary startup processing, open and close connections to the middleware API (and therefore to the database), and create global variables. Similarly, when the server is shut down, it looks for the `web_shutdown` routine—accessed by all users of the web application—to perform processing such as clean up. Both procedures are in the JPL module `initial.jpl` which is in the tutorial library. By copying `initial.jpl` to your client library and setting `SMINITJPL` in `proweb.ini`, these routines automatically run when the web application server is initialized.

More About JPL

JPL is a powerful scripting language with a C-like syntax. You can write and edit JPL modules directly in the editor environment, using the JPL editor or your preferred test editor. You can also write JPL procedures directly to a library and call these procedures from objects in your screens.

You'll practice writing and executing JPL in Lesson 13 of this tutorial. For more information on JPL commands and syntax, refer to the online [Programming Guide](#).

36 For a UNIX client:

- Log on to your Panther application server and change to the application directory:

```
cd username/proltut
```

- Reapply the settings:

```
. setup.sh
```

- Invoke the editor by starting up the client executable, `prodev`. At the command line, type:

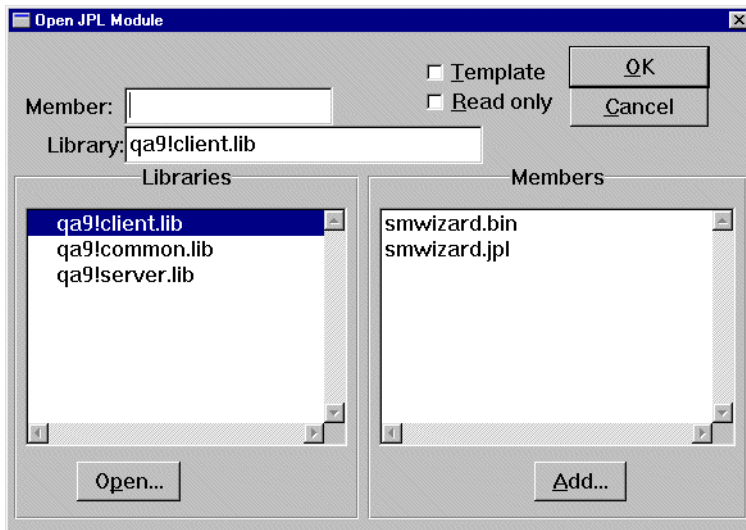
```
prodev
```

37 For Windows client:

- From the Start menu, choose Programs→Panther Client→Tutorial (folder)→Tutorial.
- When the Connect screen opens, check your Host and Port settings for the middleware session and choose OK.

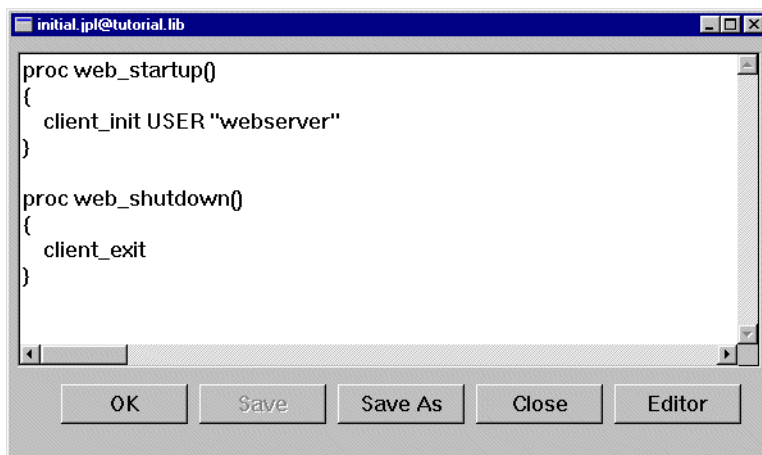
38 Choose File→Open→JPL.

The Open JPL Module dialog opens.

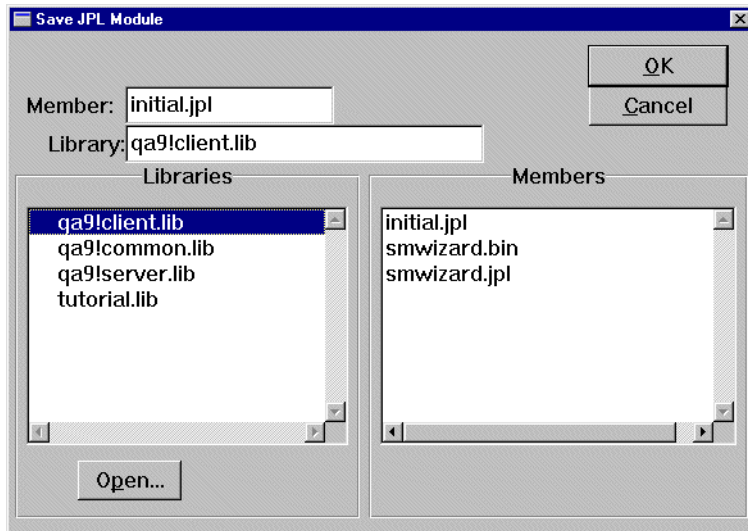


- If `tutorial.lib` is not open, choose Open and select it from the Open Library dialog box. (You will need to set the directory path to `/prolifics/samples/tutorial/` to find this file).
- 39** Select `tutorial.lib` from the list of open libraries, and double-click on `initial.jpl` in the Members list to open it.

The `initial.jpl` file opens in the JPL Program Text dialog box.



-
- 40 Choose Save As. The Save JPL dialog opens with `initial.jpl` in the Member box.



- 41 Select `client.lib` in the Libraries list. Choose OK.
The `initial.jpl` file is saved to `client.lib`.
- 42 Choose OK.
- 43 Choose Close.
- 44 Choose File→Exit to end your editor session.

Change permissions of shared files (UNIX only)

The Panther web application server must be able to access some files in the application directory on UNIX. Because the user account for the HTTP server is different from your user account, the web application server is normally not allowed to write to your directory.

- 45 Switch to the `/proltut` directory.

If `error.log` does not exist, create an error log file and set its permissions so that the web application server has read and write access to the `file:touch error.log`

```
chmod 666 error.log
```

- 46** Change permissions so that the Panther web application server has read and write access to the server configuration file, `broker.bin`:

```
chmod 666 broker.bin
```

Test the connection

Now you can start the Panther web application server and test the client/server connection by trying to invoke the service routine from your client screen with a web browser.

- 47** If your Panther application server is not running, reactivate it now from the server machine (refer to Steps 27 to 31 on [page 1-11](#) in Lesson 1 for UNIX or refer to Steps 22 to 26 on [page 1-25](#) in Lesson 1 for Windows). Also, start the web browser, if it is not already running.

- 48** Since the Panther web application server was started when you ran the web setup manager, you must restart it so that it will read the new JPL file:

```
WebInstallDir/util/monitor -stop webAppname  
WebInstallDir/util/monitor -start webAppname
```

Note: On Windows, if your web application has been installed as a service, then you use the following command to start the web application:

```
WebInstallDir/util/net start webAppname
```

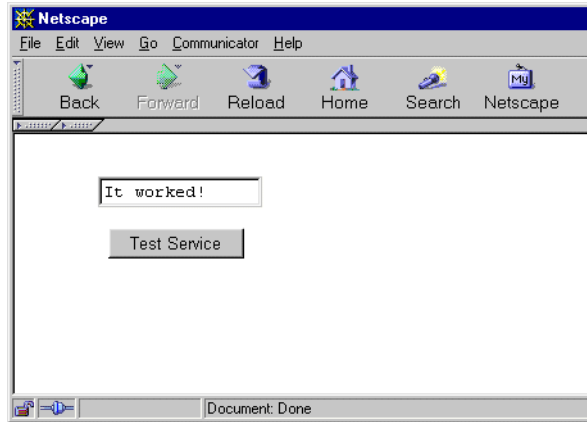
- 49** Request the test screen with this URL:

```
http://hostName/ProgramDirectory/webAppname/test
```

The client screen appears.

- 50** Click on the Test Service button.

The phrase “It worked!” is displayed in the text box.



If you have a problem either starting the web application server or seeing the message returned from the server, refer to Appendix B on [page B-1](#).

Stop and restart server after making any changes

If you make any changes to the web initialization (`webAppname.ini`) file, or if you see any error messages indicating that you failed to connect to the server, you must stop and restart your web application server to have the changes take effect. Shut down the server by going to a UNIX window and typing at a command prompt:

```
$SMBASE/util/monitor -stop webAppname
```

Edit the initialization file the same way you created it, by typing in the browser:

```
http://hostMachineName/ProgramDirectory/websetup
```

and choosing to update the application.

You can then restart the server by repeating the procedure outlined in Step 47 above.

Shut down the server

After you successfully start the web application server, you can shut it down. The lessons that immediately follow do not require its usage.

51 Shut down the web application server:

```
$SMBASE/util/monitor -stop webAppname
```

What did you do?

In this lesson, you performed these tasks:

- Used the Panther Web Setup Manager to create a copy of the CGI/ISAPI/NSAPI program so that your application is available on your HTTP server.
- Used the Panther Web Setup Manager to create a web application server initialization file, `webAppname.ini`, which has the same name as your web application. and edited it to reflect variable settings for your application.
- Used the Panther Web Setup Manager to specify the application's settings for your applications.
- Saved a copy of a JPL module, `initial.jpl`, to your client library. This file contains startup and shutdown routines that run when the web application server is initialized.
- Changed permissions on the error log file and configuration file `broker.bin`, so the web application server has read/write access to them.
- Started up the web application server, tested the connection, and shut the server down.

What did you learn?

You learned:

- A Panther web application is only accessible on an HTTP server if there is a link to, or a copy of, its requester executable in the server's program directory.
- Each application must have an initialization file, which contains variables determining the behavior of the web application, the Panther environment used by the `jserver` program, and the database environment.
- Certain processing, such as opening a middleware session or setting global variables, occurs each time a web application server starts. By creating JPL

routines to handle this processing, and then setting variables in the web application initialization file, you can set up default processing to take place each time the web application server is initialized.



Module 2— Creating and Testing Screens



6 Creating a Repository

When you begin to build a database application, you need to create a repository. A repository that resides on the server lets you as well as other members of the development team take advantage of its contents. You can directly, or via the screen wizard, use the database-derived objects in the repository to create client screens and service components.

In this lesson, you learn how to create a remote repository using the editor and populate it with imported database objects. The import process creates a screen, or repository entry, in the open repository for each database table that you import. The repository entry contains widgets representing each column defined in the database table.

In this lesson you learn how to:

- Access the editor and open a middleware session.
- Create a remote repository.
- Connect to a local copy of the vidsales database.
- Import the vidsales database tables into the remote repository.

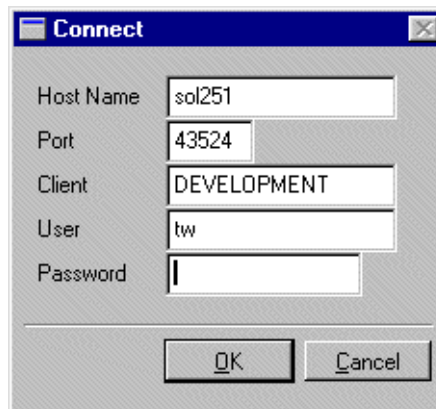
-
- 1 If necessary, reactivate the application from the server machine ([UNIX-page 1-15](#); [Windows-page 1-28](#)).

Connect to the middleware

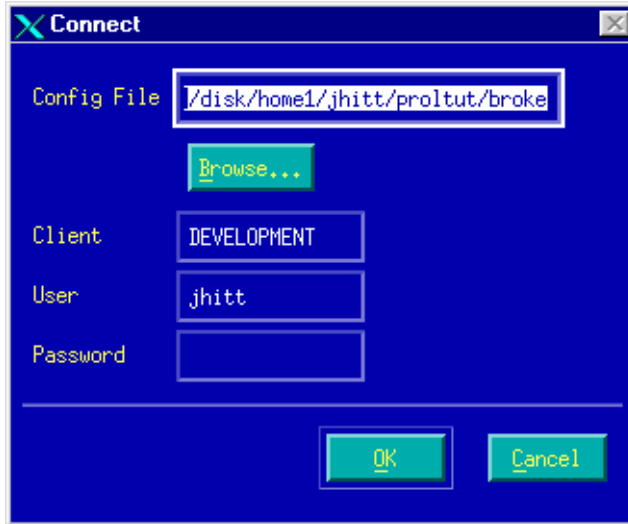
When you start one of Panther's editors, such as the screen editor, menu bar editor, or JIF editor, and you want to access libraries through the middleware, you need to open a middleware connection. After you connect to the middleware, you can access libraries and repositories located on the server machine, and the data base specified in the server configuration.

- 2 If necessary, start the editor.
- 3 Depending on whether you invoke the editor from Windows (as a remote client) or UNIX (as a local client), use one of these procedures:

Windows – The Connect dialog opens and displays the information you provided in your Windows initialization file. Choose OK to accept the settings.



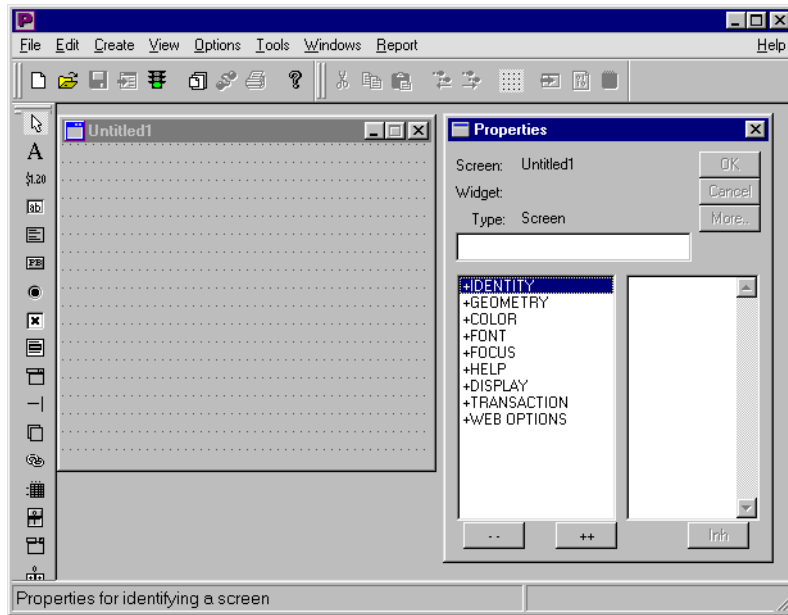
UNIX – Choose File→Open→Middleware Session. The Connect dialog displays the location of the configuration file (broker.bin) as you specified in your environment setup.



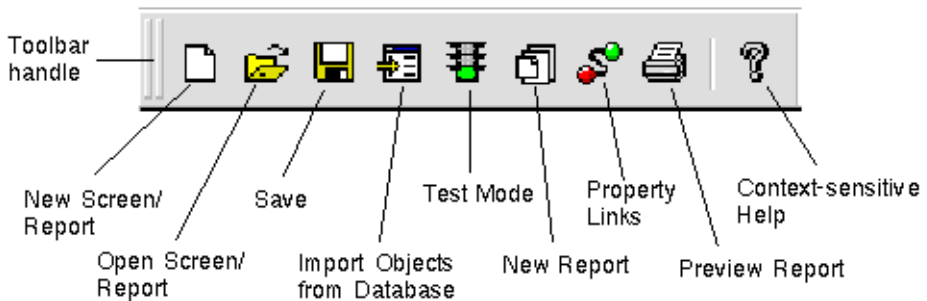
- 4 Choose OK.

If the information is correct—that is, it matches the entries in your server setup—you are connected to the middleware. Otherwise, you have no access to the remote libraries on the server which are required for the tutorial.

The editor workspace opens.



Many commonly used commands and tasks can be executed by choosing the appropriate toolbar item. This tutorial acquaints you with most of the File menu commands that are associated with these toolbar buttons:



More About Toolbars

The editor's toolbar:

- Simplifies use of the editor and facilitates screen design.

- Includes tool tips so you can quickly learn what the buttons do.

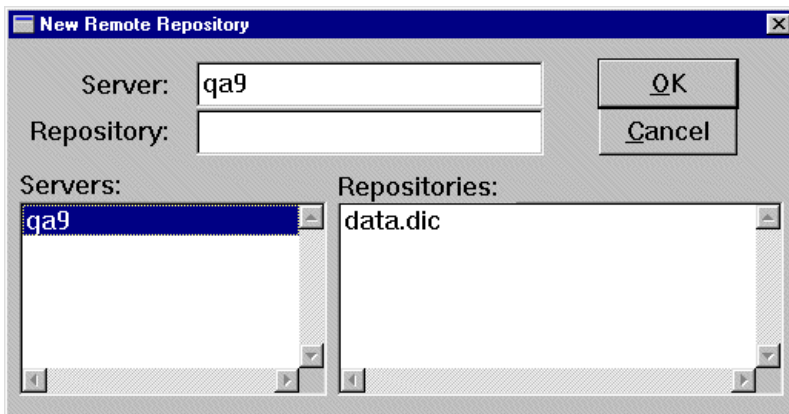
You can also use the Panther menu bar editor to create menus and toolbars to attach to individual screens or to the application as a whole.

Create a repository

Create a remote repository where you can store imported database objects.

- 5 Choose File→New→Remote Repository.

The New Remote Repository dialog opens.



- 6 Enter `data.dic` as the name of a repository in the Repository field.

On startup, the editor automatically opens any local repository with the name `data.dic`.

On Windows, to automatically open the remote repository, set the `SMDICNAME` variable to `host!data.dic` in your `prtut.ini` initialization file.

- 7 Choose OK.

Creating a new repository automatically closes any repository that is open.

More About Repositories

You can create local as well as remote repositories when you are connected to the middleware and running a file access server. Remote repositories can be accessed by the development team and thereby provide a consistent

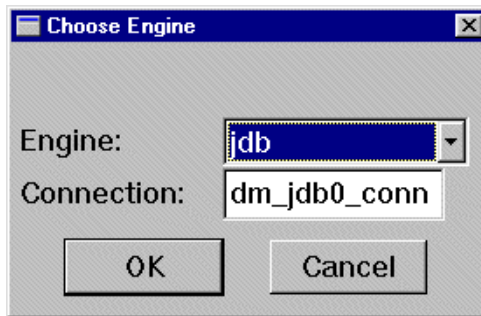
appearance and behavior to your application. You can control the look and feel of application objects by modifying objects in a repository. From this centralized location, you can use and reuse the objects throughout the development of your application. Changes that you make to repository objects automatically propagate to the various screens that contain copies of the objects. This greatly simplifies both application development and maintenance.

Connect to the database

A direct connection to the database (that is, not connecting through the middleware) allows you to import database definitions. A direct connection also lets you test a simple client/server model (two-tier processing) or the service component behavior in a three-tier, distributed application model.

- 8 Choose File→Open→Database.

The Choose Engine dialog opens.

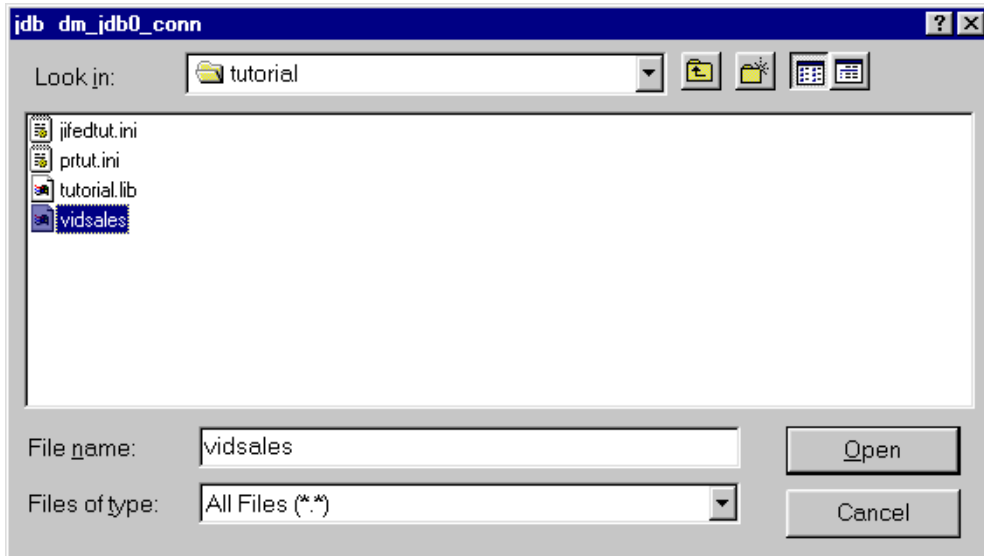


- 9 Choose OK to accept the defaults (in this case, jdb is the engine and the connection is dm_jdb0_conn). A database-specific dialog appears.

More About JDB

JDB is Panther's single-user SQL database. You can use it to quickly prototype your application and refine multi-user database applications without the need for an external third-party database.

- 10 Select or specify vidsales in the File Name field, then choose Open.



You are now connected directly to the local copy of the database. You also have a connection, via the middleware, to the remote database on your server which was made available when the server was activated.

Note: If the vidsales database is not in your current directory, locate it with the dialog.

Import database tables

When you import database objects into the repository, you can take advantage of the data definitions already in the database. The import process creates one screen in the repository for each selected database object. The contents of the resulting repository screens can be used to build application screens that access data. The next few steps show how to import database information into Panther.

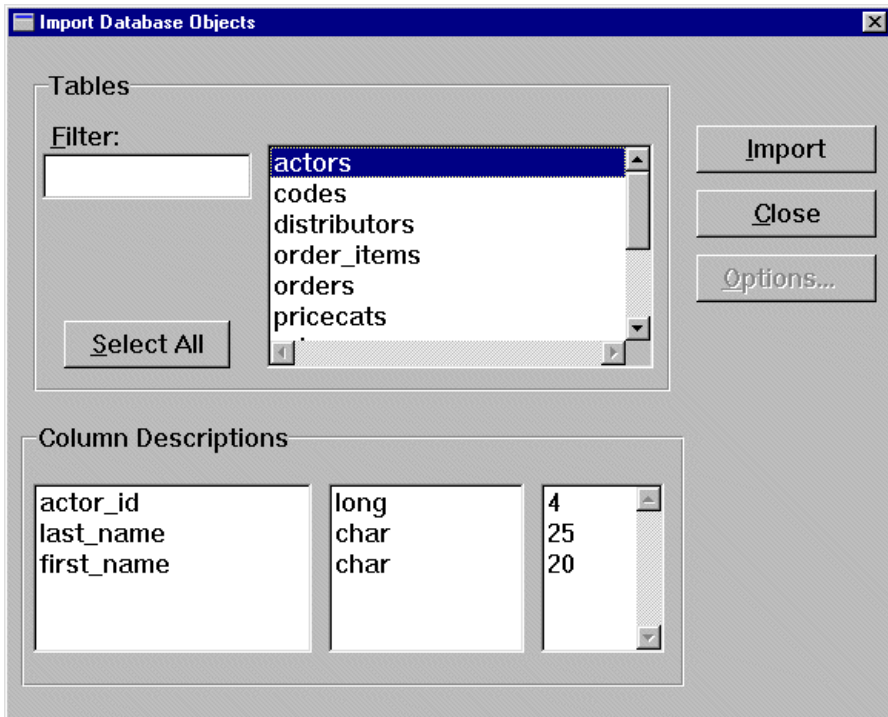
More About Importing Database Objects

You can import database table definitions into the repository at the outset of your development process, and reimport whenever the database schema changes. Also, if your database engine supports other database object types, such as views and synonyms, you can import those as well.

All screen objects that inherit from repository objects are maintained without your having to access the database or manually edit client screens and service components. Development can continue even when the database is inaccessible or unavailable.

- 11 Choose Tools→Import Database Objects on .

The Import Database Objects dialog opens. All tables in the vidsales database are listed.



When you select a table, the detail information for the table is displayed in Column Descriptions: column name, data type, and length. You can select multiple tables simultaneously: click+drag or Shift+click to select contiguous objects; Ctrl+click to select noncontiguous objects.

- 12 Choose Select All to select all of the tables in the database.
- 13 Choose Import.

The status line informs you of each table imported into the repository and notifies you when the import process is completed.

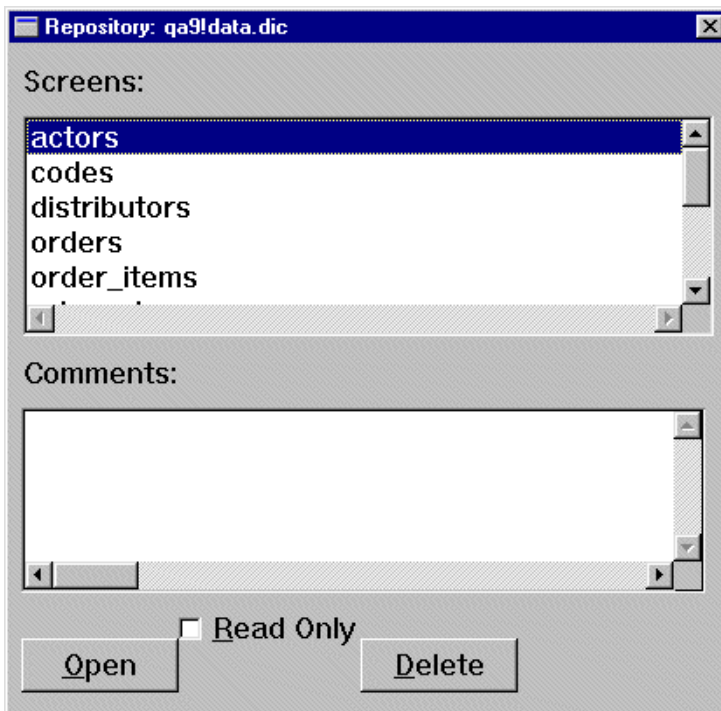
- 14 When the import is complete, choose Close to return to the editor.

View repository contents

You can use the repository table of contents (TOC) to select and open repository entries successively for copying and editing purposes. The TOC can remain open during your editing session.

- 15 Choose View→Repository TOC.

This menu option toggles the display of the Repository table of contents. The TOC opens and displays a list of all entries in the open repository.



-
- 16** (Optional) Leave the Repository TOC open so you can easily access repository entries. To close the TOC, choose View→Repository TOC, or choose Close/Quit from the TOCs system menu.
-

What did you do?

In this lesson, you invoked the editor and:

- Connected to the application server by way of the middleware.
- Connected directly to a local copy of the database so that you could import the database definitions from the vidsales database.
- Created a remote repository for developing your database application.
- Imported the database tables associated with the local copy of the JDB database to the remote repository.
- Viewed the contents of the remote repository using the Repository TOC.

What did you learn?

You learned:

- The repository is a development tool for storing application and database objects that the entire development team can use and reuse throughout application development. A shared repository can minimize duplication of effort and enforce application consistency.
- You must have a direct connection to the database to import database definitions to a repository.
- The import process creates an entry in the open repository for each database table that you import.
- You can test service components with a direct database connection, using them like a client screen to see if the correct data returns from the database.

-
- JDB is a prototyping database that can be used during application development when an external database is unavailable or is in the design/development phase.
 - On application server initialization, a database connection can automatically be opened. Thus, clients can access a database via their middleware connection to that server.



7 Using the Screen Wizard

The screen wizard provides an easy way to create transactional database application components. Use the screen wizard to build:

- Client screens for a traditional client/server application for two-tier processing.
- Client and server components for an application that uses three-tier architecture.
- Screens that can provide the interface to your Internet or intranet application.

In this lesson, you create a master-detail client screen and its corresponding service component. The client screen will allow a user to view, edit and add information for video distributors and their video orders. The master section of the screen displays the address information associated with a single distributor, while the detail section displays information about that distributor's orders.

The screen wizard uses information you provide to also build the service component that will carry out the requests—view, update, and add.

The screen wizard creates both screens and service components by using objects and database columns from the open repository (the remote repository, `data.dic` for the tutorial). After the components are created, you will save them to their appropriate libraries, thereby ensuring that service requests made by the client can be carried out by the server.

In this lesson you learn how to:

- Create a functional client screen and its supporting service component using the Panther screen wizard.

-
- Save the wizard output to the appropriate libraries.
-

- 1 If necessary, reinvoke the editor.

Open the repository

To use the screen wizard, a repository that contains imported database objects must be open. The repository that you created in Lesson 6 opens automatically when you invoke the editor if the `SMDICNAME` variable is set in the development environment (in Windows, `prtut.ini`) or if the repository is located in the application directory, and you can skip to step 3; otherwise, you must open the repository manually (step 2).

Notes: (UNIX clients only) Because the tutorial is in the application directory, Panther opens the local repository by default; you do not have to set the variable.

- 2 (Windows only) If `SMDICNAME` is not set in the initialization file `prtut.ini`, choose File→Open→Remote Repository and select `data.dic` from the Open Repository dialog.

Create screens with the screen wizard

Instead of creating a screen from scratch, use the screen wizard to build your screen.

More About the Screen Wizard

The screen wizard guides you step-by-step in creating database screens that automatically incorporate tables and columns imported to the repository from your database.

The screen wizard prompts you for basic design information and uses that information to build fully functional screens which you can use as-is, or as a basis for further development. The screen wizard eliminates many of the mechanical steps of screen design, thereby increasing productivity.

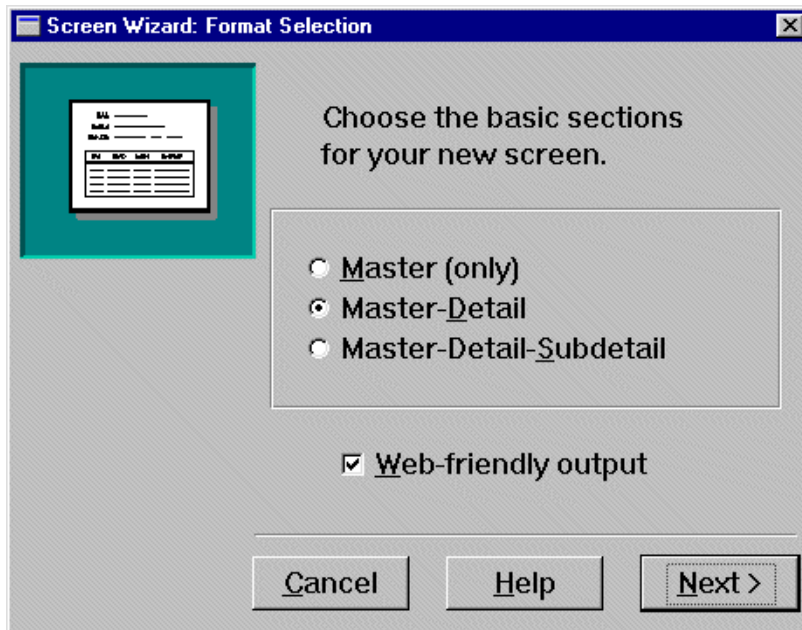
- 3 Choose File→New→Screen or .

The New Screen Tool dialog opens.



- 4 Choose Yes to use the screen wizard.

The Format Selection dialog opens.



More About Screen Wizard Formats

When you use the screen wizard you can choose to create a Master only, Master-Detail, or Master-Detail-Subdetail screen. If you choose the Web-Friendly Output check box, the screen wizard creates a client screen that uses the appropriate layout and push button controls for a web application.

- 5 Select the Master-Detail (default) option to define the sections.
- 6 Deselect the Web-Friendly Output option.

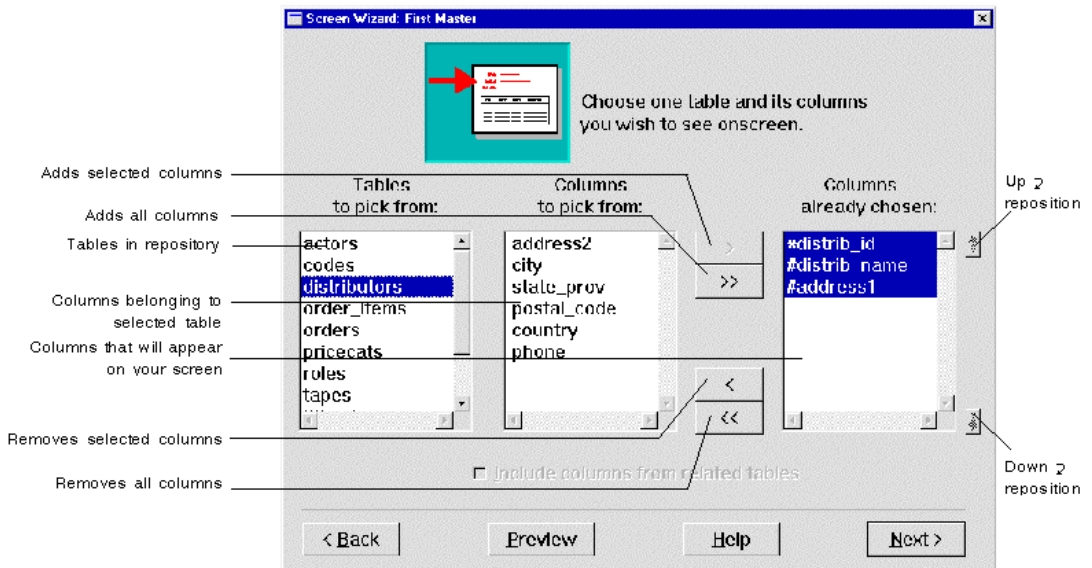
For the purpose of this module, the screen is not being designed for a Web application.

More About Web-friendly Output

The Web-friendly Output option enforces certain web-specific restrictions on the widgets and layout used in the final screen to make it usable in the Internet/Browser interface.

- 7 Choose Next.

The First Master dialog opens.



The list that displays contains repository entries that represent the database tables you imported from the `vidsales` database.

Specify the contents of the master section

Specify the primary table for the screen's master section by selecting from the list of tables in the open repository.

- 8 Select distributors from the list of Tables To Pick From.

The columns belonging to the distributors table are displayed. The primary key (indicated with an asterisk), `distrib_id`, is automatically included as one of the columns that will be on the completed screens. The columns indicated with a number sign (#) must have values when you insert rows into the database.

More About Primary Keys

A primary key is the column, or combination of columns, that uniquely identifies each row in a database table. If the first table you select in a section lacks a primary key definition, the Primary Key Selection dialog opens where you can identify a primary key for the table. These custom-defined primary keys are indicated with a plus (+) sign in the list of tables that ultimately appear on your wizard-generated screen.

- 9 Choose .

All columns are added to the list of columns to display on the finished screen.

Note: You can click+drag or Shift+click to select contiguous items or use Ctrl+click to select non-contiguous items.

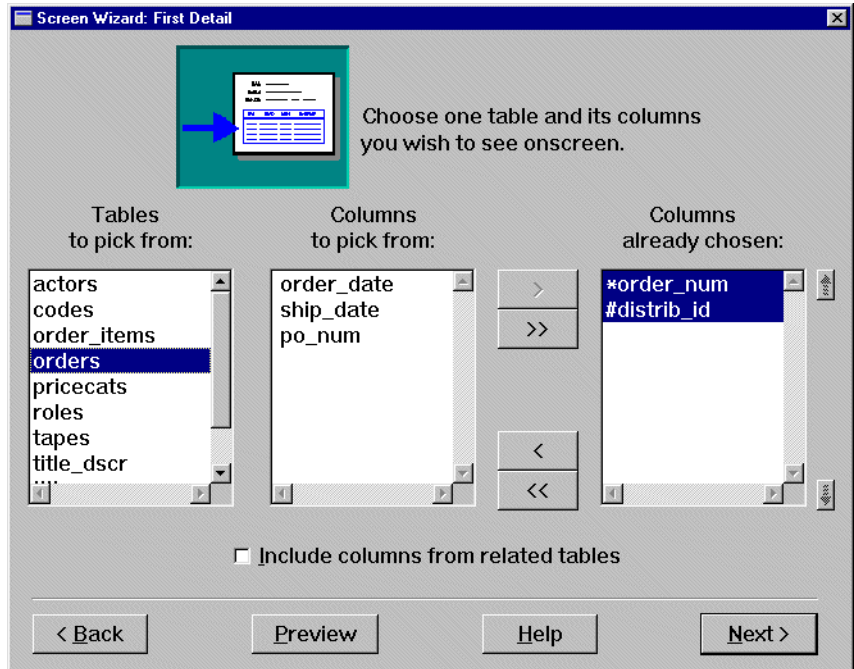
- 10 Choose Next.

The First Detail dialog opens.

Define the detail columns

Choose from the list of remaining tables to specify the columns that you want in the screen's detail section. This screen should display all orders for a given distributor.

- 11 Select orders from the list of tables.



12 Choose .

All columns in the orders table are added to the list of columns to include on the screen.

The finished screen will join the orders table to the distributors table via the `distrib_id` column (the foreign key). This allows the screen to display all orders associated with a given distributor.

More About Links

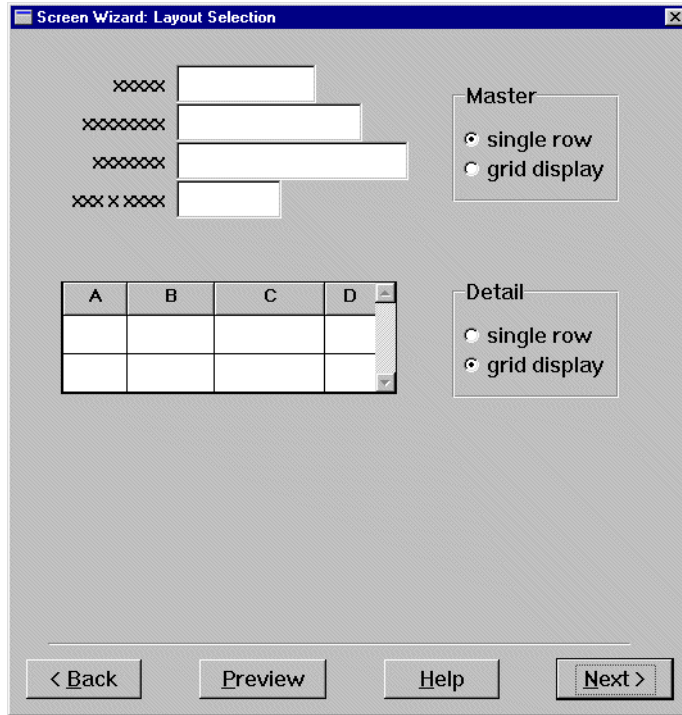
Links are widgets that are automatically created when you import database tables. They are visible in the editor and hidden at runtime. The link is used by the transaction manager to describe relationships between parent and child tables. In the editor, the link appears like a label widget, with the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

The screen wizard expects a link widget to join the first master and first detail tables. The link specifies how the two tables are connected through a list that matches columns in the first master to columns in the first detail. If you select a

first detail table that lacks a link to the first master table on your screen, the screen wizard lets you define the link on the Master Detail Link dialog.

13 Choose Next.

The Layout Selection dialog opens. The master table has a single-row layout and the detail section specifies a multi-row grid display. Use the default layout.



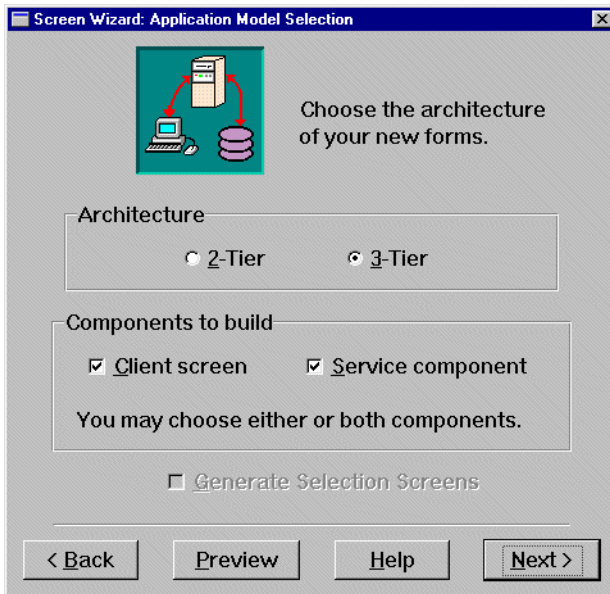
More About Screen Wizard Layout Options

You can choose the type of layout that serves the screen's immediate requirements. For each section, choose one of these layout options:

- Single row—Each database column is represented as a widget (adopting the widget type as defined in the repository) and has a corresponding label.
- Grid display—Each database column is represented as a vertical column in a grid widget and has a column title.

14 Choose Next.

The Application Model Selection dialog box opens.



Since you created a simple master-detail screen with no additional tables included, the option to generate selection screens is not available (you'll do this in Lesson 15).

Specify application architecture

Select the architecture, two- or three-tier, and specify the screens that you want the screen wizard to generate: client screen, service component, selection screens (if applicable) and corresponding selection service components.

More About Two- and Three-tier Architecture

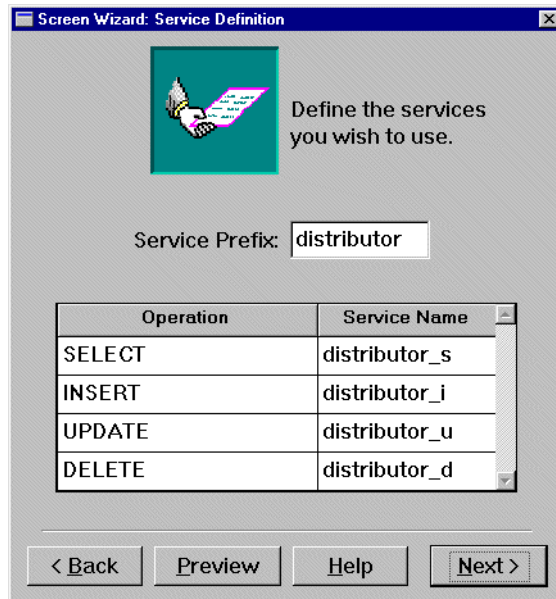
One way your choice of architecture affects your application is in the components that will be included. If you choose two-tier, the Components to Build section of the dialog box becomes inactive. This is because two-tier processing only involves building the client screens, or user interface. Three-tier processing, on the other hand, includes building client screens and their corresponding service components.

Service components are graphical representations of services, and provide a physical means of sending, receiving and processing data between a client

screen and a service. A service component looks very much like its corresponding client screen, but is invisible to the user at runtime.

- 15 Choose Next to accept the default architecture (three-tier) and components (both client screen and service component).

The Service Definition dialog box opens.

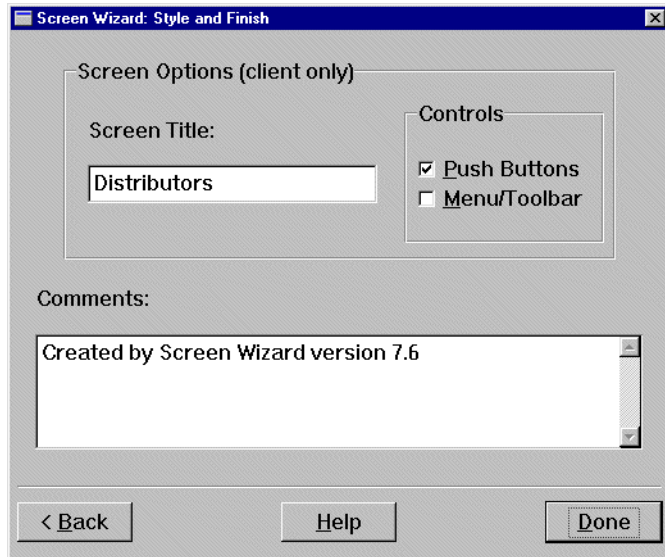


Determine service operations

By default, the screen wizard provides service names that your client screen can use to make requests to select, insert, update, and delete database records. It assigns service routine names by using the name (possibly a truncation) of the master table followed by a mnemonic corresponding to an operation. Therefore, the four services for this particular screen, which uses distributors as the master table, are `distributor_s` for a select operation, `distributor_i` for an insert operation, `distributor_u` for an update operation, and `distributor_d` for a delete operation.

- 16 Choose Next to accept the default service routine names.

The Style and Finish dialog opens.



Customize the output screen

The Style and Finish dialog lets you can customize the screen.

- 17 Change the screen title to Distributor Orders.

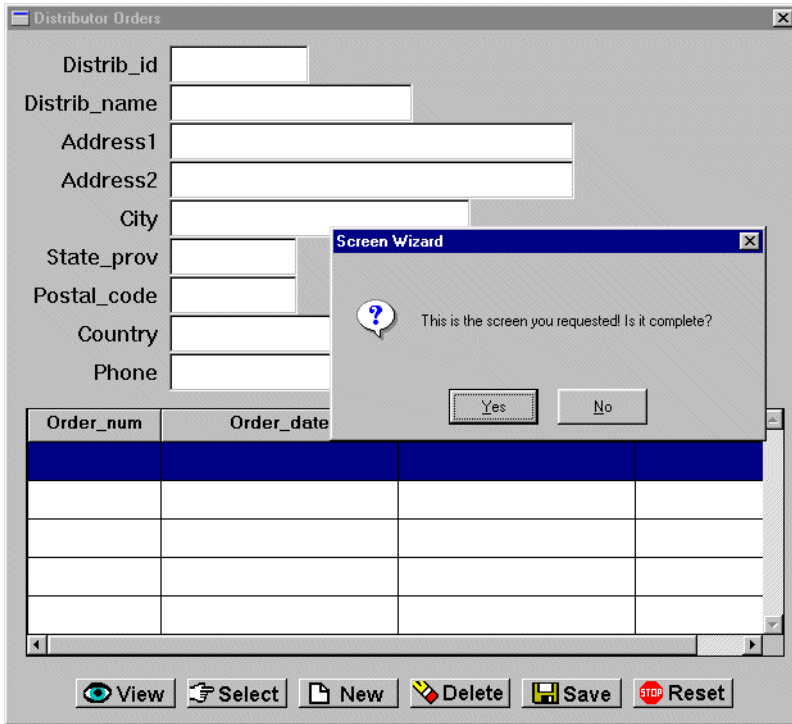
This title displays in the client screen's title bar at runtime (the screen's filename displays in the title bar while in the editor).

- 18 Set Onscreen Controls to Push Buttons (default).

- 19 (Optional) Enter comments about the screen in the Comments field. Although this information has no runtime effect, it can serve as a valuable record of this screen's history—for example, its function within the larger application, idiosyncratic behavior, and so on.

- 20 Choose Done.

Panther displays the finished screen with this prompt:



- 21 Examine the screen and choose Yes to confirm that the screen includes your specifications.

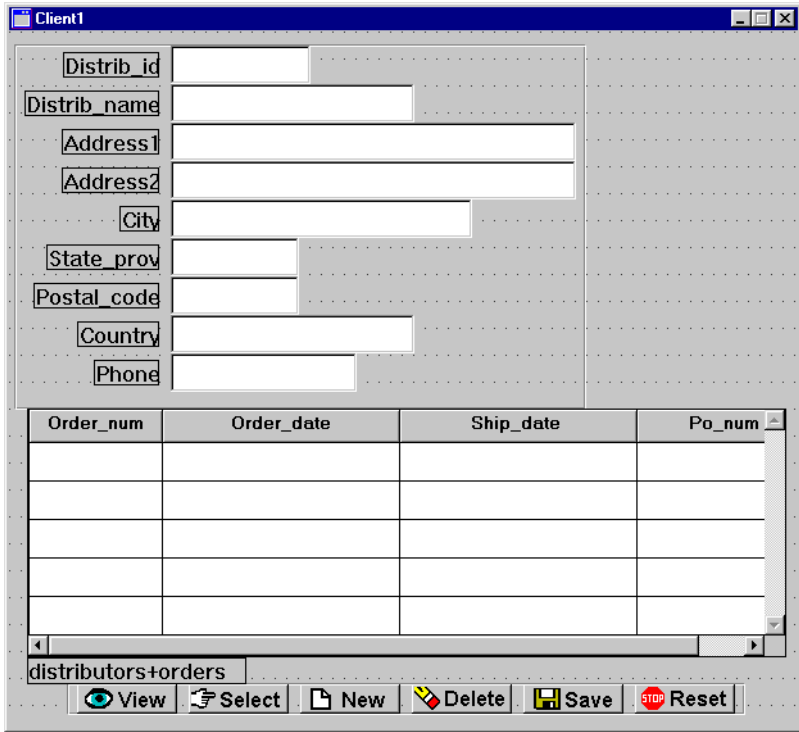
If you choose No, you return to the screen wizard's Style and Finish dialog. You can make further changes there and in previous wizard dialogs.


When you confirm that the screen is complete, the screen wizard builds the screen, then returns control to the editor. The screen appears in the editor workspace.

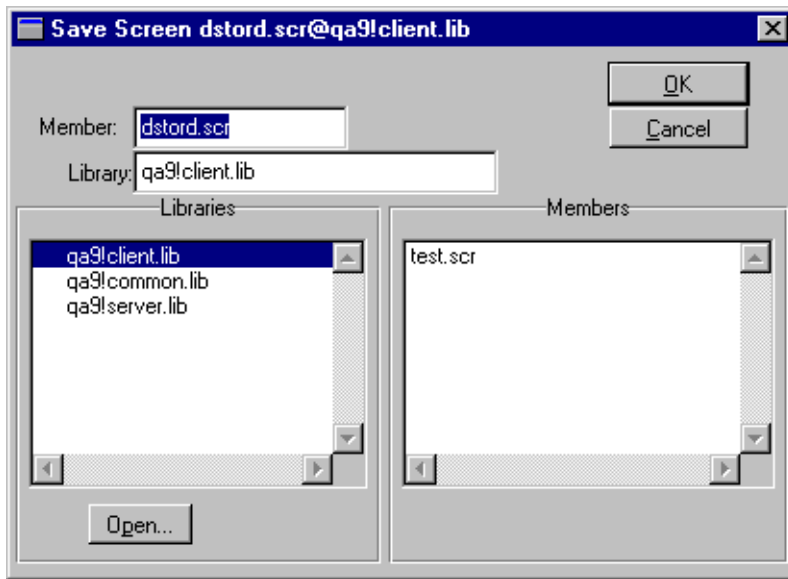
Save the screens

This is a good time to save the screens. You should save the screen in the library `client.lib`.

- 22 Bring focus to the screen.



23 Choose File→Save or .



- 24 Enter `dstord.scr` in the Member field.
- 25 Select `client.lib` as the library in which to store the screen and choose OK.

The filename is displayed in the screen's title bar. This is the screen that users of your application will see. It includes all the decorations and controls that define the user interface.

Note: If the file already exists, Panther prompts you to overwrite the existing one. Choose Yes to overwrite.

More About Naming Conventions

However you name screens, JPL modules, and other application components, you should consistently adhere to a naming convention. If you use extensions, apply them to all application components. For example, use `*.scr` for screens and service components, and `*.jpl` for library JPL modules.

- 26 Move (or minimize) the `dstord.scr` client screen and bring focus to the service component (Server1).

The screenshot shows a window titled 'Server1' with a dotted background. It contains a data entry form with the following fields:

- Distrib_id
- Distrib_name
- Address1
- Address2
- City
- State_prov
- Postal_code
- Country
- Phone

Below the form is a table with the following columns:

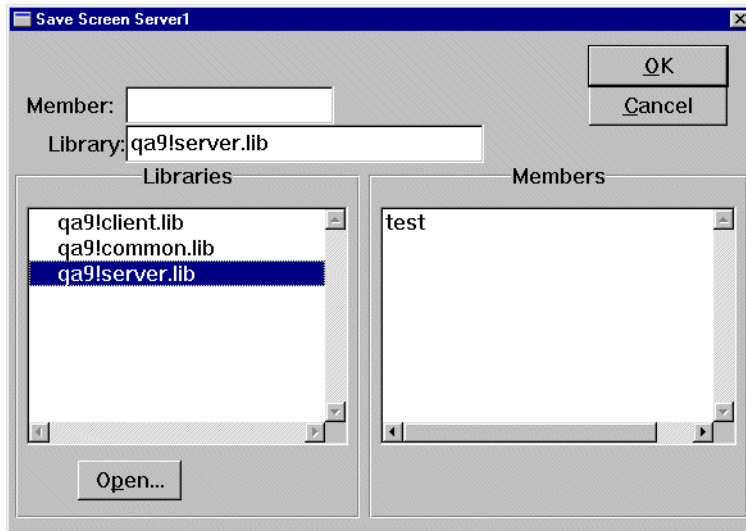
Order_num	Order_date	Ship_date	Po_num

At the bottom of the window, there is a label 'distributors+orders'.

The service component will reside on the server, and therefore, will never be visible to the user. It looks similar to the client screen; however, there are no push buttons, and screen decorations are omitted from service components. However, data entry widgets on the client screen must be identical to those on the service component. That's because the service component must act as a receptacle for data that are ultimately returned to the client and made visible to the user.

27 Choose Save or  .

The Save Screen dialog box opens.



- 28 Enter `dstord.scr` in the Member field as the name of the service component.
- 29 Select `host!server.lib` as the library in which to store the service component and choose OK.
- 30 If you'd like to take a break and exit the editor, choose File→Exit.

What did you do?

In this lesson, you created a master-detail client screen that performs database transactions. To do this, you performed these tasks:

- Used the screen wizard to join two database tables that reside in a repository
- Told the screen wizard that you are implementing a Web-compatible screen.
- Saved the screen in the appropriate remote library.

What did you learn?

You learned:

- The screen wizard uses database-derived objects in an open repository to build fully functional screens.
- The screen wizard uses information from the repository to include the appropriate code in the screens that it creates.

8 Defining Services

The service component that you saved in Lesson 7 contains service routines that display data on the client screen and allow the client to update the database. In Lesson 4, you defined a service, test, in the JIF, and identified the service routine and service component.

All services must be defined for your application in the JIF, an interface file that maps each service name to a service routine and component. The JIF also defines other attributes of a service, such as its input and output requirements.

In general, the JIF is accessed for these reasons:

- A server is activated and checks the JIF to determine which services to advertise.
- A client issues a service call. The JIF provides service parameter attributes—types, number, and direction, and verifies the validity of the service call.
- A server receives a service request and its data, either directly from a client screen or forwarded from another service. The server consults the JIF to find the service routine and its service component (if any).
- The JIF is modified; all active servers are notified of the change and check the JIF in order to readvertise their services.

If service components were created with the screen wizard, you must update the JIF with definitions of the services that are associated with that service component. The JIF has its own editor which lets you access those service definitions and edit them, or add new definitions for services that you create yourself.

In this lesson you learn how to:

- Invoke the JIF editor.

-
- Define services for a wizard-based screen that fetch, add, delete and update distributor's orders.
-

- 1 If necessary, reactivate the application from the server machine (refer to [page 1-15](#) for UNIX and refer to [page 1-28](#) for Windows).

Invoke the JIF Editor

You can add and update services while you are using the editor by invoking the JIF editor.

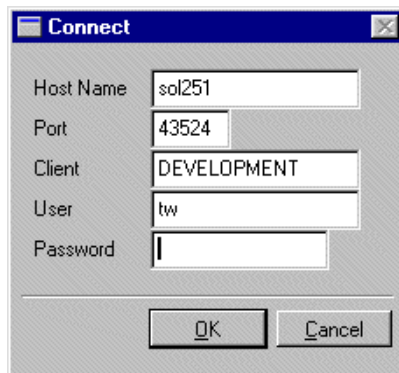
- 2 Invoke the JIF editor by doing one of the following, depending on your platform:

UNIX: Start the editor, and choose Tools→JIF Editor.

- Choose File→Open→Middleware Session.

Windows: From the Start Menu, choose Programs→Panther Client→Tutorial→Tutorial JIF Editor.

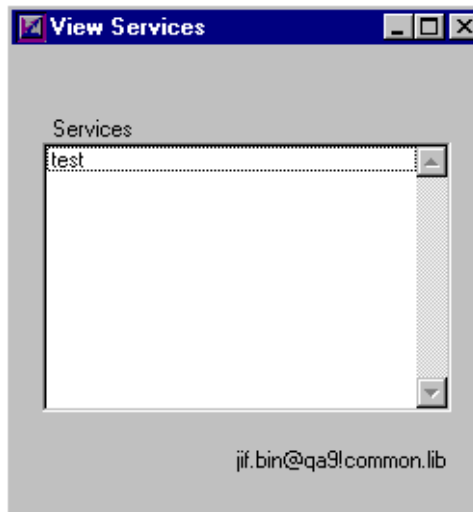
- The Connect dialog box opens and displays the information you provided in your Windows initialization file (`jifedtut.ini`).



- 3 Enter any needed information, and choose OK.

If the information is correct, that is, if it matches the entries you made in your server setup, you are connected to the middleware and the JIF editor workspace opens. The View Services dialog box is displayed showing the test service you defined (Lesson 3) in the `jif.bin` from the remote `common.lib`.

Note: (Windows only) If you did not set the `SMFLIBS` variable in the `jifedtut.ini` to open the remote `common.lib` (`host!common.lib`), choose `File→Open→Remote Library` and select the remote `common.lib` that resides in the `pro1tut` directory. Choose `File→Open→JIF` and select the `jif.bin` library member.




If the connection to the middleware is invalid or is not made, the JIF editor opens, but you will not have access to `common.lib` on the application server. In addition, updates to the JIF will not take effect and new services will not be advertised since the server won't know of the changes.

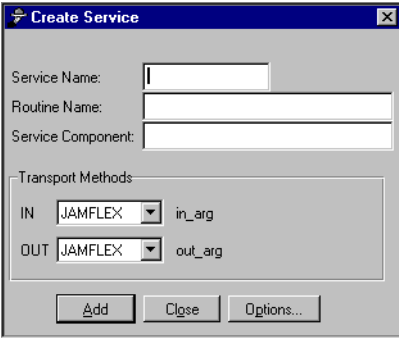
Determine the correct host name (`SMRBHOST`) and port value (`SMRBPORT`), and under UNIX, the `broker.bin` (`SMRBCONFIG`) location. Try connecting to the middleware by entering the correct values in the Connect dialog box.

Define a service

Define services to carry out basic database transaction requests. For the screen you just created with the screen wizard, you want to define services that let users add, update, select, and delete distributor records.

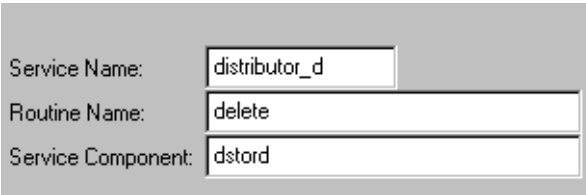
- 4 Choose Service→Create or .

The Create Service dialog box opens.



The image shows the 'Create Service' dialog box. It has a title bar with a maximize button and a close button. The dialog contains three text input fields: 'Service Name', 'Routine Name', and 'Service Component'. Below these is a section titled 'Transport Methods' containing two rows. The first row has 'IN' on the left, a dropdown menu with 'JAMFLEX' selected, and 'in_arg' on the right. The second row has 'OUT' on the left, a dropdown menu with 'JAMFLEX' selected, and 'out_arg' on the right. At the bottom are three buttons: 'Add', 'Close', and 'Options...'.

- 5 Enter `distributor_d` in the Service Name field and press TAB.
The JIF editor supplies the routine name, in this case `delete`, because of the `_d` suffix in the Service Name. It also provides a default service component name—using the prefix of the service name.
- 6 Press TAB to advance to the Service Component field.
- 7 Change the service component name to `dstord` (to match the name you provided in the editor when you saved the service component to `server.lib`).



The image shows the 'Create Service' dialog box with the following values entered: Service Name: `distributor_d`, Routine Name: `delete`, and Service Component: `dstord`.

- 8 Choose Add.

The fields clear so that you can create another service.

More About Service Options

There are additional service specifications you can define in the JIF that let you control such things as:


- Which transaction a service implements.
- When a service component is opened and cached to memory.
- Behavior of the service itself when it is called.

By default, the JIF uses the service name to determine the transaction type; therefore, a service name that ends with `_i` sets the service to the transaction manager Insert operation. In addition, the Cache Service Component option by default is set to open or cache a service's service component when the service is advertised by the server. Repeat steps 5 through 8 for each of the other services: `distributor_i`, `distributor_s`, and `distributor_u`. Remember to provide the correct name of the service component, `dstord`.

- 9 Repeat steps 5 through 8 for each of the other services: `distributor_i`, `distributor_s`, and `distributor_u`. Remember to provide the correct name of the service component, `dstord`.
- 10 After you create all four services, choose Close to close the Create Services dialog box.

The View Services dialog box displays the services you created.

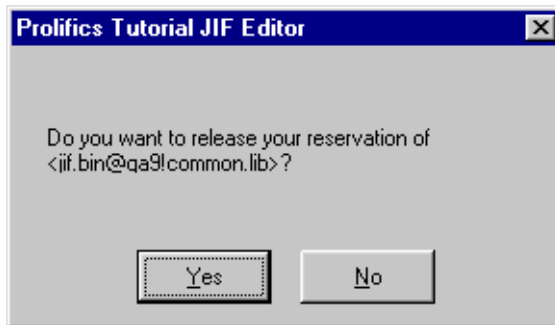


- 11 Choose File→Save or  .

Since you are connected to the middleware, servers are notified immediately when changes are saved to the JIF. In addition, new services (depending on the advertise specification and if the server is configured to advertise all services) are advertised and immediately available to your application.

12 Choose File→Exit.

You are prompted to release the reservation on the JIF since it is stored in a remote library and is shared by others.



13 Choose Yes to release the reservation.

The JIF editor closes.

What did you do?

In this lesson, you invoked the JIF editor and:

- Created services devised by the screen wizard for your distributor/orders service component.
- Defined services that will be called by the client to select, update, insert, and delete distributors and their orders.

What did you learn?

You learned:

-
- The JIF stores service definitions.
 - The JIF is accessed by client and server to determine what parameters and information should be used when processing service calls.
 - The JIF editor can be invoked while you are working in the editor.
 - When you are connected to the middleware, new and changed services are advertised by the server and are immediately available to your application.
 - When you edit a `jif.bin` that is shared by the entire development team, you hold a reservation on that file.



9 Testing the Screens

You now have a client screen and its corresponding service component. Now you are ready to test the screens to make sure they function appropriately.

In this lesson you learn:

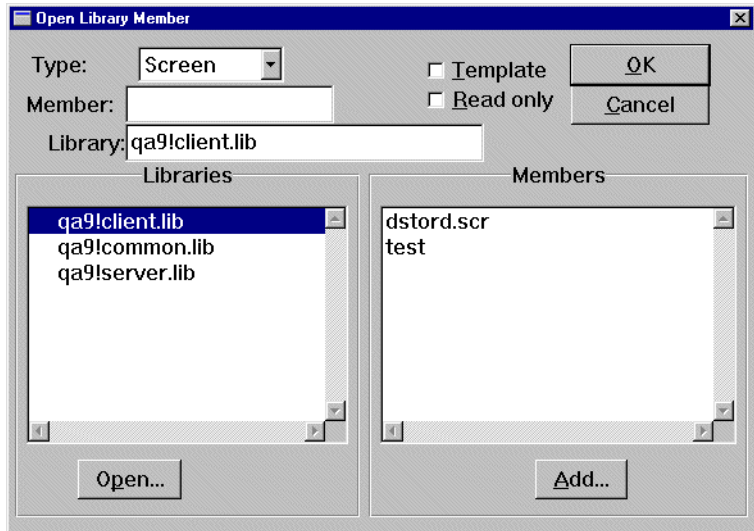
- To test the client screen and the services it uses.
- How the transaction manager controls an application's behavior and appearance.

-
- 1 If necessary reactivate your application, invoke the editor, and connect to the middleware.

If you closed the `dstord.scr` client screen, you need to reopen it:

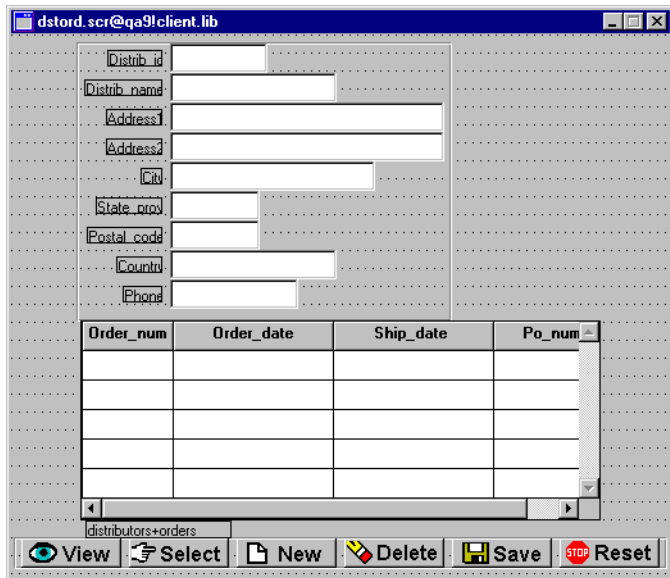
- 2 Choose File→Open→Screen or  .

The Open Screen dialog box opens.




- 3 Double-click on `dstord.scr` in the `client.lib` library on the application server.

The client screen opens in the workspace.

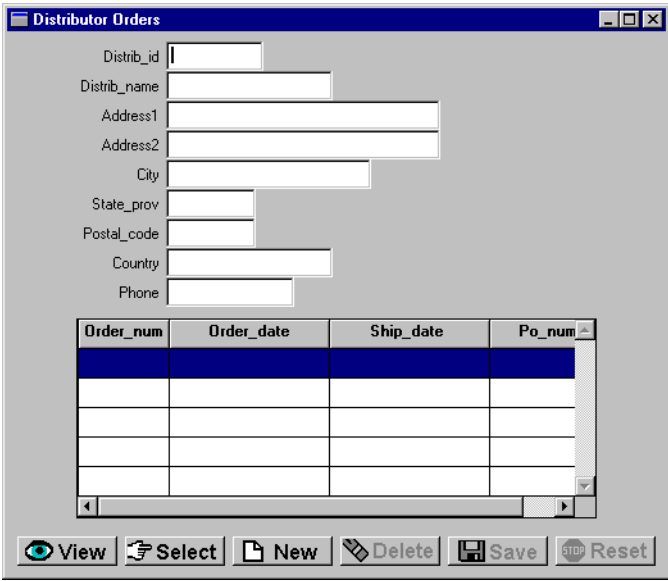


Access test mode

At any time, you can test a screen that you are editing to see how it behaves and appears to an end user.

- 4 Bring focus to the client screen, `dstord.scr`.
- 5 Choose File→Test Mode (press F2) or .

The Distributor Orders screen opens in Test mode. All text fields are enterable.



Order_num	Order_date	Ship_date	Po_num

More About Test Mode

Any changes you make to a screen in the editor can be tested immediately without saving or compiling your edits so you are free to experiment without committing to the changes. However, you must save service components to the server library to test service calls.

Test mode is fully functional. Your client screen appears and behaves as it would in the final application. All screen attributes and logic can be executed and tested (including data validation, database interactions, and 4GL/3GL code). Also, you can test any screen that is called by the current, or active, screen.

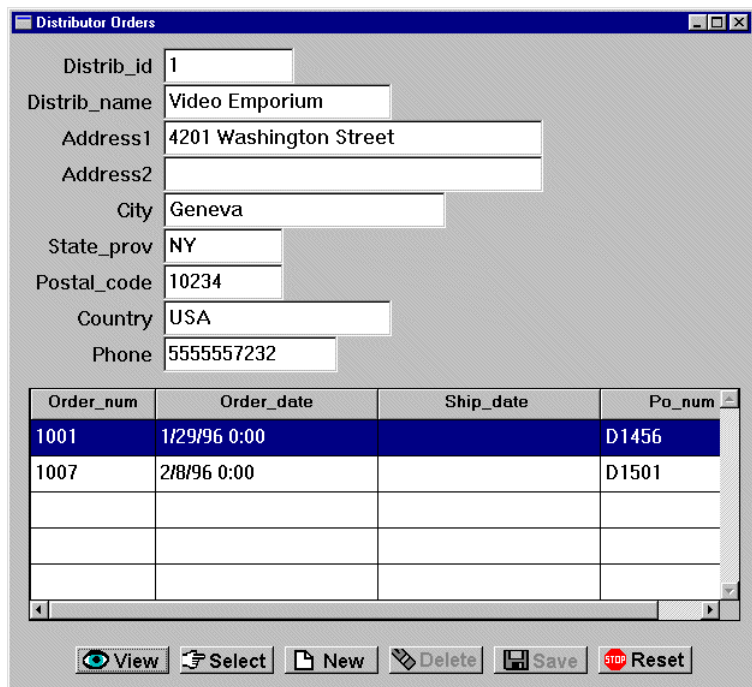
Test mode is flexible and powerful. It lets you verify your application design in its component modules as well as test the entire application as a whole.

View data

You can access “real” data with the screen that you just built. The commands associated with the screen's buttons let you access and maintain data in the database.

- 6 Choose  .

The first record in the distributors table displays.



Order_num	Order_date	Ship_date	Po_num
1001	1/29/96 0:00		D1456
1007	2/8/96 0:00		D1501

After you press View, the Save and Delete buttons become inactive. The active or inactive state of buttons depends on the last command to execute. In this case, the request to view records invalidates requests to save or delete records.

- 7 Try to enter data in any field. Panther prevents data entry because the View command only allows read access.

Panther's transaction manager protects widgets from data entry by the style that it applies to each one. Styles can set a widget's color and protections. In this case, they activate and deactivate (gray out) push buttons without requiring you to write any code. You can change the default styles with the styles editor.

More About the Transaction Manager

You can create complex database query/update screens without having to write any code. That's because the transaction manager "knows" about the interaction between database tables and columns (via information retrieved from the database during the importation process). Given this information, the transaction manager generates the appropriate SQL statements for fetching or updating the database, and keeps track of any data changes. When your application issues the SAVE command, the transaction manager automatically generates SQL commands to update the database to match the data on the screen.

Edit the data

After you select a record for update, you can change its contents. When you choose to update a record, by default Panther protects the primary key fields from data entry. This is a result of Panther's application of a style to each widget.

- 8 Choose  Select .

The Select command selects a record for update. The first record in the distributors table displays.

- 9 Try to enter data in the `distrib_id` field.


Panther prevents changes to `distrib_id` because it is a primary key in the distributors table; in general, primary key fields in database tables cannot be changed.

- 10 Click in or tab to the Address2 field and enter P.O. Box 133. Here you can enter data and edit existing data on the screen.

Distrib_id	1	protected from user input
Distrib_name	Video Emporium	updatable
Address1	4201 Washington Street	
Address2	P.O. Box 133	
City	Geneva	
State_prov	NY	
Postal_code	10234	
Country	USA	

Save the changes

To save your changes to the database, you must issue a Save command.

- 11 Choose  Save .

Panther calls the update service `distributor_u` to update the database.

- 12 Choose  Reset .

All values are cleared from the fields and the Reset command closes the current transaction, so that you can execute another transaction command.

Add a new record


You can also add a new distributor record to the database.

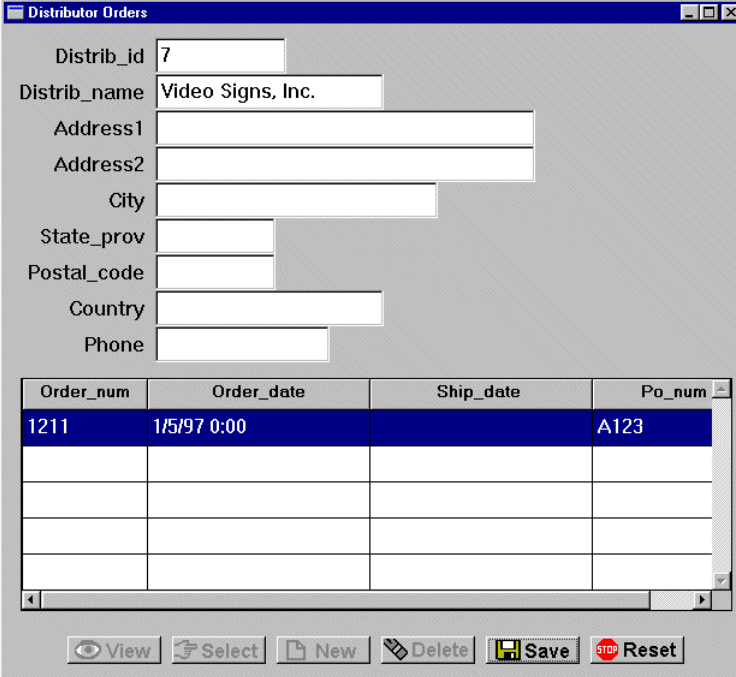
- 13 Choose  New .

After you choose New, only the Save and Reset buttons are active. All other buttons are grayed and unavailable. As before, styles control the appearance of the buttons.

- 14 Enter values in each of the following fields:

In this field	Enter the following value
Distrib_id	7
Distrib_name	Video Signs, Inc.
Order_num	1211
Order_date	01/05/97 0:00
Po_num	A123

15 Choose  Save.



Order_num	Order_date	Ship_date	Po_num
1211	1/5/97 0:00		A123

You just updated the server database via the middleware. The local database remains unchanged. Recall that the tutorial server that you configured in Lesson 2 automatically declares a database connection to the `vidsales` database on the server. So, you don't have to connect every time you start up the editor.

Note: The JDB model enforces unique primary keys (`distrib_id`, in this case). Therefore, to add a new record, you must enter an ID that is unique in the database. A message is displayed if the entered ID already exists. In this case, enter a different number. Lesson 13 shows how to programmatically assign new and unique distributor ID numbers.

16 Choose  .

More About Wizard-Generated Buttons

The transaction-specific buttons generated by the screen wizard let users update, insert, select, and delete database records. In general, the buttons operate on the master table and any other updatable tables on the screen. However, on some screens, the default behavior might have unwanted results. For instance, the Delete button on the `dstord` client screen deletes the master and the associated details. Because the order items associated with the detail are not present on the screen, these would be orphaned. Therefore, you might want to remove the Delete button from this type of screen.

The Transaction menu options in test mode offer functionality that is equivalent to the buttons. Some options, particularly Continue, are invalid in a three-tier model.

17 Return to the editor in one of these ways:

- Choose Options→Editor (press Shift+F5). (Windows only) Press Esc twice.

You exit test mode and the editor workspace reopens.

What did you do?

In this lesson, you tested the Distributor Order screen by performing these tasks:

- Accessed test mode from the editor workspace.
- Executed a variety of database commands that let you view, update, and add data in the `vidsales` database.
- Saved changes to the database.

What did you learn?

You learned:

- The screen wizard creates screens that can be tested and used almost immediately.
- The transaction manager knows about the interaction between database tables and columns. It automatically builds the appropriate SQL statements. It also knows when to activate and deactivate specific widgets (primary keys and command buttons) on the client screen, and thereby cue users which actions they can take next.
- The push buttons that are created by the screen wizard can handle most database requests. With the screen wizard, you can build screens that retrieve and update data without writing a single line of code.



10 Setting Properties to Query the Database

In this lesson, you enhance the Distributor Orders (`dstord.scr`) screen so users can query the database for a specific distributor record. You learn about Panther widget properties and how to set them appropriately.

Properties can be set and changed locally for the current screen and its components, and globally for objects throughout the application:

- Properties for some objects should be set on a screen-by-screen basis, because the behavior or appearance of the object depends on how it is used. For example, on a data maintenance screen, you might want to enforce data entry in a field by setting its `Required` property to `Yes`, while on another screen, the equivalent field does not have the same requirement.
- Objects throughout your application can inherit their property settings from the repository. For example, you might set a widget's validation, font specification, size, and format in the repository. All widgets that inherit from this repository object have the same appearance and behavior.

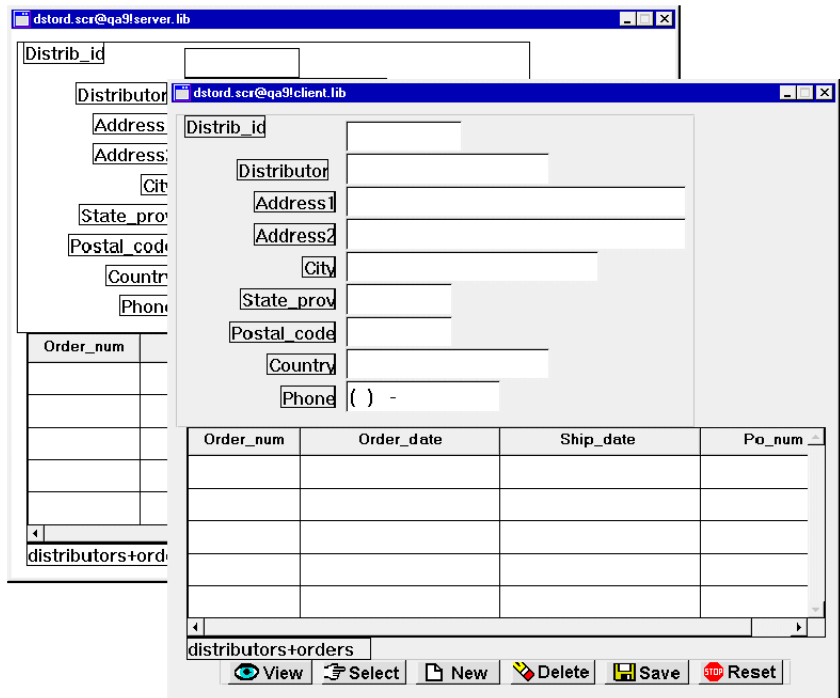
In this lesson, you set properties that allow users to search the database for a specific distributor by entering an identification number. The transaction manager automatically generates the appropriate SQL query statement based on this input, submits the input to the middleware API, and ultimately returns the desired distributor data to the client.

In this lesson you learn how to:

- Set widget properties on both the client screen and its corresponding service component.
 - Specify a particular widget to act as a query field.
 - Test the resulting query screen by searching for specific distributor records.
-

1 If necessary:

- Reactivate the application, invoke the editor, and connect to the middleware.
- Reopen the client screen `dstord.scr` in `client.lib` and the service component `dstord.scr` in `server.lib` on the application server.



Using the Properties window

You can define the appearance and behavior of screens and widgets through the Properties window. If the Properties window is not open in the editor workspace, open it with one of these actions:

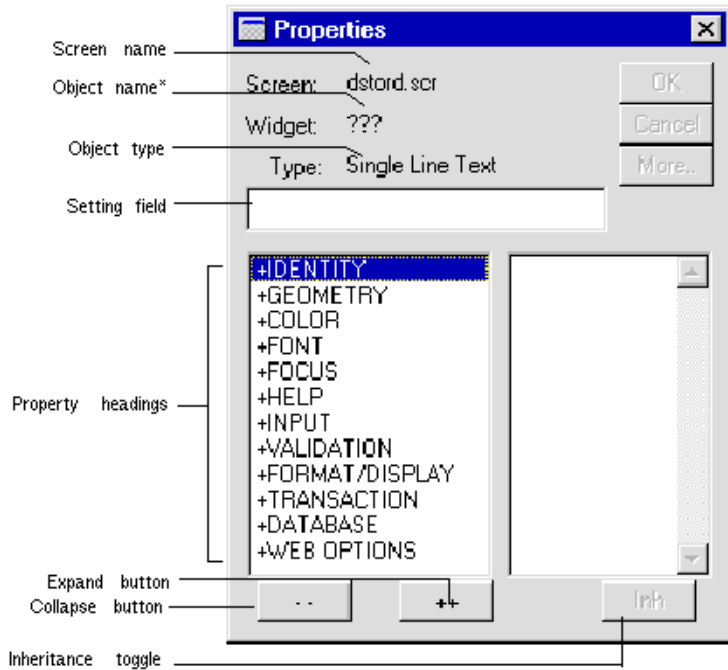
- Double-click on a widget or on the screen.
- With focus on the screen or a widget, press Enter.
- Choose View→Properties Window.

More About the Properties Window


The Properties window lets you easily view and set properties for all Panther objects—for one object at a time or for multiple objects simultaneously. If multiple objects are selected, the Properties window displays those properties that are common to all, so you can assign the same font, colors, and formats. If the selected objects have different values for a given property, three question marks (???) are displayed as the property value.

When no widgets are selected, the Properties window displays the properties of the current screen.

Properties are grouped by category under descriptive headings on the left. Initially, the properties list is collapsed and displays only headings.



*Three question marks indicate that more than one object is selected, and the selected objects do not share a common value for the given property.

- 2 In the Properties window, choose .

All property headings expand, displaying the properties and their respective values on the right. To change a property value, select the property and type or select a new value in the Setting field/option menu of the Properties window.

You can also expand and collapse headings individually. To expand or collapse a single heading, simply click it.

- 3 Choose .

The properties list collapses, displaying only the property headings.

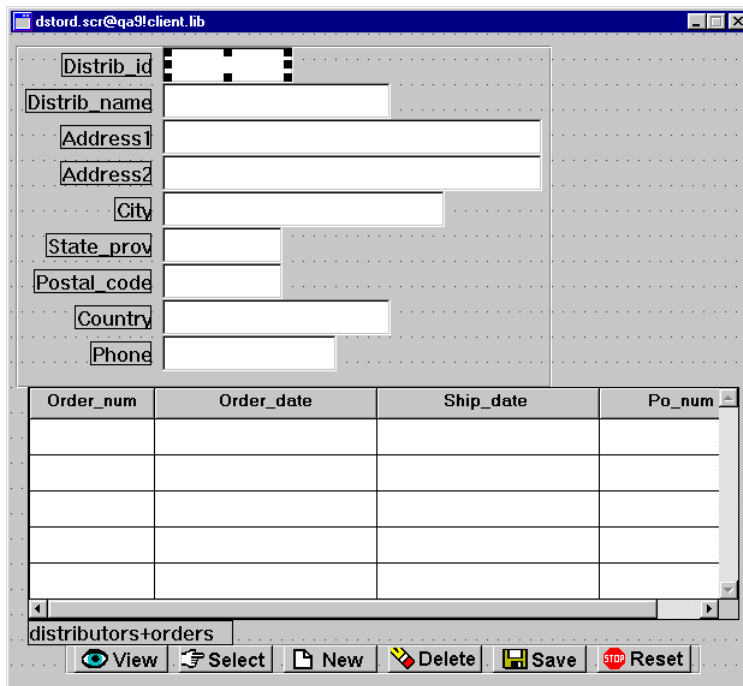
No matter how you set a property, you can reverse any change with

Edit→Undo or .

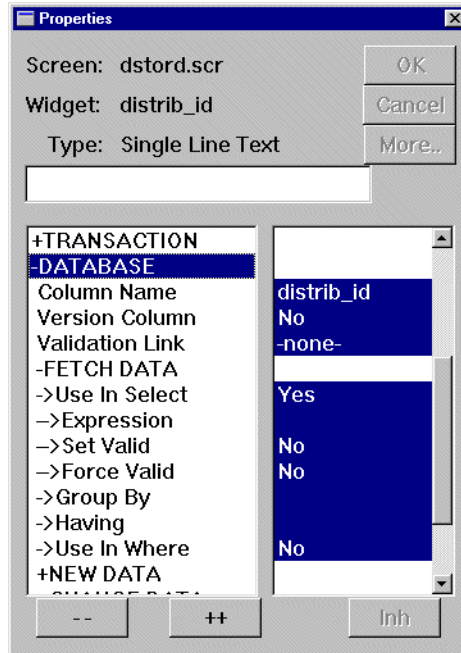
Change properties locally

The next several steps show how to set a property on a widget at the screen level. You need to set a Database property for the `distrib_id` widget so that users can obtain information about a specific distributor and its orders via its ID number. The ID number is used to search for a matching record in the database.

- 4 On the client screen `dstord.scr`, select the single line text (data entry) widget `distrib_id`, which is adjacent to the `Distrib_id` label.



- 5 In the Properties window, select the Database heading. Database-specific properties are displayed.



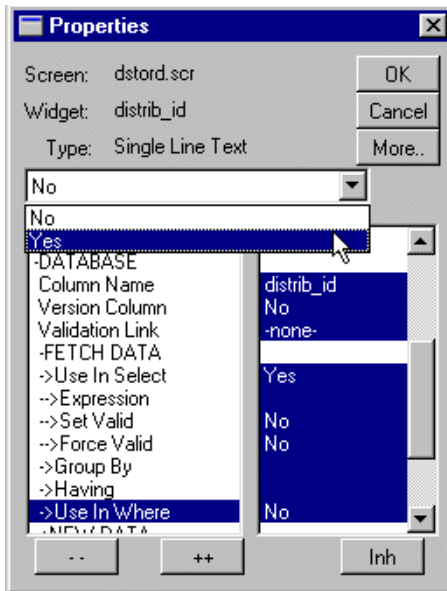
- 6 Under Fetch Data, select the Use In Where property.

An option menu is made available in the Properties window. Here you select from a list of predefined values.

More About Setting Predefined Property Values

You can set a property with predefined values in several ways after you select it:

- Click directly on the displayed property value to scroll through all possible values.
 - Type the initial character to specify a value. For example, type y for yes or n for no.
 - Select values from the option menu.
- 7 Click on the option menu to display its drop-down list, and select Yes. Press Enter or choose OK.



Yes tells Panther to use the field in the WHERE clause of the database query. When you change the Use In Where property to Yes, two other changes occur:


- Related subproperties display, including the Operator property. By default, the Operator property is set to = (equal sign).
- The Use In Where property value no longer displays in reverse video. This indicates that the inheritance link no longer exists for this property—in other words, the property no longer gets its value from the repository.

Database property settings give Panther's transaction manager the information it needs to build an SQL statement to fetch, display, and update the requested record. At runtime now, when a distributor ID is entered in the `distrib_id` widget and a View or Select command is issued, Panther searches the `distrib_id` column in the distributors database table for a record with a matching ID.

More About Inherited Property Values

The Properties window uses reverse video to show which property settings are inherited. The Inherit From property under the Identity heading identifies the source of inheritance. Objects within the repository that are imported directly from a database have their Inherit From property set to @DATABASE.


Screens that are created by the screen wizard inherit screen, push button, and grid property values from prototype wizard-specific repository entries, while labels and data entry-type widgets inherit their property values from database-derived repository entries.

- 8 With focus on the `dstord.scr` client screen, choose File→Save or  .
The screen is saved to `client.lib` on the application server.

Edit the service component

It is important to keep the service component and its calling client screen synchronized, because the service component must perform the actual passing of data to and from the database server. In general, all property settings that affect processing of data must be made to both the client screen and its corresponding service component.

- 9 Repeat steps 5 through 8 for the `distrib_id` single line text widget on the `dstord.scr` service component.


- 10 Choose File→Save or  .

The service component is saved to `server.lib` on the application server.

View specific records

Now test the client screen to find a specific distributor's records.

- 11 Bring focus to the `dstord.scr` client screen.

- 12 Choose File→Test Mode or  .

The Distributor Orders screen opens in test mode.

- 13 Enter 3 in the `distrib_id` field.

- 14 Choose  .

The distributor record that contains ID number 3 displays.

The screenshot shows a window titled "Distributor Orders" with the following fields and values:

Distrib_id	3
Distrib_name	Videos Tonight
Address1	2501 River Road
Address2	Lafayette Square Mall
City	Needham
State_prov	NY
Postal_code	10236
Country	USA
Phone	5555554040

Order_num	Order_date	Ship_date	Po_num
1004	2/2/96 0:00		D1472
1008	2/8/96 0:00		D1501

At the bottom of the window, there is a toolbar with the following buttons: View, Select, New, Delete, Save, and Reset.

- 15 Choose .

The Reset button executes a Close command, which closes the search transaction and clears all fields of data so that you can search for another record.

- 16 Enter 7 in the `distrib_id` field.

- 17 Choose .

The distributor record that you added to the database in Lesson 9 displays and can be edited.

- 18 Type 5551234567 in the Phone field.

The screenshot shows a window titled "Distributor Orders". It contains a form with the following fields:

- Distrib_id: 7
- Distrib_name: Video Signs, Inc.
- Address1: [Empty]
- Address2: [Empty]
- City: [Empty]
- State_prov: [Empty]
- Postal_code: [Empty]
- Country: [Empty]
- Phone: 5551234567

Below the form is a table with the following data:

Order_num	Order_date	Ship_date	Po_num
1211	1/5/97 0:00		A123

At the bottom of the window are several buttons: View, Select, New, Delete, Save, and Reset. The Save button is highlighted with a yellow border.

19 Choose  .

The service `distributor_u` is called to update the database with the changes.

20 Choose  .

The transaction closes. You can try all the buttons on the screen. However, you must remember to choose Reset to close each transaction and clear the fields.

Note: Using the Delete button on this particular screen deletes the selected distributor and its orders. However, this also orphans records in the `order_items` table that are associated with the deleted orders records. Therefore, you should not delete distributors via the `dstord.scr` screen.

21 Press Shift+F5 (Esc twice or choose Options→Editor) to exit test mode and Return to the editor.

What did you do?

In this lesson, you created a query screen that lets a user enter data that is used to search for a specific database record. You did this by performing these tasks:

- Define a query field on both the client screen and service component by setting their Use In Where property.
- Test the resulting screen by entering search criteria.

What did you learn?

You learned:

- The Properties window lets you set an object's behavior.
- Inherited property values can be selectively overridden, without affecting inheritance links for other properties.
- It is important to keep client screens and their corresponding service components synchronized with the same data entry widgets.

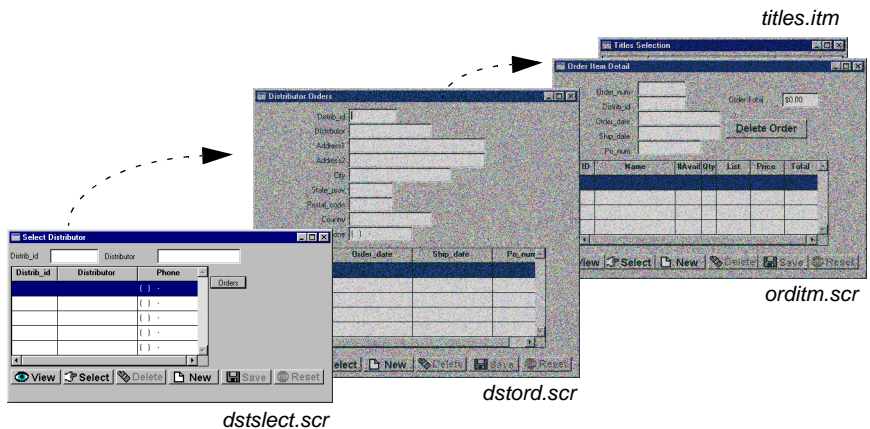


Module 3— Connecting the Screens

OVERVIEW

11 Enhancing the Screen

An additional client screen and its corresponding service component are provided in `tutorial.lib`. They use a grid format to display a list of distributors. In this lesson, you enhance the screen and service component so users can query the database for a specific distributor record through two different search criteria: either a distributor ID number or a partial or full name string. The transaction manager uses user input to generate automatically the appropriate SQL query statement. If a query field is empty, the transaction manager excludes its data from the SQL generation.



In this lesson you learn how to:

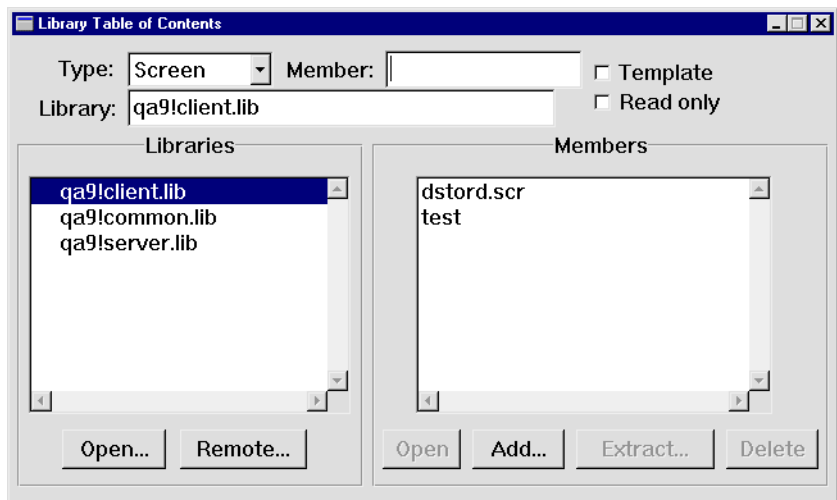
- Resize the screen and copy widgets to it from a repository entry.
- Specify and define more than one widget to act as a query-by-example field, so users can use a variety of search criteria to query the database.

-
- Copy widgets from the client screen to the service component so both have the same data entry widgets and are thereby synchronized.

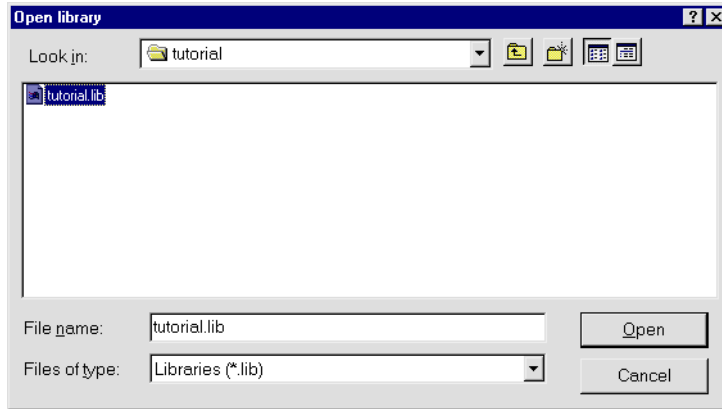
In this lesson, you open a wizard-built client screen and service component from the local tutorial library. You then save them to their appropriate libraries on your remote application server, and enhance the user interface by including query fields.

- 1 If necessary, reactivate the application, invoke the editor, and open a middleware session.
- 2 Choose View→Library TOC.

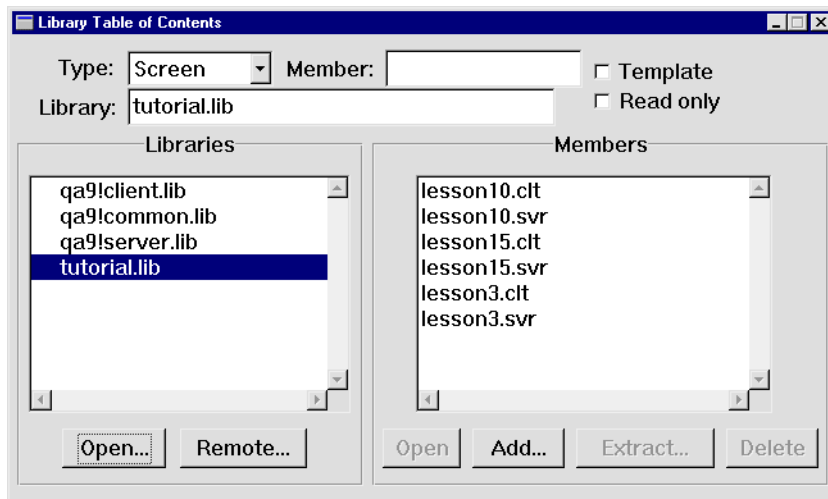
The Library TOC opens.



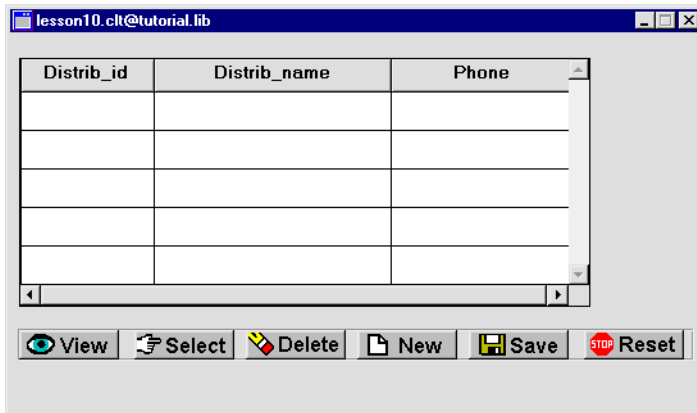
- 3 If `tutorial.lib` is not among the list of open libraries:
 - From the Library Table of Contents, choose Open under the list of open libraries. The Open Library dialog opens.



- Select `tutorial.lib`.
 - Choose Open. The Library TOC redisplay.
- 4 Select `tutorial.lib` from the list of open libraries and `lesson10.clt` from the list of library members. Choose Open.



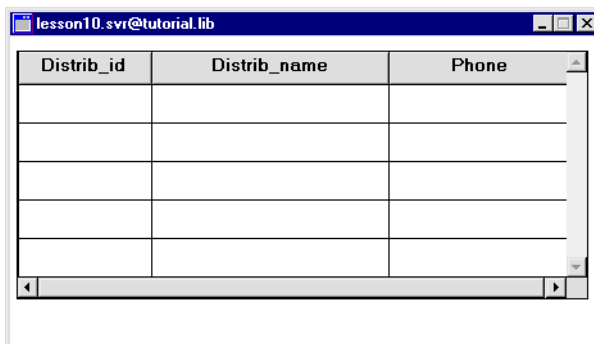
The `lesson10.clt` screen opens in the editor workspace.



This screen is a master-only screen for the distributors table and uses a grid format. The grid contains three of the table's columns: `distrib_id`, `distrib_name`, and `distrib_phone`.

- 5 Choose File→Save As→Library Member. The Save Screen dialog opens.
- 6 Save the screen as `dstslect.scr` in `client.lib` on the application server.
- 7 From the Library Table of Contents dialog, select `lesson10.svr` from the list of library members.

The service component, `lesson10.svr`, opens in the editor workspace.



This screen contains the same grid as the client screen.

- 8 Choose File→Save As→Library Member and save `lesson10.svr` as `dstslect.scr` in `server.lib` on the application server.

Access table view properties

Since the client screen and service component were created using the screen wizard, the services that will be requested by the client screen were also generated. These values are properties of the client screen's table view. Select the screen's invisible table view widget and determine the names of its Service properties so that you can define those services in the JIF.

More About Table Views

A table view widget is automatically created on a repository screen when you import database objects. Table views store the following types of database information:

- Primary key
- List of table columns
- Database attributes such as, sort order, distinct, etc.
- Service specifications

Panther's transaction manager uses the information stored in table views (and links which specify the join relationships between multiple table views) to determine what SQL statements should be generated for each transaction command.

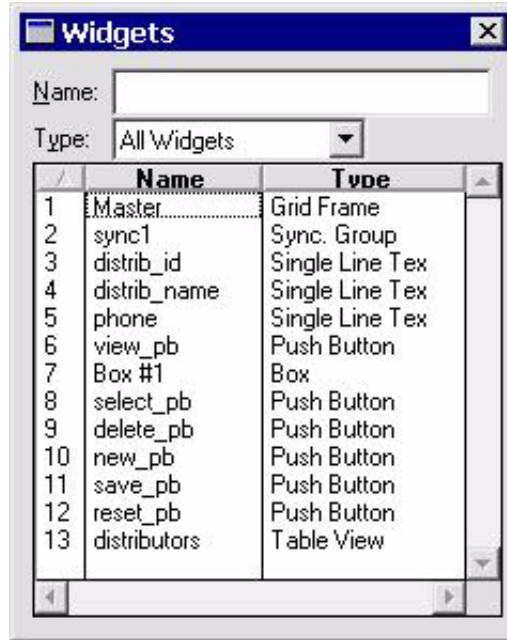
The JetNet transaction model uses the service information to determine how to respond to service calls.

In addition, table views provide you with a quick entry point for modifying the default transaction manager behavior.

You can select the table view widget by using either the [Widget List](#) or the DB Interactions window in order to gain access to its properties.

- 9 Bring focus to the `dstselect.scr` client screen and choose View→Widget List.

The Widget List opens and lists all the widgets on the current screen: the widget's name, field number or contents is in the middle column and its type in the right column.



More About the Widget List

You can use the Widget List as an alternative way to select widgets. All widgets on the current screen are listed in the Widget List, including invisible widgets, such as selection groups, synchronization groups and table views.

When you select an item from the list, the widget on the screen is also selected. The Properties window displays the properties common to the widgets that are currently selected, or of the screen if no widgets are selected.

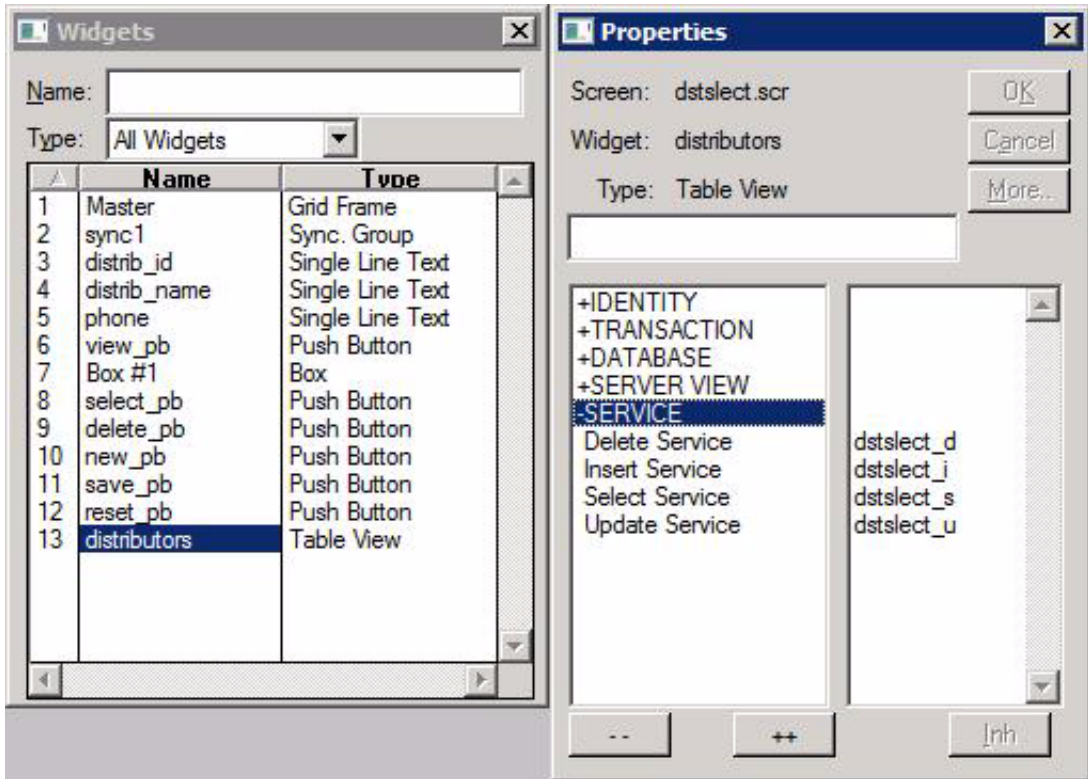
You can select multiple contiguous widgets in the list with a click+drag or Shift+click; Ctrl+click to toggle membership in the selection set or to select non-contiguous items.

- 10 Select distributors from the list of names. It is identified as the `distributors` table view widget.

The table view properties are displayed in the Properties window (Table View displays in the Type field).

- 11 Expand the Service heading in the Properties window.

The Delete Service, Insert Service, Select Service, and Update Service properties use the `dstselect` prefix for the names of the services used by this screen.



More About Wizard-Generated Service Names

When the screen wizard generates screens for a three-tier model, you will recall that it also creates services that the screen will use. It uses the master table on the client screen to determine the name of the services. Since the client screen, in this case, uses the `distributors` table as its master table, the services, by default, would have `distributor` as the prefix. Service names must be unique within the JIF, and since services with this prefix were already defined for a screen you created earlier, different services needed to be specified.

The services used by a screen are specified in the Service properties associated with the client screen's root table view. The screen wizard automatically sets

these property values based on input you provide on the screen wizard Service Definition dialog box.

Refer to Chapter 31, “Building a Transaction Manager Screen.” of the Developer’s Guide for more information on root table views and table view processing.

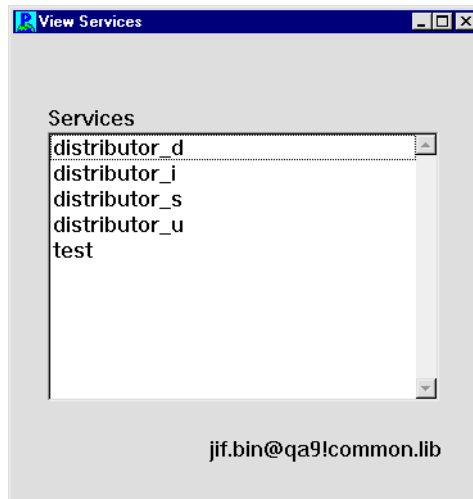
Update the JIF

When you add or change services in the JIF, Panther rereads the JIF, which in turn, causes the application server to readvertise services (assuming the server was configured to do so).

In the case of the tutorial, you did configure the standard server to advertise all services in the JIF. Therefore, when you add services to the JIF, they are made immediately available to your application.

- 12 Invoke the JIF editor and connect to the middleware (refer to [Lesson 8](#) for details on using the JIF editor).

The services in `jif.bin`, located in the remote `common.lib`, are displayed in the View Services dialog box.



- 13 Define the four services needed for the `dstslect` client screen: `dstslect_d`, `dstslect_i`, `dstslect_s`, and `dstslect_u` (refer to [Lesson 8](#) for details on creating services). In each case, name the service component `dstslect`.

14 Choose File→Save.

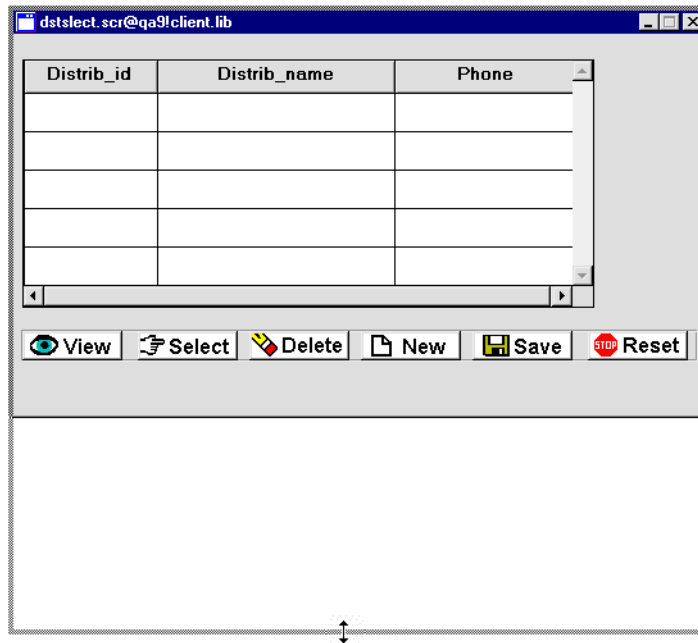
15 Choose File→Exit to close the JIF editor and return to the editor workspace.

Resize the screen

Increase the screen's vertical dimension so you can add other widgets to it.

16 In the editor workspace, resize the `dstselect.scr` client screen in one of the following ways:

- Drag on the upper or lower edge of the screen until it is the size you want.



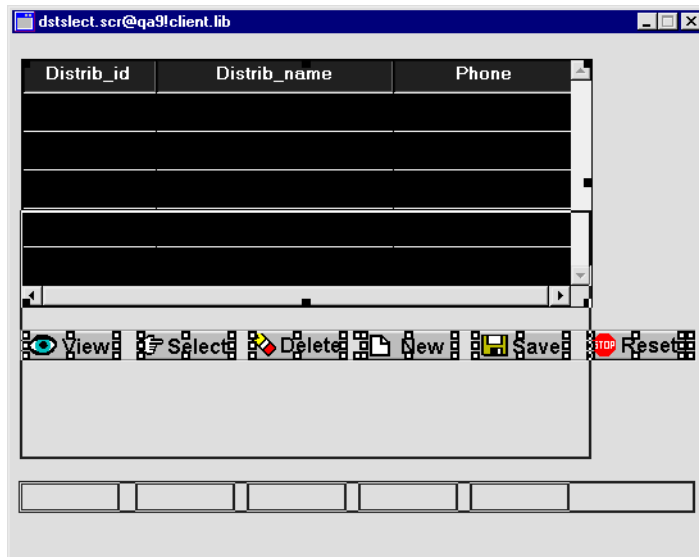
- Click in an empty area of the client screen to deselect all widgets.

The screen properties are displayed in the Properties window. Under Geometry, set the screen's Height property to the desired size (default is in grid units).

Move widgets

Make room for more widgets above the grid widget by moving the grid widget and push buttons down to the screen's lower portion.

- 17 Choose Edit→Select All and drag the widgets to the bottom of the screen, leaving space at the top for more widgets.

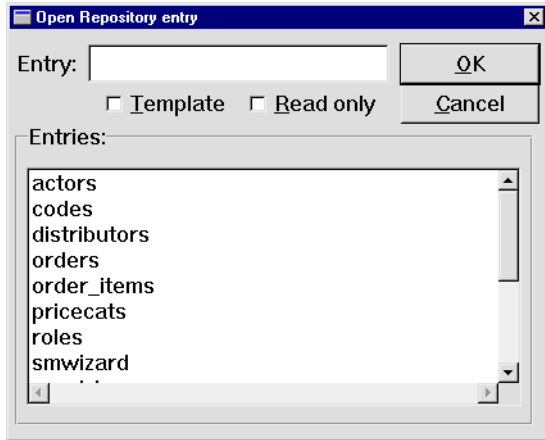


Open a repository entry

You want to populate the screen with widgets that are associated with a particular database table. You can access these widgets in the repository, just as the screen wizard does.

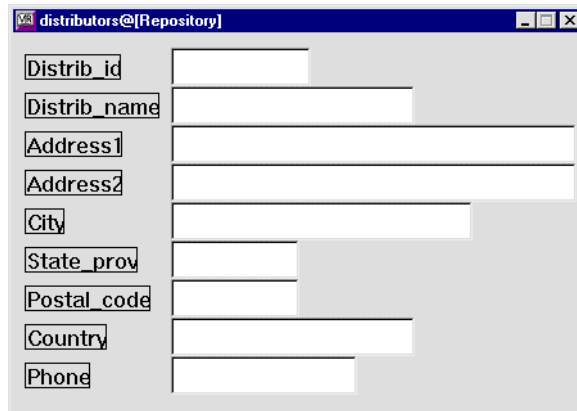
- 18 If the repository is not open, choose File→Open→Repository, and select `data.dic` in the `pro1tut` directory.
- 19 Choose File→Open→Repository Entry.

The Open Repository Entry dialog opens and displays the contents of the `data.dic` repository.



- 20 Select the distributors repository entry and choose OK.

The distributors@[Repository] window opens.



Copy widgets

You can use widgets from the repository to serve as query fields on the `dstselect.scr` screen. When you create a copy of a repository widget, the copy has an inheritance link to its parent in the repository. You can use inheritance to propagate changes from the repository to application screens and service components, as shown in the next lesson.

In the following steps, you copy `distrib_id` and `distrib_name` from the repository to the client screen `dstselect.scr`. The copies inherit their property values from the repository.

- 21** With focus on the `distributors` repository screen, Shift+click to select the `Distrib_id` label and its corresponding text widget (`distrib_id`), and the `Distrib_name` label and its corresponding text widget (`distrib_name`).

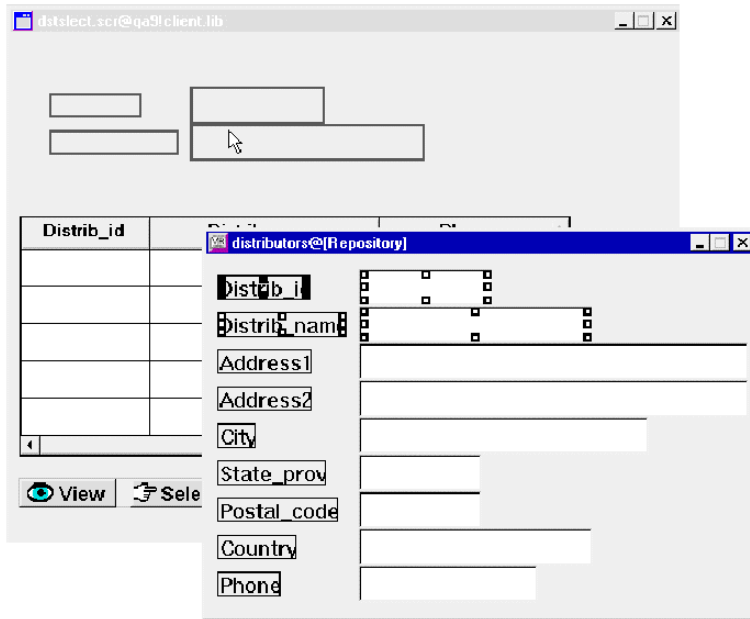
Selecting multiple widgets creates a selection set, which is useful for defining common property values. The first widget you select is the dominant widget. You can Ctrl+click on another widget in the selection set to make it dominant.

More About Selecting More than One Widget

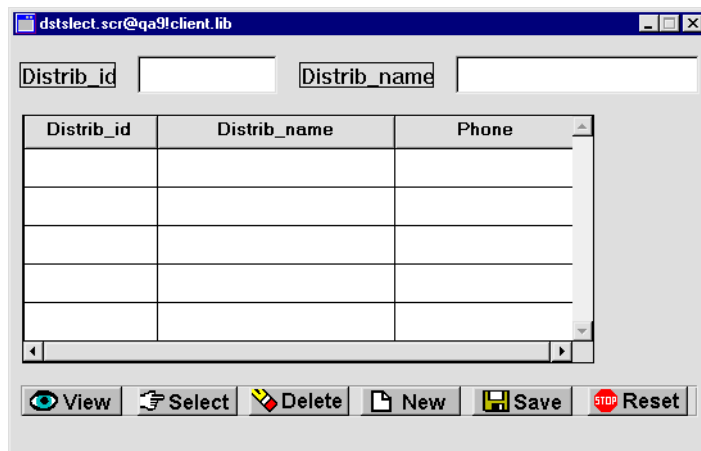
When more than one widget is selected, the first one you select is considered the dominant selection and is indicated by little solid black squares around its border (square brackets in character mode); all other widgets in the selection set are indicated with hollow boxes (curly braces `{}` in character mode). The position and size of the dominant widget determines how the other widgets in your selection set will align or resize when you use Edit menu or toolbar options.

There are a variety of ways to select multiple widgets:

- Press Shift+click on a widget to add or remove it in the selection set.
 - Press Cntl+click to select a new widget and make it dominant
 - With the mouse button pressed, drag a bounding box, also known as a rubber band, around the desired widgets. Any widgets that fall within the rubber band boundary are selected.
 - Use the Widget List (refer to page 10-3) in *Using the Editors*.
- 22** Drag the widgets from the repository screen `distributors` to the top of the screen `dstselect.scr`.



- 23 (Optional) Bring focus to the distributors repository entry and choose File→Close→distributors. The repository window closes.
- 24 Arrange the widgets so they are horizontally aligned at the top of the screen, as shown below. Use alignment options from the Edit menu or toolbar.



Name the widgets

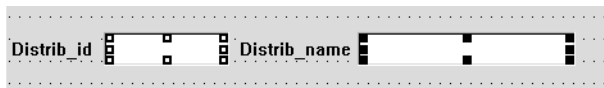
It is good practice to name all data entry widgets, especially if you need to access them programmatically. Names of all widgets on a screen must be unique. Because the `dstslect.scr` screen already contains widgets named `distrib_id` and `distrib_name`, the copies from the repository are left unnamed. You need to assign different names to the copies via their Name property.

- 25** Select the copied text widgets on the client screen `dstslect.scr` and set each one's Name property (under Identity) as follows:
- `distid_qbe`—This widget will serve as a query-by-example field.
 - `distname_qbe`—This widget will serve as an alternative query-by-example field where users can enter a full or partial name string.

Define the query fields

Use database properties to provide the transaction manager with information it needs: identify the query fields, define the data to retrieve, and ensure that query field data is not used to update the database.

- 26** Select the `distid_qbe` and `distname_qbe` widgets.



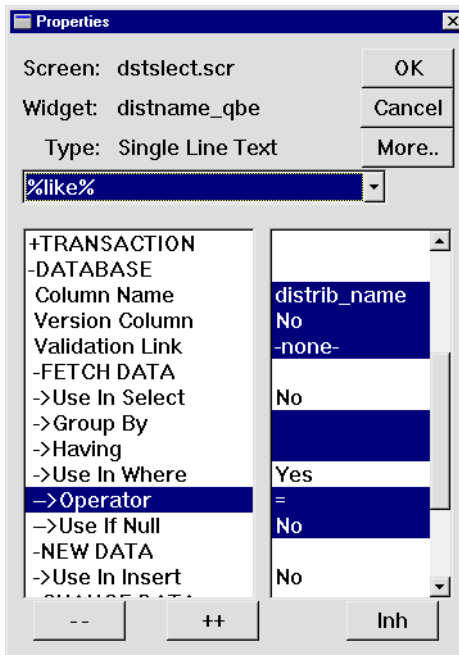
- 27** Under Database, select `CHANGE DATA`. Under `CHANGE DATA`, set the following property for both widgets:
- Use In Update = No
- These settings ensure that the data in these widgets is ignored by `SELECT` or `UPDATE` statements of automatically generated SQL.
- 28** Under Database, select `NEW DATA`. Under `NEW DATA`, set the following property for both widgets:
- Use In Insert = No
- 29** Under Database, select `FETCH DATA`. Under `FETCH DATA`, set these properties for both widgets:

- Use In Select = No
- Use In Where = Yes

Related subproperties display. Leave the Operator property set to =. At runtime, when an ID is entered in `distid_qbe`, Panther searches the `distrib_id` column in the `distributors` table for a record (or row) with the same distributor ID.

30 Select the `distname_qbe` widget.

31 Under the Use In Where property, set the Operator property to `%like%`.

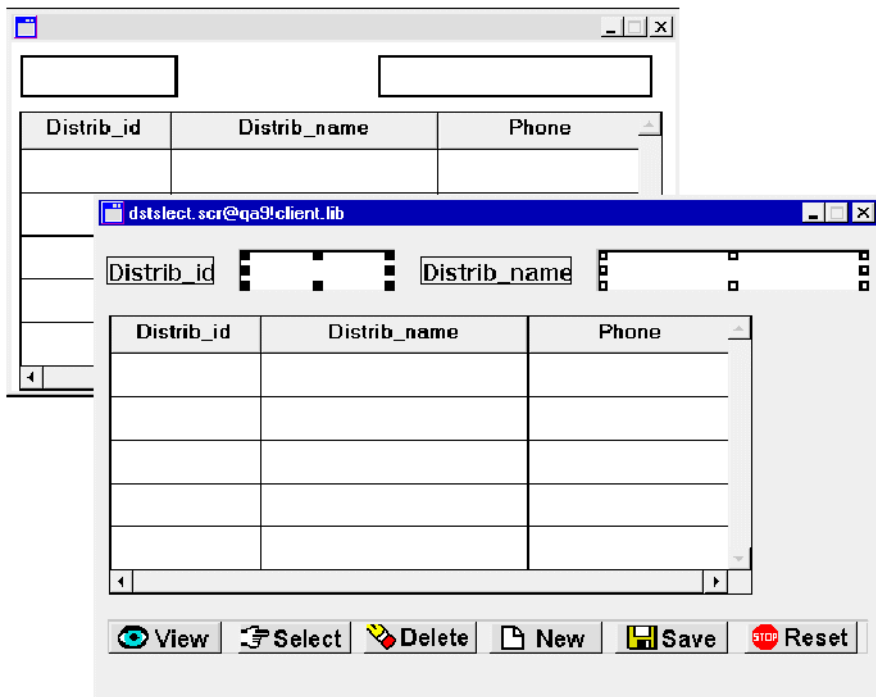


The percent sign (%) sign is a wildcard matching any sequence of zero or more characters. This pattern matching operator tells Panther to search the database for all records that contain the string in the `distname_qbe` field.

Synchronize the service component

Because this is a three-tier model, the service component associated with the `dstselect.scr` client screen must include the new query-by-example widgets that you just copied and modified. Simply copy them from the client screen to the service component; the properties you defined are copied with the widgets. Users never see the service component, so you don't have to worry about how it looks.

- 32 Select text widgets `distid_qbe` and `distname_qbe` on client screen `dstselect.scr`.
- 33 Drag the widgets to any area of the `dstselect.scr` service component.




- 34 Save both the screen and the service component (press F6).


Query the database

You can try it out!

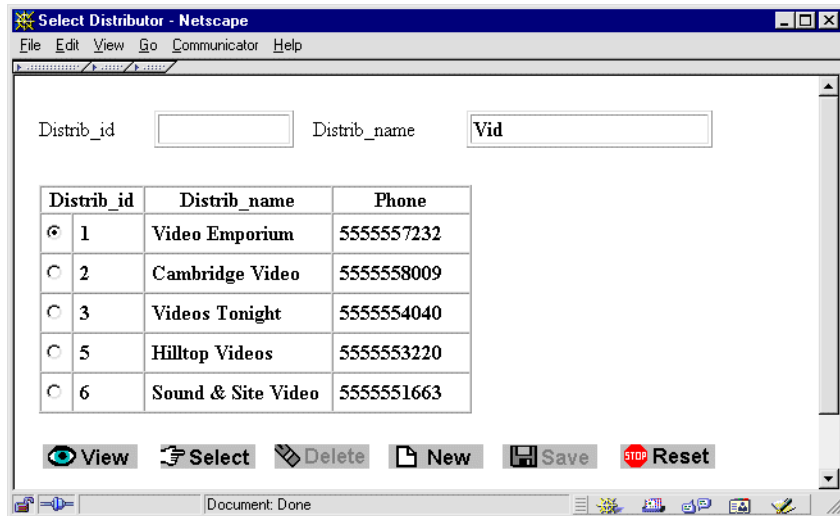
35 Bring focus to the `dstselect.scr` client screen.

36 Choose File→Test Mode or .


37 Type `vid` in the `Distrib_name` field.

38 Choose .

All distributors that have `vid` in their name are listed in the grid.



39 Choose .

40 Type `6` in the `Distrib_id` field and choose .

Panther looks for a record with an exact match—a `distrib_id` with a value of `6`. The record corresponding to ID `6` is displayed in the grid.

41 Now return to the editor.

What did you do?

You enhanced a screen so users can search for distributors by name or ID. You did this by performing these actions:

- Opened the distributors repository entry and copied the desired widgets to your client screen.
- Assigned the appropriate database properties to the copied widgets.
- Copied the query-by-example widgets to the service component.

What did you learn?

You learned:

- Widget names must be unique on a screen.
- Copying widgets copies their property settings as well.
- The editor provides the editing tools you need to enhance the user interface.

12 Inheriting from the Repository

The repository provides a development team with a central storage mechanism and access point for commonly used application objects and database-derived widgets. You can easily modify the contents of the repository and propagate changes throughout an application to client screens and service components alike.

In this lesson you learn how to:

- Control user input by defining an edit mask.
- Edit a repository screen and propagate the changes from the repository to client screens and service components.
- Create a widget on the client screen and copy it to a repository entry for later use.

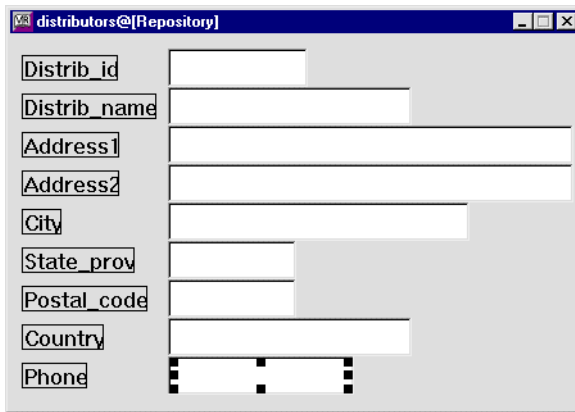
1 If necessary:

- Reactivate your application, invoke the editor, and open a middleware session.
- Reopen the `dstselect.scr` client screen and service component from `client.lib` and `server.lib` respectively.
- Reopen the distributors repository entry: choose `File→Open→Repository Entry` and select distributors from the Open Repository Entry dialog.

Define user input

You can apply an input filter to a widget so it conforms to specific data requirements, such as restricting the length of data or allowing only numeric input. Or, in the case of a telephone number, apply an edit mask so users have a visual cue as to what format is expected when entering data.

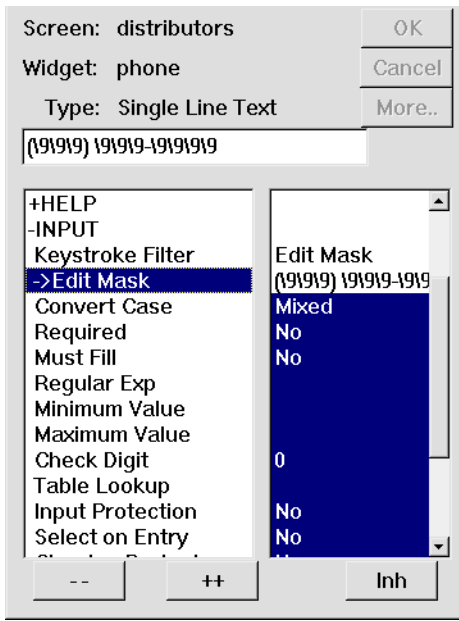
- 2 Select the phone single line text widget (next to the Phone label) on the distributors repository entry.



- 3 Under Input, set the Keystroke Filter property to Edit Mask.

An Edit Mask subproperty appears, where you define the edit mask. In an edit mask, a character that is preceded by a backslash allows a data character to be entered in this position. A character that has no leading backslash is treated as a literal. For example, the string `(\9\9\9)` specifies to display open and close parentheses around three spaces in which the user can only enter digits, such as `(415)`.

- 4 Define the Edit Mask subproperty as `(\9\9\9)\9\9\9-\9\9\9\9`.



The widget displays () - and accepts only numbers as input.

More About Input Filters

Panther provides built-in input filters to help guide user input and, at the same time, reduce validation requirements. Some input controls include:

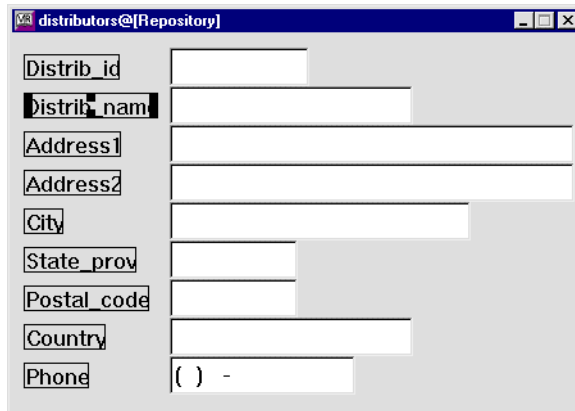
- Digits-only—Allows entry of the digits 0 through 9 only.
- Alphanumeric—Allows entry of any digits, the letters a-z and A-Z, and the space character.
- Yes/No fields—Allows entry of the initial letters of "yes" and "no."
- Case style—Enforces upper-case, lower-case, mixed case conversion.
- Required data—Requires at least one non-blank character for validation to succeed.
- Minimum/maximum value specifications—Specifies a range of values.
- Various protection modes—Protects data from being cleared, or the field from receiving focus, thereby protecting it from data entry.
- Edit masks—Imposes a pattern of symbols or characters that limit the kinds of characters a user can type into a field.

- Regular expressions—Enforces or excludes a specific pattern of letters and/or numbers.
- Table lookups—Verifies input against a list of possible values.

Define what the user sees

You can set properties to enhance specific widgets, for instance, using more descriptive label text or using a different font for data entry widgets. These next few steps show how to set a variety of properties on different widgets. By setting these properties once in the repository, you ensure that they affect the entire application.

- 5 Select the `Distrib_name` label widget on the distributors repository entry.



- 6 Under Identity, change the Label property to Distributor.
- 7 Select single line text widget `distrib_name`.
- 8 Under Identity, change the Column Title property to Distributor.
This property controls the label that appears within the grid widget.
- 9 Select label widget `Ldistrib_id` (`Distrib_id`).
- 10 Under Identity, change the Widget Type property to Dynamic Label.
Dynamic label widgets have a broader scope of properties; therefore, they are more easily manipulated programmatically (more on this in the next lesson).
- 11 Under Geometry, set the Size to Contents property to Yes.

This property lets the widget resize dynamically according to its label content. If the label changes at runtime, the widget size automatically adjusts. This feature is useful for labels that display a name or date whose length is variable.

Propagate changes to screens and service components

Changes made to parent widgets in the repository are visible in child widgets on screens and service components that are open in the editor when the repository entry is saved. To propagate changes to child widgets on other screens and service components, open those screens and service components in the editor, or run the utility `binherit` to propagate changes as a batch process to all the application's screens.

12 While the `distributors@[Repository]` entry has focus, choose `File→Save`.

Notice that the mask you added to the phone widget in the repository entry appears in the phone grid member on both the client screen and service component. Also, the label next to the query field is updated, as is the column title in the grid widget.

More About Inheritance

When you import a table from a database, the text widgets in the resulting repository entry represent columns in the table. These widgets inherit database-related properties from the database. Like the screen wizard, you can use these widgets to build application screens by copying them from repository entry to screen. The result is an inheritance hierarchy of database to repository to screen (and service component). Also, the next time you use the screen wizard, these changes are implemented.

If changes in the database occur such as length specifications, the changed table can be reimported to the repository. These changes are automatically propagated to all application objects that are copies of those repository objects. Also, any custom attributes that you apply to repository objects, such as color, font specification, and validation, can also be defined and propagated to the screens that inherit from these widgets.

Importation and inheritance simplify application maintenance and facilitate the enforcement of a consistent look and feel to an application's interface.

13 Choose `File→Close→distributors`.

The repository entry closes.

Edit inherited property values

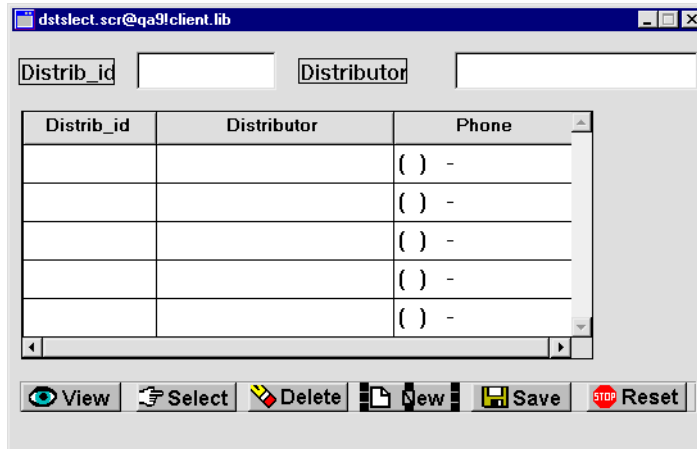
You can modify the behavior established by the screen wizard by writing your own procedure to carry out a particular action. For example, the `dstselect.scr` client screen acts as a search or query screen for finding distributor records, so you might consider using a different screen for adding new distributors to the database. You can change the way the New button behaves through its control string, so it invokes a procedure that opens another screen for adding new records.

More About Control Strings and Control String Syntax

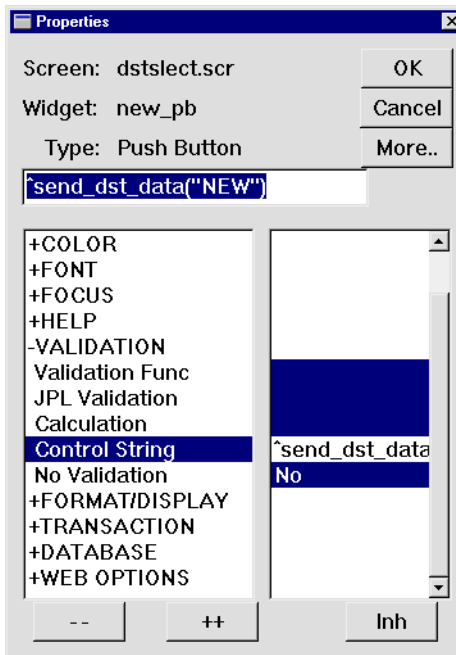
You can attach actions to widgets, menu items, and specific logical keys through control strings. Control strings are a shorthand notation for doing common tasks:

- Execute a function—A caret (^) precedes function names. It tells Panther to search for and execute the named function.
- Display another screen—Supply the screen name to tell Panther to search for and open the named screen, and close all other screens.
- Display another screen as a stacked or child window—An ampersand (&) precedes a screen name. It tells Panther to search for and open the named screen as a stacked window. A stacked window becomes the top window and is the only window that can have focus
- Display another screen as a sibling window—A double ampersand (&&) precedes a screen name. It tells Panther to search for and open the named screen as a sibling of the calling screen. Users can bring focus to any window that is a sibling of the active window.
- Invoke a system command or program—An exclamation point (!) precedes commands. It tells Panther to invoke the specified operating system command.

- 14 Select the New button on the `dstselect.scr` client screen.



- 15 Under Validation, change the Control String property from its inherited property value `^do_new` to `^send_dst_data("NEW")`.



When the user chooses the New button with this control string, Panther invokes the `send_dst_data` procedure and supplies `NEW` as an argument. Lesson 13 describes this procedure.

The inheritance link no longer exists for this property, as indicated by the absence of reverse-video display.

More About Flexible Inheritance

You can override inherited values on individual properties. For instance, to enforce application standards, all application widgets can inherit colors and fonts from their parent widgets in the repository, but you can also define a font or validation routine for a child widget that differs from its parent. This breaks the inheritance link for the given property.

To reestablish inheritance for a property, select the property and choose the `Inh` button on the Properties window.

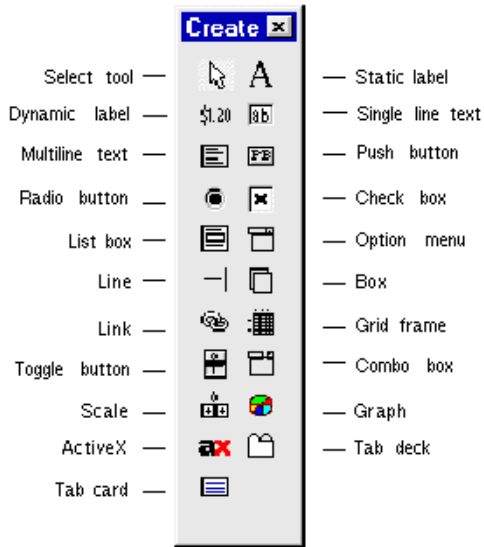
Create a push button widget

Create a push button on the client screen that executes a procedure to send data from this screen to client screen `dstord.scr`, which you created in Module 2.

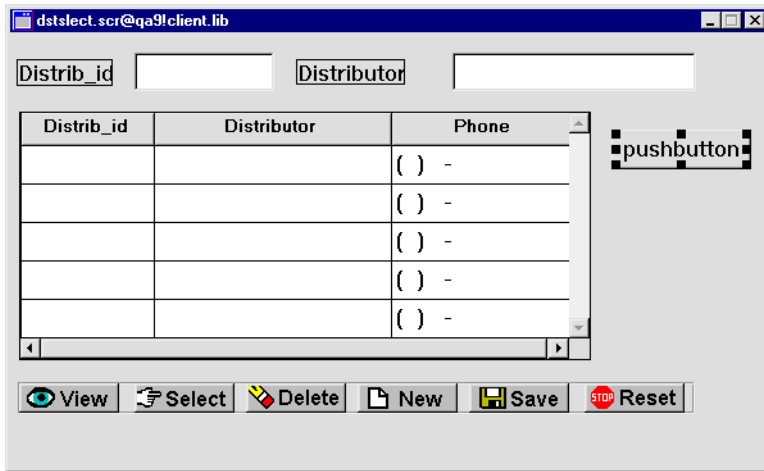
You can create widgets with either the Create menu or the Create toolbar. The next several steps show how to use the Create toolbar.

16 If the Create toolbar is not open:

- *Windows* Choose Options→Configure Toolbars. Select Create.
- *UNIX* Choose View→Tool Box



- 17 Choose the Push button tool on the Create toolbar.
- 18 Click near the right side of the dstselect.scr client screen.
A default-sized push button appears at the cursor position.



- 19 With the button selected, set its Name property to `order_pb`.

-
- 20** Set the Label property to Orders.

The button now has this label:



- 21** Set the Default/Cancel property to Default.

A push button defined as the Default button can be activated at runtime by pressing Enter.

- 22** Resize the Order button to match other buttons on the screen:

- Ctrl+click on one of the other buttons (such as the View button) whose size you want to match. This adds the button to the selection set as the selection set's dominant widget.
- With both buttons selected, choose Edit→Size→Adjust Both.

The Orders button resizes to the same height and width as the dominant selection.

Define push button behavior

Attach a control string to the push button so it performs the desired action. At run time, the control string executes when the Order button is clicked—in this case, it calls the `send_dst_data` procedure and is supplied an argument of `SELECT`.

- 23** Select the Orders push button.
- 24** In the Control String property (under Validation), enter
`^send_dst_data("SELECT")`.
- 25** Choose File→Save.

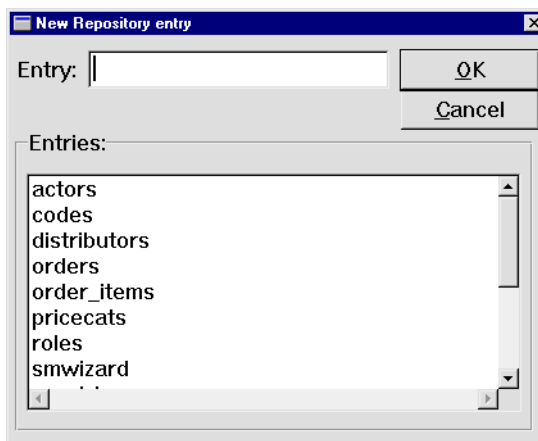
The `dstselect.scr` screen is saved to the remote `client.lib` library.

Create a buttons repository entry

You can copy commonly used objects from a screen to a repository, thereby providing the entire development team with application objects and code and facilitating consistent behavior and appearance across the application. In this case, you copy the Orders push button from the client screen to a repository entry. An inheritance link is automatically established between the client object and its parent in the repository.

- 26 Choose File→New→Repository Entry.

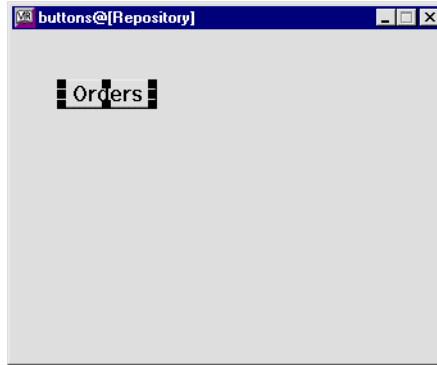
The New Repository Entry dialog box opens.



- 27 Enter buttons as the name of the entry. Choose OK.

An empty screen with the name `buttons@[Repository]` opens in the workspace.

- 28 Select the `order_pb` push button on the client screen `dstselect.scr` and drag it to the buttons repository entry.



- 29 Return to the client screen `dstselect.scr` and select its `order_pb` push button again. Check the `Inherit From` property (under `Identity`). It is now set to inherit its property values from the buttons repository entry (`buttons!order_pb`).
- 30 Give the buttons repository screen focus and choose `File→Save`.
The buttons repository screen is saved to the open repository.
- 31 (Optional) Bring focus to each of the screens and service components and choose `File→Close`.

What did you do?

You applied global and local changes to specific widgets, performing these tasks:

- Set properties on widgets in the repository and then propagated those changes to the widgets on client screens and service components that inherit from the repository.
- Selectively changed properties at the screen level to override inherited property values.
- Created a prototype push button and then copied it to a repository entry where it can be used and reused by all members of the development team.

What did you learn?

You learned:

- How to define input filters and control what the user sees and can do in the application interface.
- You can copy application objects to and from repository entries. In either case, the repository objects are the parents, and the child objects on the application screens inherit from them.
- How to write a control string to alter the default behavior of wizard-generated push buttons.



13 Writing and Executing JPL

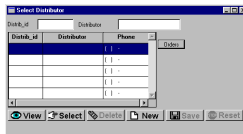
You can embed special processing in your application screens with Panther scripting language JPL. Because the JPL is saved with the screen or service component, it is accessible to the screen and its widgets.

In this lesson you learn how to:

- Attach JPL procedures to screens: one procedure to send data and the other to receive the transmitted data.
- Implement error handling by displaying a message to the user.
- Use the DB Interactions window to view database objects and their relationships on your application screens, and to gain access to their properties.
- Write transaction manager hook functions: one for the service component that automatically generates a distributor ID number when a distributor is added to the database, and one for the client screen that displays the generated ID.

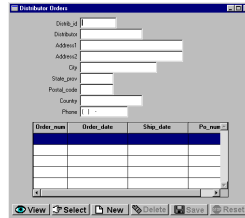
This lesson shows how to use the JPL editor and describes its capabilities. The JPL procedures that you need are already written and stored in `tutorial.lib`. The following diagram illustrates the JPL procedures you will enter in this lesson.

1. JPL to send data to dstord.scr



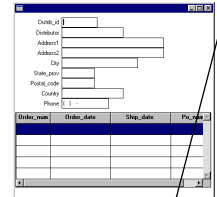
dstselect.scr

2. JPL to receive data



dstord.scr

3. JPL to generate ID

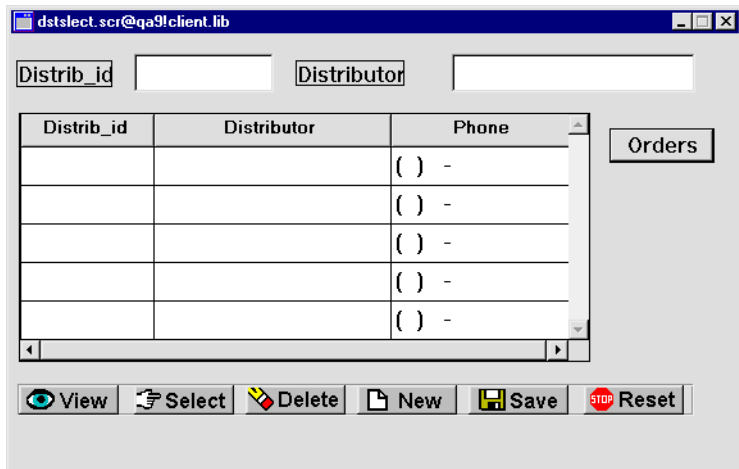


dstord.scr

4. JPL to unhide ID field

1 If necessary:

- Reactivate your application, invoke the editor, and connect to the middleware.
- Reopen client screen `dstselect.scr` from the remote `client.lib` library.



Write a procedure to access a distributor's orders

JPL to send data



dstselect.scr

Write a JPL procedure that is called when a user chooses the New or Orders button on the *dstselect.scr* client screen. Both buttons call a JPL procedure that opens the Distributor Orders screen; each button passes to this procedure a different parameter (NEW or SELECT), which determines whether the screen is used to add a new distributor to the database, or view orders of an existing distributor.

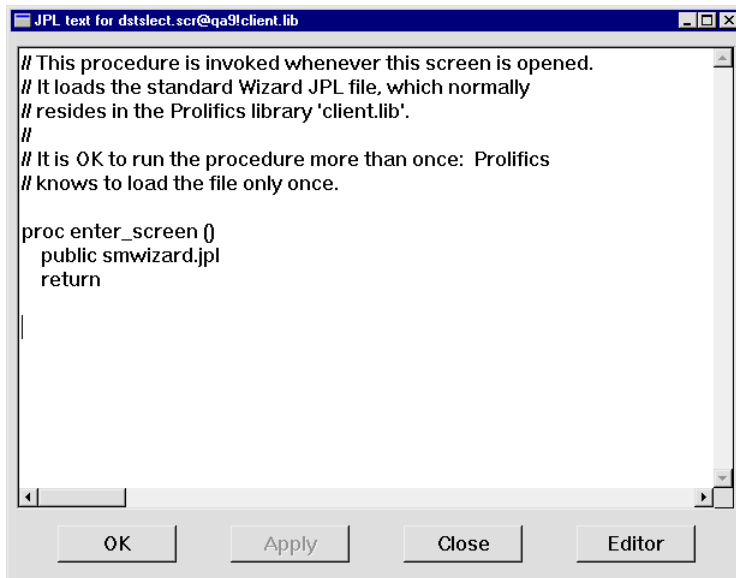
Because the same procedure is called by two different screen objects, make the procedure available to both by attaching the JPL code to the screen's JPL Procedures property.

- 2 Select the *dstselect.scr* client screen—click on an empty area in the screen to deselect all widgets.

The Properties window displays screen properties.

- 3 Under Focus, select the JPL Procedures property.

Panther opens the edit window for this screen's JPL:



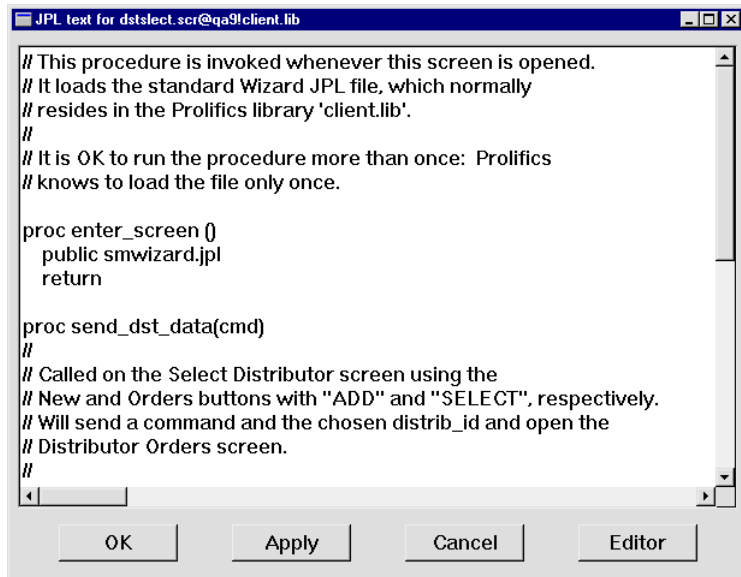
```
JPL text for dstselect.scr@qa9fclient.lib

// This procedure is invoked whenever this screen is opened.
// It loads the standard Wizard JPL file, which normally
// resides in the Prolifics library 'client.lib'.
//
//
// It is OK to run the procedure more than once: Prolifics
// knows to load the file only once.
//
//
proc enter_screen ()
  public smwizard.jpl
  return
//
//
```

The screen entry procedure *enter_screen*, produced by the screen wizard, is already attached to the screen's JPL. It calls the public command, which makes

the code in the `smwizard.jpl` module available to all application screens. Leave the wizard JPL code unchanged and add new procedures to this module.

- 4 Position the cursor a line or two below the return line of the `enter_screen` procedure. Insert the JPL procedure `send_dst_data`, which is stored in `tutorial.lib`:
 - Choose Edit→Insert From Library. The Open JPL Module dialog opens.
 - If necessary, open `tutorial.lib` by choosing Open and selecting it from the Open Library dialog.
 - From the Open JPL Module dialog, select `tutorial.lib` from the list of open libraries, and select `send_dst.jpl` from the list of members. Choose OK.



The screenshot shows a window titled "JPL text for dstselect.scr@qa9client.lib". The text inside the window is as follows:

```
# This procedure is invoked whenever this screen is opened.
# It loads the standard Wizard JPL file, which normally
# resides in the Prolifics library 'client.lib'.
#
# It is OK to run the procedure more than once: Prolifics
# knows to load the file only once.

proc enter_screen ()
  public smwizard.jpl
  return

proc send_dst_data(cmd)
#
# Called on the Select Distributor screen using the
# New and Orders buttons with "ADD" and "SELECT", respectively.
# Will send a command and the chosen distrib_id and open the
# Distributor Orders screen.
#
```

The `send_dst_data` procedure is read into the JPL edit window:

```
proc send_dst_data(cmd)
{
  vars occ
  if (cmd == "NEW")
  {
    send DATA cmd, ""
  }
  else
```

```

{
  occ= Master->grid_current_occ
  if (distrib_id[occ] == "")
  {
    msg emsg "First select a distributor."
    return 1
  }
  else
  {
    send DATA cmd, distrib_id[occ]
  }
}
call sm_jwindow("(+5,+5)dstord.scr")
return 0
}

```

More About the send_dst_data Procedure

The `send_dst_data` procedure is called when a user chooses either the New button or Orders button on the Select Distributor (`dstselect.scr`) screen. Each button supplies a different string argument to the procedure's `cmd` parameter, which controls how the procedure executes:

- If New is chosen, `NEW` is supplied as an argument. The `send` command puts this string into a specialized global buffer, or bundle. Bundle data is accessible to any other screen through the receive command.
- If the Orders button is chosen, `SELECT` is supplied as an argument. In this case, the `send` command puts two items into the bundle: the `SELECT` string, and the distributor ID in the selected distributor record. To ensure that a distributor ID is supplied, the procedure checks whether a record is selected from the grid, and issues an error message if one is not.

In both cases, the `dstord.scr` screen is opened through the call to `sm_jwindow`, which opens the Distributor Orders (`dstord.scr`) screen at the specified (+5, +5) position: five grid units right and five grid units down from the calling screen.

- 5 Choose OK to save the procedure and close the JPL editor.
- 6 Choose File→Save.
- 7 (Optional) Close the `dstselect.scr` client screen.

More About the JPL Edit Window

You can write and edit JPL procedures within Panther's own edit window. This window offers access to JPL that is attached to a screen or stored in a library.

You can also use any text editor that your system offers and that is specified in the application variable `SMEDITOR`—for example, Notepad on Windows or vi on UNIX. To access this editor:

- Bypass the JPL edit window and go directly to the system editor by choosing Options→Direct to External Editor.
- Invoke the system editor from the JPL edit window by choosing Edit→External Editor.

When you exit the system editor, you resume in the JPL edit window, where you save your changes back to its source by choosing File→Save. You can also save library JPL modules to a new library member by choosing File→Save As→Library Member.

To save the current module's contents to a disk file, choose File→Save As→ASCII Text File. Use this option in order to print JPL code from your system, or to send it in an email.

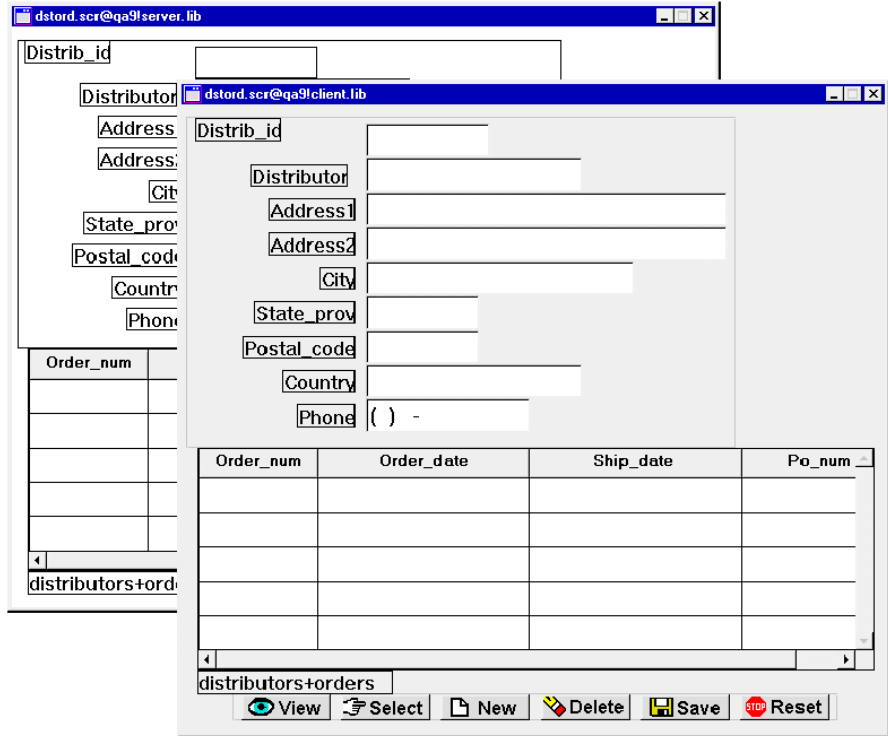
You can insert the contents of other JPL modules into the current one: choose Edit→Insert From Library to insert from a library module, or Edit→Read File to read from a disk file. Panther inserts the file's contents at the cursor's current position.

Write a procedure to receive data

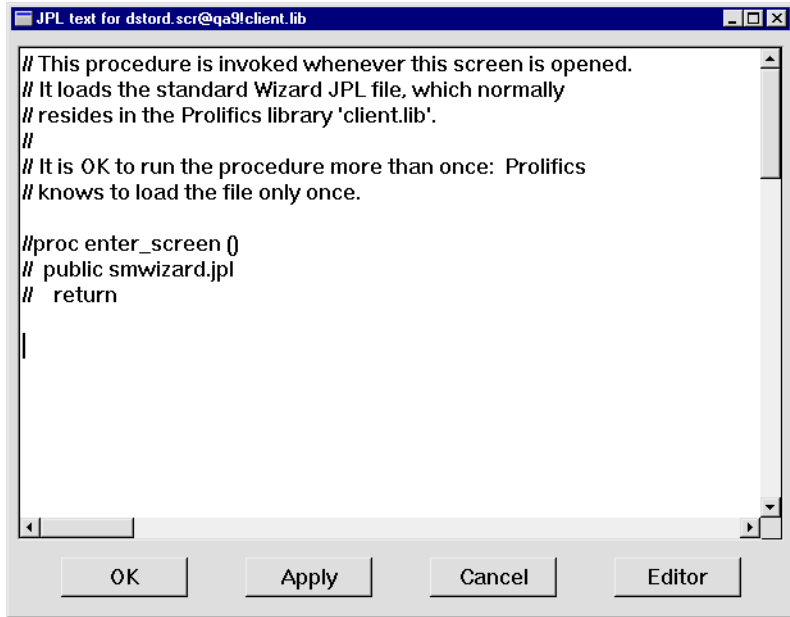
JPL to receive data The `send_dst_data` procedure sends data that is needed by the `dstord.scr` screen. Now you must write a screen entry procedure for `dstord.scr` that captures this data and uses it to determine the screen's behavior when it is opened by the `dstselect.scr` screen's New or Orders buttons.



- 8 If the `dstord.scr` client screen and service component are not open, choose View→Library TOC and open them from the `client.lib` and `server.lib` libraries, respectively.



- 9 Select the `dstord.scr` client screen by clicking in an empty space within its borders, so no widgets are selected.
- 10 Under Focus, choose JPL Procedures.
The JPL edit window opens and displays screen wizard-generated comments and code. Replace this with code that is in `tutorial.lib`.
- 11 Comment out the original `enter_screen` procedure by placing `//` at the beginning of each of the three lines.



The screenshot shows a window titled "JPL text for dstord.scr@qa9fclient.lib". The window contains the following text:

```
// This procedure is invoked whenever this screen is opened.  
// It loads the standard Wizard JPL file, which normally  
// resides in the Prolifics library 'client.lib'.  
//  
// It is OK to run the procedure more than once: Prolifics  
// knows to load the file only once.  
  
//proc enter_screen ()  
// public smwizard.jpl  
// return  
  
|
```

At the bottom of the window, there are four buttons: "OK", "Apply", "Cancel", and "Editor".

- 12 Position the cursor a line or two below the return line of the commented procedure.
- 13 Choose Edit→Insert From Library.
- 14 From the Open JPL Module dialog, select `se_dst.jpl` from `tutorial.lib` and insert the file by double-clicking on its name or choosing OK.

The screenshot shows a window titled "JPL text for dstord.scr@qa9!client.lib". The text inside the window is as follows:

```

// This procedure is invoked whenever this screen is opened.
// It loads the standard Wizard JPL file, which normally
// resides in the Prolifics library 'client.lib'.
//
// It is OK to run the procedure more than once: Prolifics
// knows to load the file only once.

//proc enter_screen ()
// public smwizard.jpl
// return

// This procedure is called whenever this screen opens.

proc enter_screen (screen_name, context)
{
    // On screen entry, load the standard Wizard JPL file,
    // which normally resides in client.lib.
    // The Wizard JPL is loaded only once, no matter

```

At the bottom of the window, there are four buttons: "OK", "Apply", "Cancel", and "Editor".

The enter_screen procedure is read into the JPL edit window:

```

proc enter_screen (screen_name, context)
{
    public smwizard.jpl
    vars cmd

    if ((context & K_EXPOSE) || !(sm_is_bundle("")))
    {
        return 0
    }
    else
    {
        receive DATA cmd, distrib_id
        if (cmd == "NEW")
        {
            LDistrib_id->hidden=PV_YES
            distrib_id->hidden=PV_YES
        }
        call sm_tm_command(cmd)
        return 0
    }
}

```

The `enter_screen` procedure is invoked as the `dstord.scr` screen's entry procedure. First, the public command makes the wizard-generated JPL module `smwizard.jpl` available to the application. Next, the procedure determines whether a user can add a distributor record or edit an existing one:

- The first if statement checks for two conditions: the screen is already open and is reexposed—`(context & K_EXPOSE)`; or no data was sent from the `dstselect.scr` screen—`!(sm_is_bundle(" "))`. If either condition is true, the procedure returns.
- If both if conditions are false, the `receive` command captures the bundle data sent from the `dstselect.scr` screen. If data is received for `distrib_id`, it is placed in that widget.
- An if statement tests the value of `cmd`. If it evaluates to the string `NEW`, text widget `distrib_id` and its label `LDistrib_id` are hidden. `sm_tm_command` passes the value of `cmd` to call the appropriate transaction manager command, `NEW` or `SELECT`. If the `NEW` command is called, it prepares the screen to accept new data. If the `SELECT` command is called, it fetches the appropriate data using the value in `distrib_id`.

15 Choose Apply.

16 Save the client screen (choose File→Save).

More About Sharing Information Across Screens

The `send` and `receive` commands let you transfer data between screens in an application. The `send` command specifies the data items to send and stores them in a buffer. When the target screen executes the corresponding `receive` command, the buffered data is retrieved.

Panther provides several ways for screens to access each other's data:

- JPL can directly reference variables and widgets on other open screens with the syntax convention `screen_name!widget_name`. However, this syntax is invalid for referencing between client screens and service components.
- JPL global variables, created with the `global` command, can store data that is accessible to the entire application.
- Local Database Blocks (LDB) contain widgets whose values are automatically written to same-named widgets on a screen when it opens or regains focus, and which read the values of those widgets when the screen exits.

For more information about using `send` and `receive` commands and LDBs, [CLICK HERE](#).

Generate a unique ID number

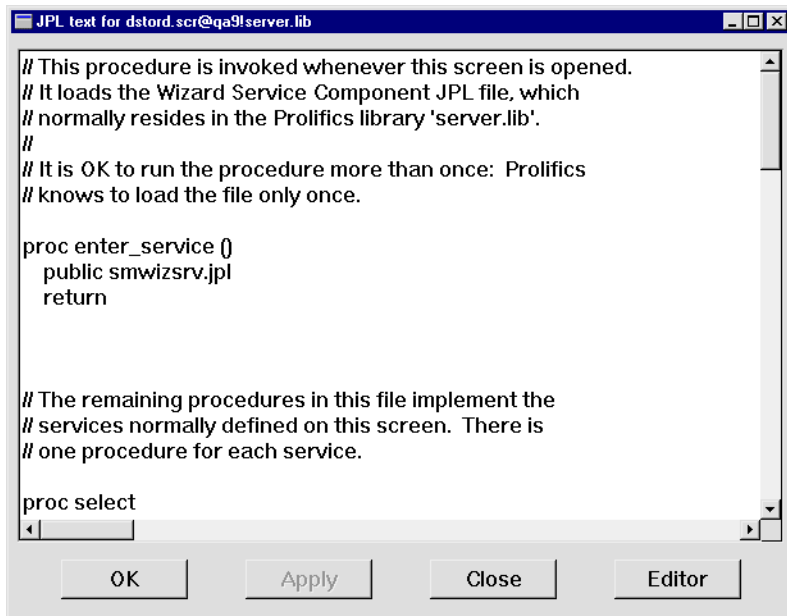
JPL for unique ID



dstord.scr

The JPL that you have so far written opens the `dstord.scr` screen when a user chooses the New button on the `dstselect.scr` (Select Distributor) screen, and opens `dstord.scr` (Distributor Orders) in insert mode. You can now write a JPL hook function for the `dstord.scr` service component that generates a unique distributor ID number. This hook function executes during transaction manager processing and supplements the default transaction model.

- 17 Select the `dstord.scr` service component by clicking in an empty space within its borders (no widgets should be selected).
- 18 Under Focus, choose JPL Procedures.
- 19 Position the cursor after the `enter_service` procedure and before the comments to the select procedure.



- 20 Choose Edit→Insert From Library.

-
- 21** From the Open JPL Module dialog, select `evnt_svr.jpl` from `tutorial.lib` and choose OK.

The `tm_events_svr` procedure is read into the JPL edit window:

```
proc tm_events_svr (event_id)
{
  if ((event_id == TM_INSERT_EXEC) && (distrib_id == ""))
  {
    DBMS ALIAS distrib_id
    DBMS QUERY SELECT MAX(distrib_id)+1 from distributors
    DBMS ALIAS // Clears prior aliasing
  }
  return TM_PROCEED
}
```

The `tm_events_svr` procedure checks whether an Insert transaction manager event occurred and the distributor ID is blank. If both conditions are true, the first `DBMS ALIAS` command ensures that the database value from the `DBMS QUERY` statement is written to the variable `distrib_id`.

To generate a unique identification number, the second `DBMS ALIAS` statement specifies a `SELECT` to find the highest distributor ID in the distributors database table and increments that number by one. The service component returns that number to the client.

- 22** Choose Apply.

More About `DBMS QUERY` and `DBMS RUN` Statements

In general, you'll use the transaction manager to automatically generate SQL statements. However, you can also write your own SQL. The Panther `DBMS` statements let you:

- Declare open connections.
- Manage cursors explicitly.
- Execute SQL statements, stored procedures, RPC calls.
- Execute a variety of database-related tasks.

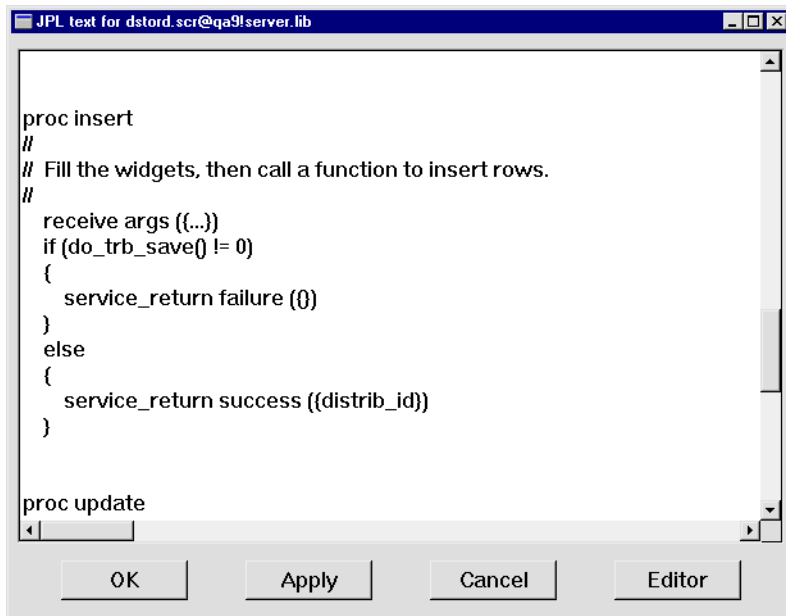
Refer to the *Application Development Guide* for more information.

Insert the ID in the Database

When a user adds the necessary distributor information and chooses the Save button on the Distributor Orders screen, the record must be inserted into the database. Here you modify the insert JPL procedure on the `dstord.scr` service component to explicitly add the `distrib_id` into the distributors database table.

- 23 In the JPL edit window, scroll down to the insert procedure. Find the statement with the `service_return` command.
- 24 Insert `distrib_id` within the braces of the `service_return` command:

```
service_return success ({distrib_id})
```



- 25 Save your changes: choose OK.
- 26 Save the service component.

Invoke the hook function on the server

When you used the screen wizard to create the `dstord.scr` service component and client screen in Lesson 7, you specified two tables as their data source. When the wizard created these screens, it copied an invisible table view widget from each table's corresponding repository screen, distributors and orders. Each table view widget contains information about its repository screen's source table; it is created automatically when you import the table into the repository (Lesson 6).

Also, because `dstord.scr` is based on two tables, it also contains a link widget that specifies their join relationship. Like table widgets, link widgets are automatically created during the import process for each foreign key that is defined for a database table. When the screen wizard creates a screen with multiple table views, it copies from the repository the links that describe their relationship.

More About Table Views

Table views store the following types of database information:

- Primary key
- List of table columns
- Database attributes such as, sort order, distinct, etc.
- Service specifications

Panther's transaction manager uses the information stored in table view and link widgets to determine the SQL statements to generate for each transaction command. The JetNet transaction model uses the service information to determine how to respond to service calls.

Also, table views let you easily modify the default transaction manager behavior.

At runtime, the transaction manager traverses all screen table views in the order that they are linked, and issues transaction commands to populate the master and detail sections accordingly. If a table view has a hook function attached to it, the transaction manager executes the function when it traverses that table view.

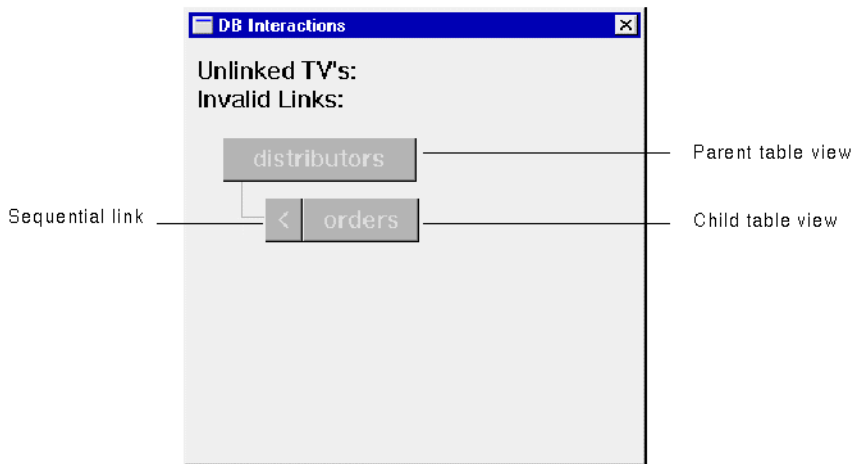
You attach a hook function to a table view through its Function property. In this case, you want to attach a hook function to the distributors table view. You can select a table widget and view its properties through either the [Widget List](#) or the DB Interactions window. In this instance, use the DB Interactions window to select the table view widget and access its Function property.

More About the Database Interactions Window

The Database Interactions window displays an interactive, graphical representation of a screen's table view widgets and link widgets. You can view the relationships between parent and child table views and the links that connect them. By selecting the toggle buttons representing these database objects you gain access to their properties.

- 27** With focus on the `dstord.scr` service component, choose View→DB Interactions.

The DB Interactions window opens, displaying a graphical representation of table views and links on the `dstord.scr` service component.



More About Sequential and Server Links

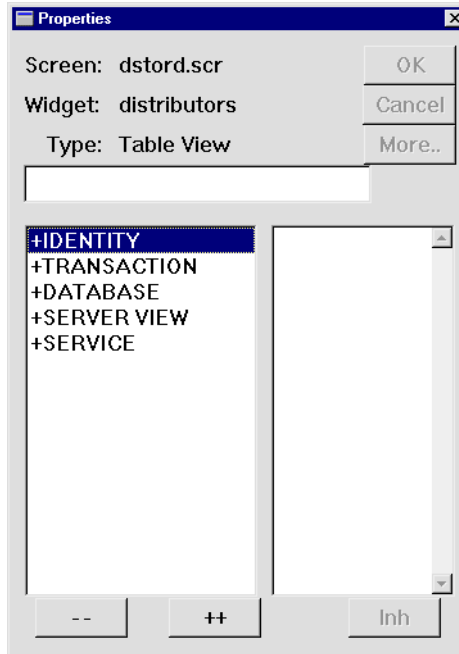
Link widgets are not visible at runtime, but are visible in the editor so that you can access their properties.

There are two type of links—sequential and server:

- Sequential links (denoted by an arrow `<` in the DB Interaction window) join two tables with a one-to-many relationship. SQL SELECT statements for the parent table view are generated and executed before any SQL statements are generated for the child table view.
- Server links (denoted by an equal sign `=` in the DB Interaction window) join two tables with a one-to-one relationship. The database server is used to join the two tables, and a single SQL SELECT statement is generated to retrieve the data.

- 28** In the DB Interactions window, select the button that represents the distributors table view.

The Properties window now displays table view properties.



- 29** Under Transaction, set the Function property to `tm_events_svr`.
- 30** Save all the open screens and service components (press F6).

Write a hook function for the client event

JPL to unhide ID



dstord.scr

So far, the service component `dstord.scr` determines whether to generate an ID, and saves it to the database if a new one is indicated. But the corresponding client screen is unaware of the new ID. Because the distributor ID field and its corresponding label are hidden when a new record is added, you can programmatically unhide these fields and display the new ID by changing their Hidden property when the record is saved.

- 31** Give focus to the JPL edit window of the `dstord.scr` client screen.

-
- 32 Scroll to the end of the `enter_screen` screen entry procedure and press Enter twice to insert blank lines between procedures.
 - 33 Choose Edit→Insert From Library.
 - 34 From the Open JPL Module dialog, select `evnt_clt.jpl` in the `tutorial.lib` and choose OK.

The `tm_events_clt` procedure is read into the JPL edit window:

```
proc tm_events_clt (event_num)
{
  if (event_num == TM_POST_SAVE)
  {
    distrib_id->hidden=PV_NO
    LDistrib_id->hidden=PV_NO
  }
  return TM_PROCEED
}
```

The `tm_events_clt` hook function checks whether a transaction manager Save command executed and, if so, unhides widgets `distrib_id` and `LDistrib_id` by changing their Hidden property to No.

- 35 Save the procedure: choose OK.

More About Accessing Properties

All Panther objects and their properties can be accessed and most can be modified programmatically through JPL or C functions. You get or set properties for any screen or widget, or the application itself.

Refer to “Setting Properties Using the Property API” in Chapter 19 of *Application Development Guide* for information about JPL syntax for identifying screens and widgets, their properties, and property values. For a list of properties and their values, refer to Chapter 1, “Runtime Properties,” in *Quick Reference*.

Invoke the hook function for the client event

Now that the function `tm_event_clt` has been added to the `dstord.scr` client screen, this hook function needs to be attached to the screen's table view widget via its Function property. At runtime, the transaction manager uses this function to display the results to the user, confirming visually that a new distributor was added to the database.

-
- 36** Using the DB Interactions window, select the button representing the distributors table view on the `dstord.scr` client screen.




- 37** Under Transaction, enter `tm_events_clt` in the Function property.
If a Save command executes and the new distributor record is committed to the database, the new distributor ID displays.
- 38** Close the DB Interactions window: use its system menu or choose View→DB Interactions to toggle the display.
- 39** Save the changes in the `dstord.scr` client screen.


Add a new database record

You now have a working application. You can now test whether the application flow functions as designed.

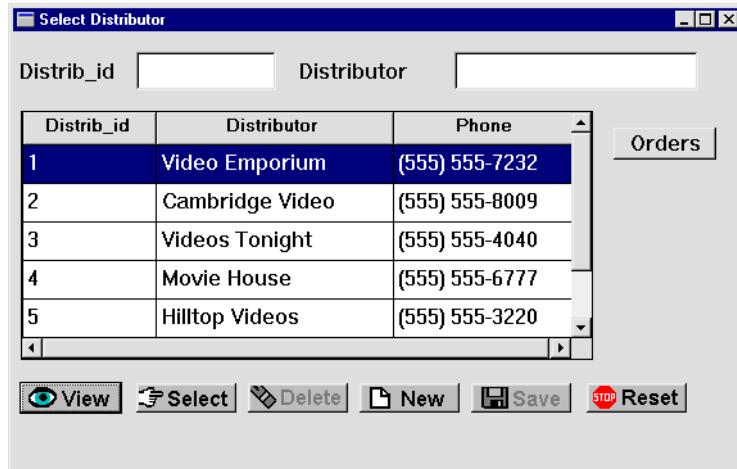
- 40** Bring focus to the `dstselect.scr` client screen.


- 41** Choose File→Test Mode or  .

The Select Distributor screen opens.

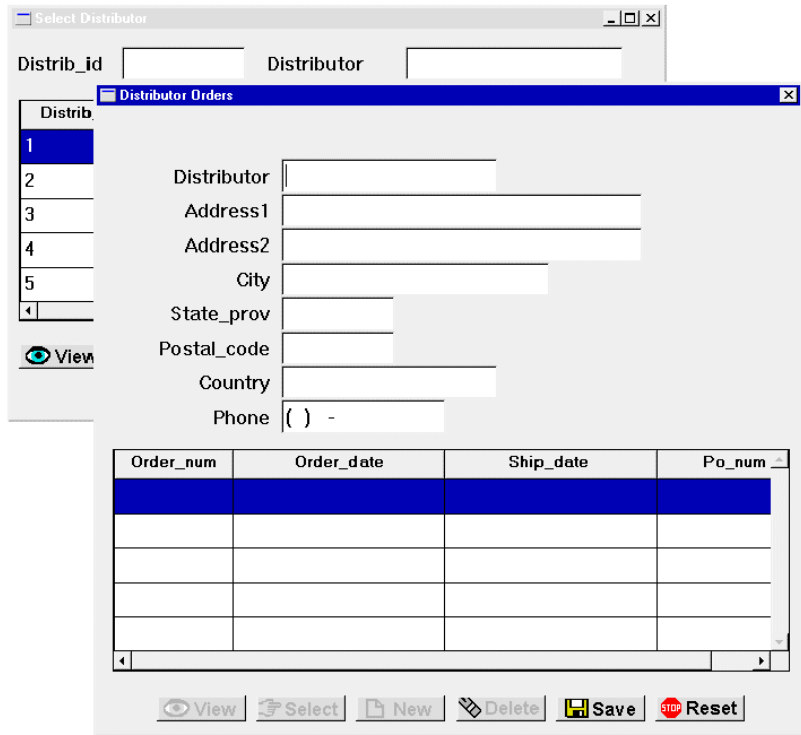
42 Choose  .

All distributors in the vidsales database are listed.




43 Choose  .

The Distributor Orders screen opens with all data fields clear. The distributor ID fields are not visible.



44 Enter Movie Time in the Distributor field.

45 Choose  .

The distributor is added to the database and the ID is displayed in the Distrib_id field.

Distributor Orders

Distrib_id 8

Distributor Movie Time

Address1

Address2

City

State_prov

Postal_code

Country

Phone () -

Order_num	Order_date	Ship_date	Po_num

View Select New Delete Save Reset

- 46** Close the Distributor Orders screen (choose Close from the system menu).

The Select Distributor screen regains focus. If you scroll down the list of distributors, notice that the new distributor is not listed. The next lesson shows how to enhance the screen entry procedure on this screen so users can see the latest data changes.

View orders

Select one of the distributors and then gain access to all of their orders.

- 47** Select the row in the grid associated with `Distrib_id 3` and choose the Orders button.

The Distributor Orders screen opens and displays all orders for the selected distributor. The screen entry procedure (`enter_screen`) on the `dstord.scr` client screen receives the ID you selected (3) and the `SELECT` command, and calls `sm_tm_command` to select the specified database record.

48 When done, return to the editor and save any unsaved screens.

What did you do?

You inserted several screen-level JPL procedures: a pair of procedures that send data from one screen and capture it to another; and two other JPL procedures that tell the transaction manager how to handle database transactions. You did this by writing these procedures:

- A procedure that submits a specific command and the appropriate data about a selected distributor to another screen. You attached the procedure to the `dstselect.scr` client screen so that it is called whenever a user chooses either the Orders or New button. These buttons invoke this procedure via their Control String property, passing as an argument the command to issue. If a `NEW` command is issued, the `dstord.scr` client screen opens and is ready to accept new data. If a `SELECT` command is issued, primary key data for the selected distributor is submitted to the `dstord.scr` client screen.
- A screen entry procedure for the called screen—the `dstord.scr` client screen—that receives a `NEW` or `SELECT` command and accordingly determines the screen's behavior.
- A transaction manager hook function in JPL for the `dstord.scr` service component that automatically generates a unique ID when a new distributor record is added to the database.
- A transaction manager hook function in JPL for the `dstord.scr` client screen that displays the new distributor ID to the user.

What did you learn?

You learned:

- The JPL edit window provides several ways to attach JPL to screens—inserting a JPL module/procedure from another library, inserting a file from a disk, and typing it directly in the edit window. You can also use your favorite text editor.

-
- Depending on the application, there are advantages to storing all the JPL procedures at screen-level where they are available to the entire screen and all widgets on it. The procedures can be called from a widget property such as Control String or Entry/Exit properties.
 - You can control application behavior and database transactions with JPL procedures.
 - You can customize transaction manager behavior with hook functions that the transaction manager calls while it processes database transactions.
 - The DB Interactions window offers a graphical representation of a screen's table views and links, and access to their properties.



14 Customizing Screen Behavior

After an application's basic functionality is in place, you typically continue to work on it to fine tune its behavior and usability. This lesson shows how to:

- Define a hook that handles double-click events. In this lesson, double-clicking in a grid row calls a procedure that opens a screen and sends the row data to it.
- Write a JPL procedure that executes a database query conditionally.

1 If necessary:

- Reactivate your application, invoke the editor, and connect to the middleware.
- Reopen the `dstselect.scr` and `dstord.scr` client screens from `client.lib` on the application server.

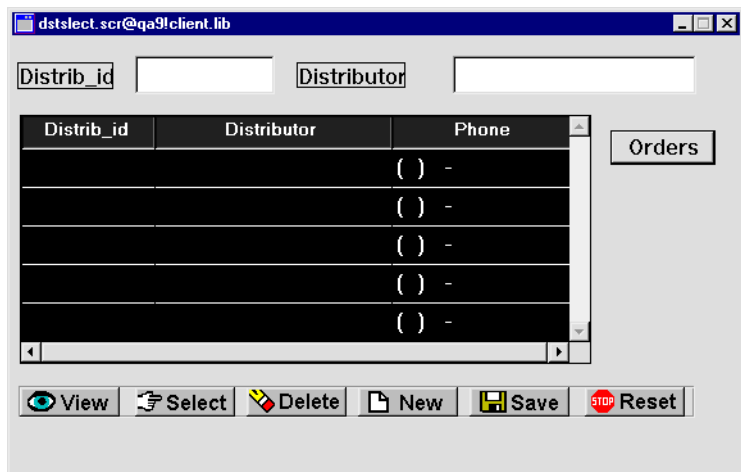
Add double-click functionality

A widget's Double-click property can be set to a control string that determines what happens when users double-click on that widget. In this lesson, you edit the client screen `dstselect.scr` to control what happens when a user double-clicks on a distributor record in the grid widget. The grid widget columns' Double-click property is set to call

the `send_dst_data` procedure, which displays the selected distributor's record for editing. Thus, double-clicking on any grid widget field emulates the behavior of the Orders button (described in lessons 12 and 13).

2 Bring focus to the `dstslect.scr` client screen and in the grid widget, select widgets `distrib_id`, `distrib_name`, and `phone` by selecting grid columns `Distrib_id`, `Distributors`, and `Phone` columns:

- *Windows* Shift+click on the column titles.
- *Motif* Shift+click directly within a cell in each column.



3 Under Validation, set these widgets' Double Click property to `^send_dst_data("SELECT")`.

When the user double-clicks in any of the columns in the grid widget, the `send_dst_data` procedure executes (as it also does when the Orders push button is chosen) and sends the selected ID to the `dstord.scr` client screen.

4 Bring focus to the `dstord.scr` client screen and select all the columns in the grid widget.

5 Under Validation, set the Double Click property to `^send_order_data()`.

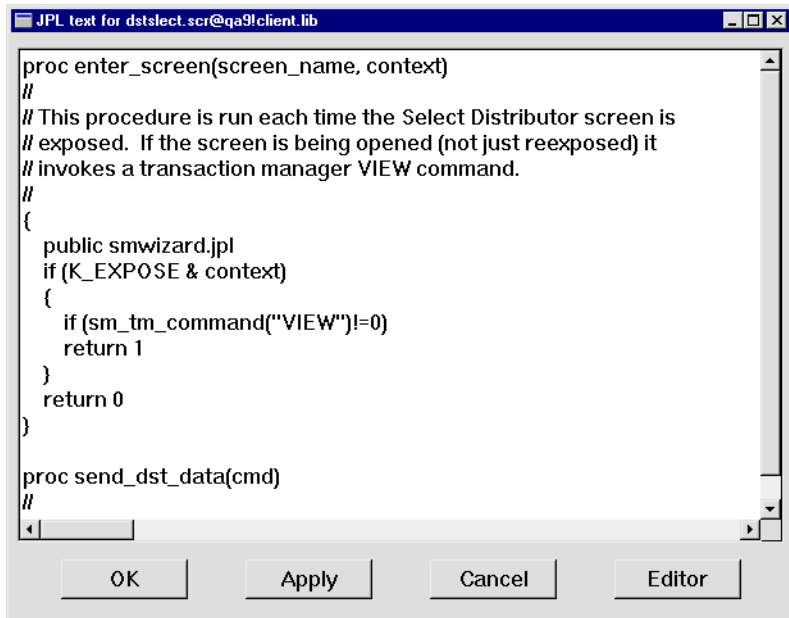
When the user double-clicks on a specific order, the `send_order_data` procedure, which is associated with the `dstord.scr` screen, is called. This procedure is described later ([page 16-20](#)).

6 Save all open screens (press F6).

Write a screen entry function that executes only on screen exposure

When you edit an existing distributor's data or add a distributor on the `dstord.scr` (Distributor Orders) screen, it is not immediately visible when you return to the `dstselect.scr` (Select Distributor) screen. You can enhance the screen entry procedure on the `dstselect.scr` client screen so that when the screen redisplay (after the Distributor Orders screen closes), a View command automatically executes. This causes the updated database records to redisplay.

- 7 If necessary, reopen the JPL edit window for the `dstselect.scr` client screen:
 - Select the `dstselect.scr` client screen (deselect all widgets).
 - Under Focus, select the JPL Procedures property.
- 8 In the JPL edit window, delete the screen wizard screen entry procedure—everything up to the `send_dst_data` procedure.
- 9 Get the new screen entry procedure from the tutorial library:
 - Choose Edit→Insert From Library. The Open JPL Module dialog opens.
 - If necessary, open `tutorial.lib` by choosing Open and selecting it from the Open Library dialog.
 - From the Open JPL Module dialog, select `tutorial.lib` and select `se_select.jpl` from its list of members. Choose OK.

A screenshot of a JPL text editor window titled "JPL text for dstslect.scr@qa9!client.lib". The window contains the following code:

```
proc enter_screen(screen_name, context)
//
// This procedure is run each time the Select Distributor screen is
// exposed. If the screen is being opened (not just reexposed) it
// invokes a transaction manager VIEW command.
//
{
  public smwizard.jpl
  if (K_EXPOSE & context)
  {
    if (sm_tm_command("VIEW")!=0)
      return 1
  }
  return 0
}

proc send_dst_data(cmd)
//
```

The window has a scroll bar on the right and a status bar at the bottom with buttons for "OK", "Apply", "Cancel", and "Editor".

The `enter_screen` procedure for the `dstslect.scr` client screen is read into the JPL Program text window.

```
proc enter_screen (screen_name, context)
{
  public smwizard.jpl
  if (K_EXPOSE & context)
  {
    if (sm_tm_command("VIEW")!=0)
      return 1
  }
  return 0
}
```


The expression in the `if` statement tests the `K_EXPOSE` bit in the `context` argument. If the expression evaluates to true (the screen was reexposed), the `View` command executes and the updated database records are displayed. Otherwise, the procedure returns without performing any actions.

- 10 Save the procedure: choose `File`→`Save`.

Test the JPL

Now when you add or edit a distributor record, those changes should display immediately after you return to the Select Distributor screen.

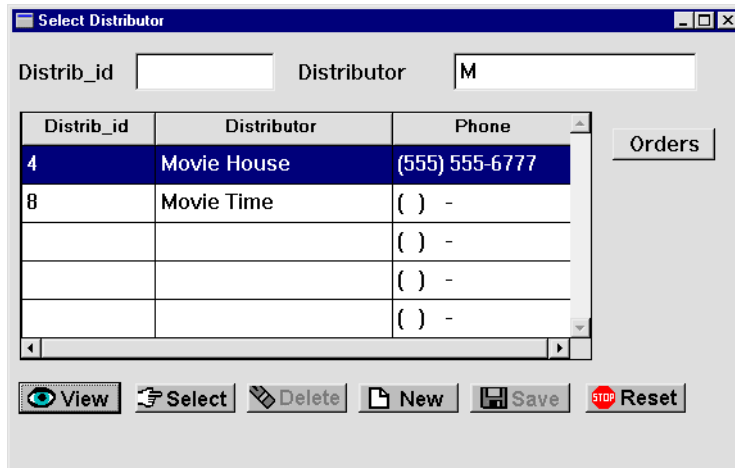
11 Bring focus to the dstslect.scr client screen.

12 Choose File→Test Mode or .

The Select Distributor screen opens.

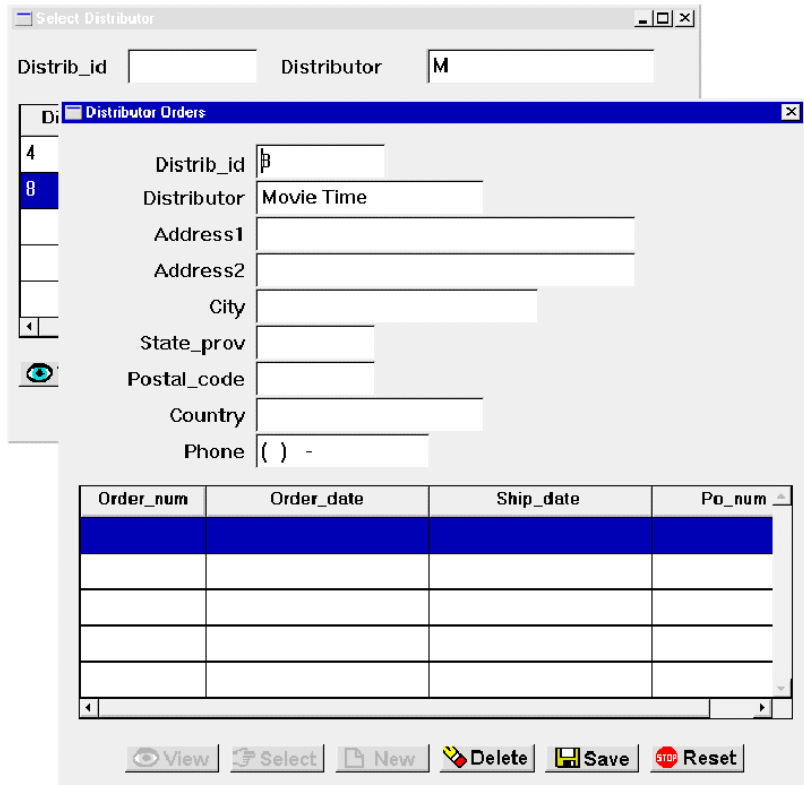
13 Enter M in the Distributor field and choose .

All distributors having an uppercase M in their name are displayed, including the one you added in Lesson 13.



14 Double-click anywhere on the Movie Time row.

The Distributor Orders screen opens, overlaying the Select Distributor screen.



- 15 Enter (555) 345-6000 in the Phone field and choose Save.
- 16 Close the Distributor Orders screen (choose Close from the system menu).
The Select Distributor screen regains focus. The phone number shows the new data that you just saved to the vidsales database.
- 17 Return to the editor when you're done. Remember to save your screens (File→Save All).

What did you do?

You enhanced the user interface by performing these tasks:

- Implemented double-click events. Now in addition to choosing push buttons, a user can invoke the `send_dst_data` procedure by double-clicking on a specific distributor record on the Select Distributor screen. This action invokes the procedure and sends the selected ID to the Distributor Orders screen.
- Wrote a screen entry procedure for the `dstselect.scr` client screen that executes only when the screen is reexposed.

What did you learn?

You learned:

- Double-click events are a useful enhancement to the user interface.
- Screen entry procedures can force a database transaction command to be issued without requiring any input from the user.



Module 4— Extending the Application



15 Implementing Selection Screens

In Lesson 7, you created a master-detail screen that joined two tables: `distributors` and `orders`. In this lesson, you create another master-detail screen that joins the `orders` table as the primary master table to the `order_items` table. In addition to columns from the `order_items` table, the screen's detail section also includes data from the `titles` table. Thus, it can display the titles of the videos.

Because the screen's detail section contains multiple tables, the screen wizard:

- Provides the link widgets that define the relationships between all tables on your screen.
- Assigns the name (in the Validation Service property) of the link operation to the link widget. The service validates new or changed data. So, for the screen you create in this lesson, when a user adds a new order item ID, the information returned from the database is, in fact, valid.
- Lets you generate a selection screen where users can pick from valid choices, and a corresponding selection service component. In this lesson, the `titles.cit` selection screen and selection service component are created.
- Assigns the name of the Select service on the master table view of the selection screen. This service is called by the selection screen to populate the selection screen with valid options from which a user can choose. For this lesson, it lets you add an order item to an order by selecting from a list of video titles (on the selection screen).

In this lesson you learn how to:

-
- Tell the screen wizard to include columns from an additional table on the screen, and build a selection screen and its corresponding selection service component.
 - Test the behavior of selection screens when adding new records to the database.
-


- 1 If necessary:
 - Restart the application now, invoke the editor, and connect to the middleware.
 - Reopen the repository, data.dic. (On Windows, File→Open→Remote Repository. On UNIX, File→Open→Repository.)

Join multiple tables

When you use the screen wizard, the first table you select for a section—master, detail, or subdetail—is considered the first (primary) table view for that section. For the screen you create here, select orders as the first master table and join it to `order_items` as the first detail table

- 2 Choose File→New→Screen or .

The New Screen Tool dialog opens.

- 3 Choose Yes to use the screen wizard.
The Format Selection dialog opens.
- 4 Select the Master-Detail option (the default) to define the sections.
- 5 Deselect the Web Friendly Output option.
- 6 Choose Next. The First Master dialog opens.
- 7 Select orders from the list of Tables To Pick From.
- 8 Choose .

This specifies to include all columns in the orders table on the completed screen.

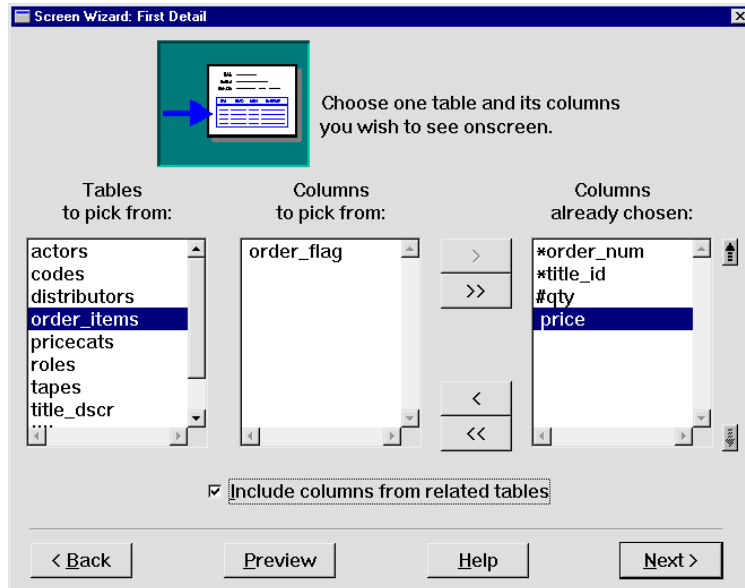
- 9 Choose Next. The First Detail dialog opens.
- 10 Select `order_items` from the list of tables.
- 11 Double-click on price.

The selected column is added to the list of those already chosen.

Add details from another table

The screen wizard lets you include information from other database tables in the same section, as long as the corresponding repository screens specify links to the section's first table. By selecting the titles tables in addition to the `order_items` table, the screen's detail section can display the name, number of available copies, and standard unit price of each video, along with the price and quantity data from `order_items`. The screen wizard includes links that define the relationship between `order_items` and titles.

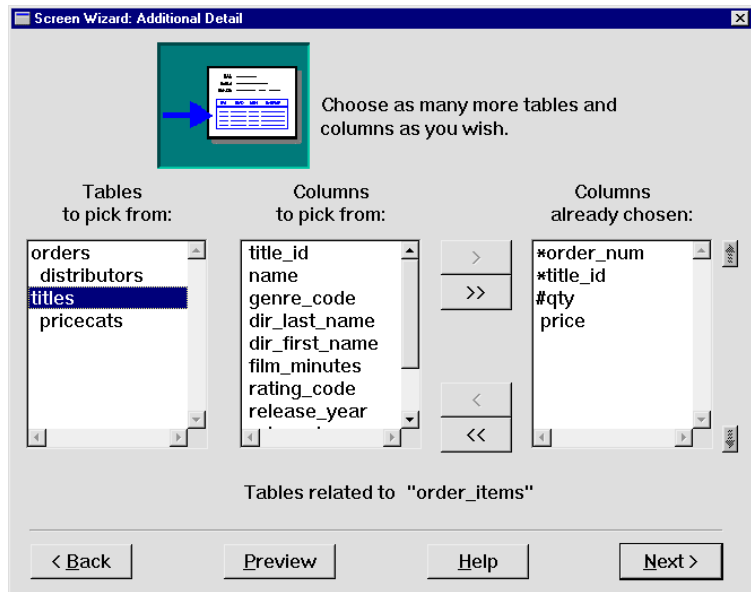
- 12 Select the Include Columns from Related Tables check box.



13 Choose Next.

The Additional Detail dialog opens and shows which tables can be joined to the `order_items` table.

14 Select titles from the list of tables.

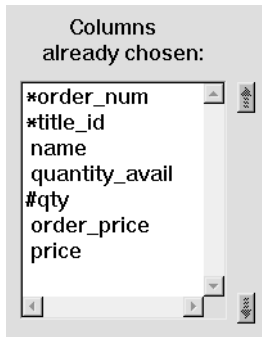


The list of columns belonging to the titles table displays.

15 Select (Ctrl+click) name, order_price, and quantity_avail.

16 Choose .

17 Reorder the columns (use the up/down position arrows) as shown:



- 18** Choose Next.

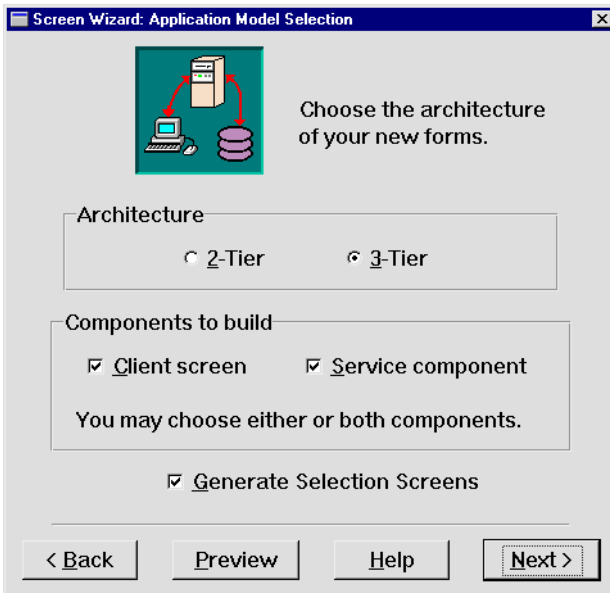
The Layout Selection dialog opens.

- 19** Choose Next to accept the default layout specification: single row for the master and grid display for the detail.

The Application Model dialog opens.

Generate selection screens

When you include columns from additional tables, the screen wizard lets you decide whether to generate selection screens (and selection service components). The usefulness of selection screens depends on the client screen's function. For example, a data entry screen might make good use of a selection screen, while a display-only client screen probably would not.



More About Selection Screens

When the Generate Selection Screens check box is selected, the screen wizard automatically creates a selection screen for every additional table that you include on your client screen. At the same time, the screen wizard also creates selection screen service components (for three-tier models). As a result, additional services are required to carry out the appropriate requests.

Selection screens, sometimes called pick lists, are useful when a user is adding a new record to the database. The selection screen displays a list of acceptable values for a field when the user requests help.

- 20** Choose Next to accept the default architecture (three-tier) and components (client screen and service component, and selection screens). The Generate Selection Screens check box is selected by default.

The Service Definition dialog opens.

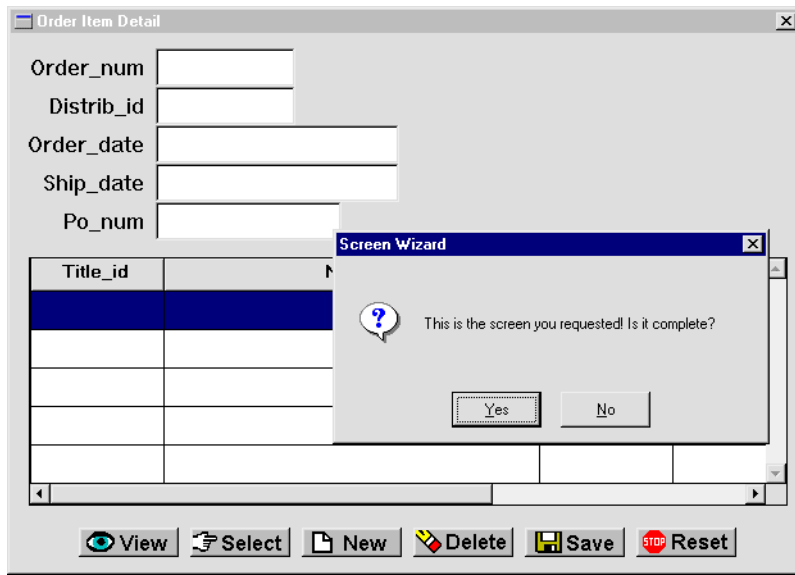
- 21** Change the Service Prefix to `orditm` and press TAB.

- 22** Choose Next.

The Style and Finish dialog opens.

- 23** Change the screen title to Order Item Detail and choose Done.

A preview of the client screen displays:

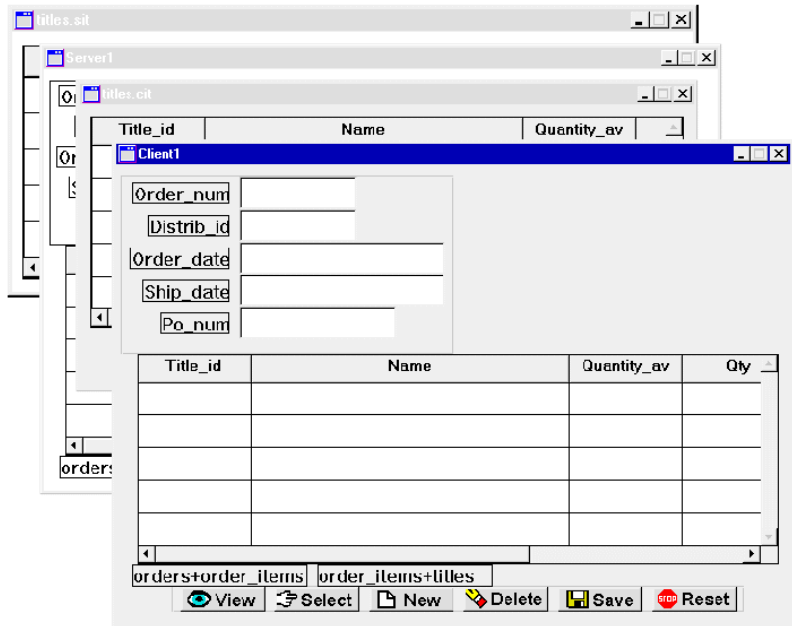


24 Choose Yes to confirm the contents of the screen.

When the wizard finishes building the screens (notice the status bar), the results include four screens:

- Client screen
- Corresponding service component
- Selection screen for the titles table `titles.cit`
- Corresponding selection service component `titles.sit` for the server

The screens are stacked, so you need to move or minimize the top screen to see the one beneath it.



Save the wizard output

Save the client screens to `client.lib` and the server screens to `server.lib`.

25 Bring focus to the client screen (Client1) and choose Save.

The Save Screen dialog opens.

26 Save the screen as `orditm.scr` in remote library `client.lib`.

27 Repeat steps 25 and 26 for the service component (Server1), saving it to remote library `server.lib`. You can close the service component after saving it.

28 Save the other screens to their appropriate remote libraries: `titles.cit` to `client.lib` and `titles.sit` to `server.lib`. You can close these screens also after saving them.

Define link and validation services

In addition to defining services to handle client requests made on the `orditm.scr` screen, you need to define the service that validates the data returned from the `titles` table, and a service that populates the `titles.cit` selection screen. Define all the services in the JIF.

- 29 Invoke the JIF editor and connect to the middleware (refer to Lesson 8 for details on using the JIF editor).
- 30 Create the following services (press TAB after entering the service name, and accept or set the appropriate values):

Service Name	Routine Name	Service Component
<code>orditm_d</code>	<code>delete</code>	<code>orditm</code>
<code>orditm_i</code>	<code>insert</code>	<code>orditm</code>
<code>orditm_s</code>	<code>select</code>	<code>orditm</code>
<code>orditm_u</code>	<code>update</code>	<code>orditm</code>
<code>orditm_ll</code> (lowercase L + number one)	<code>val_link</code>	<code>orditm</code>
<code>titles_c</code>	<code>choose</code>	<code>titles.sit</code> (include the extension)


- 31 Save the JIF back to the remote `common.lib`, exit from the JIF editor, and release the reservation on the JIF.

The application knows (as long as you are connected to the middleware) about these services as soon as you save the JIF and, therefore, you can test the client screen and the selection screen functionality immediately.

Test the selection screen

Now test the screens.

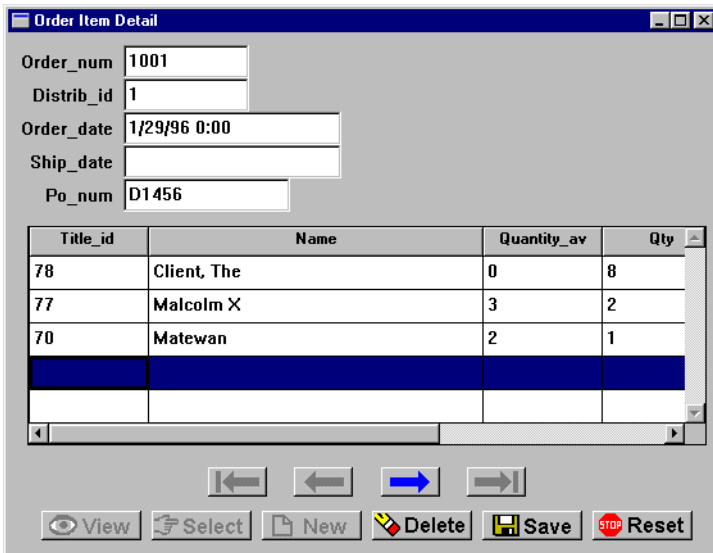
- 32 Give focus to client screen `orditm.scr`.

-
- 33 Choose File→Test Mode or  .

The Order Item Detail screen opens.

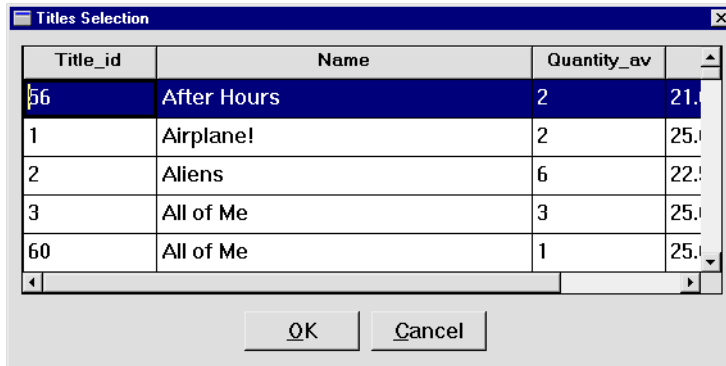
- 34 Choose  .

The first record in the orders table displays. Several `order_item` records are associated with this order. They are displayed in the grid.



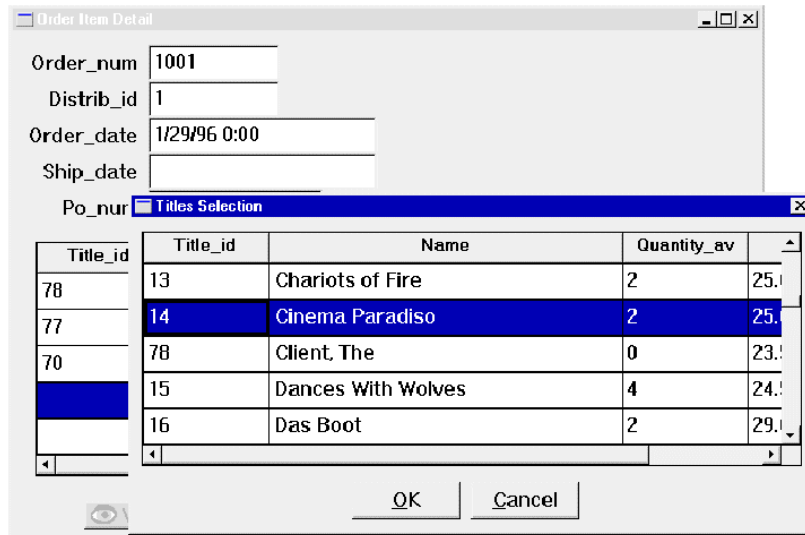
Title_id	Name	Quantity_av	Qty
78	Client, The	0	8
77	Malcolm X	3	2
70	Matewan	2	1

- 35 Click in the `Title_id` field of the first empty row in the grid.
- 36 Press F1 (or HELP).



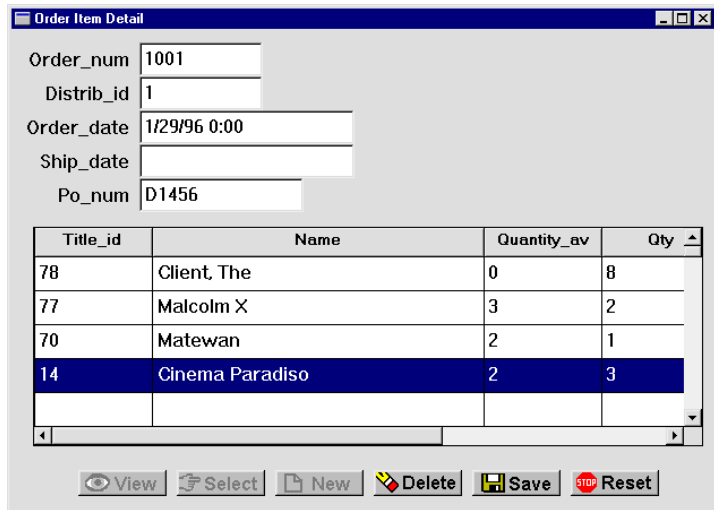
The Titles Selection screen opens and displays all video records in the titles database table. The service `titles_c`, which is defined as the Select Service property of the table view on the selection screen, is called and populates the screen.

- 37 Scroll down to the title Cinema Paradiso and double-click on the ID to select that video.



The selection screen closes and the record you selected appears in the grid on the Order Item Detail screen. The cursor advances to the next data entry field in the grid (`qty`).

38 Enter 3 in the Qty column.



The screenshot shows a window titled "Order Item Detail" with several input fields and a table. The input fields are: Order_num (1001), Distrib_id (1), Order_date (1/29/96 0:00), Ship_date (empty), and Po_num (D1456). The table below has four columns: Title_id, Name, Quantity_av, and Qty. The row with Title_id 14 and Name "Cinema Paradiso" is highlighted in blue, and its Qty value is 3. The other rows are: Title_id 78, Name "Client The", Quantity_av 0, Qty 8; Title_id 77, Name "Malcolm X", Quantity_av 3, Qty 2; Title_id 70, Name "Matewan", Quantity_av 2, Qty 1. At the bottom of the window are buttons for View, Select, New, Delete, Save, and Reset.

Title_id	Name	Quantity_av	Qty
78	Client The	0	8
77	Malcolm X	3	2
70	Matewan	2	1
14	Cinema Paradiso	2	3

Validate the data

Add another item to this order, but this time, enter the title identification number. Panther uses the validation service (`titles_11`) to ensure that the entry is valid.

39 Press TAB to advance to the next empty row and type 55 in the `Title_id` column. Press Enter or TAB.

The screenshot shows a window titled "Order Item Detail" with the following fields:

- Order_num: 1001
- Distrib_id: 1
- Order_date: 1/29/96 0:00
- Ship_date: (empty)
- Po_num: D1456

Below the fields is a table with the following data:

Name	Quantity_av	Qty	Order_▲
Client, The	0	8	23.50
Malcolm X	3	2	24.50
Matewan	2	1	25.00
Cinema Paradiso	2	3	25.00
Willie Wonka and the Chocolate Fact 2	2	1	25.00

At the bottom of the window are several buttons: View, Select, New, Delete, Save, and Reset.

Willie Wonka and the Chocolate Factory is the name of the video associated with the specified ID number.

40 Choose  .

This action saves the new items to the selected distributor's order.

41 Return to the editor.

What did you do?

You created an order entry screen that provides an easy way to add items to an order. You did this by performing these tasks:

- Used the screen wizard to create a master-detail screen that includes information from three database tables.
- Requested that the screen wizard generate selection screens for the additional table specifications.

-
- Tested the selection screen by picking a video record from the list of valid choices.
 - Used the validation service by typing an ID on the client screen and verified that a valid video title is selected.

What did you learn?

You learned:

- The screen wizard lets you build screens that include data from multiple database tables, and provides the links needed to populate the client screen at runtime.
- Selection screens provide users with a list of valid database choices for a field and eliminate the need to type in already defined information.
- The screen wizard generates selection screens for all additional database tables represented on a client screen.
- The screen wizard identifies the services needed to validate data and populate selection screens.

16 Calculating Data from Database Values

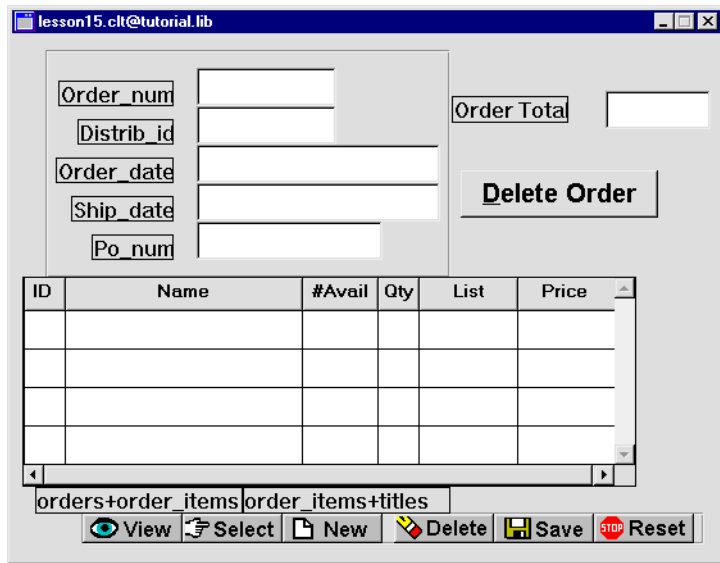
For this lesson, you are provided with an enhanced client screen and service component based on the Order Item Detail screen that you created in Lesson 15. You continue to add to this screen and ultimately connect it with the other screens that you built in the tutorial.

In general, this lesson provides additional ideas and methods for enhancing screen wizard-generated JPL procedures and extending the application's database interaction capabilities.

In this lesson you learn how to:

- Add a widget (`item_total`) to the detail section (in the grid widget) of the screen that derives its data from database values.
- Implement JPL procedures that calculate and recalculate item totals and the grand total as order items are added, deleted, or changed.
- Specify the `item_total` and `order_total` widgets as members of the appropriate table views. Because these widgets are not defined in the source database table, they are known as virtual fields. Its inclusion in the table view allows a virtual field to participate in SQL generation and in transaction manager events.
- Define a new control string and procedure for the screen wizard-generated Delete push button, so a user can delete one detail record from `order_items`.

- 1 If necessary;
 - Reactivate the application now, invoke the editor, and connect to the middleware.
 - Reopen `tutorial.lib` (File→Open→Library).
- 2 Open `lesson15.clt` and `lesson15.svr` from `tutorial.lib`. Use either the Library TOC or menu bar (File→Open→Screen).



`lesson15.clt` includes a single-record master section, an Order Total label and corresponding data entry widget, a Delete Order push button, a grid display detail section, and wizard-generated push buttons.

- 3 Choose File→Save As→Library Member and save:
 - `lesson15.clt` as `orditm.scr` in the `client.lib` library on the application server.
 - `lesson15.svr` as `orditm.scr` in the `server.lib` library on the application server.

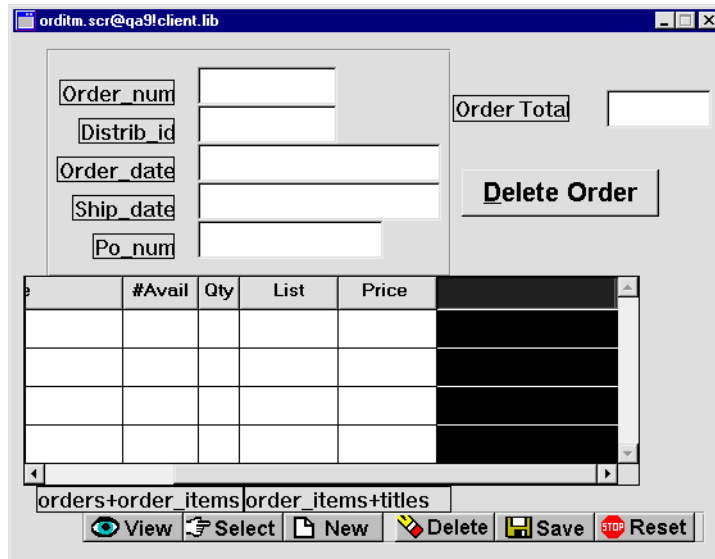
This overwrites the `orditm.scr` screen and service component that you created in Lesson 15.

Note: If you prefer not to overwrite the original `orditm.scr` client screen and service component,, open them and save them under different names before saving the new versions of `orditm.scr`.

Add a column to the grid widget

Enhance the Order Item Detail screen so the grid widget shows a total for each order item. You do so by adding a column to the grid widget.

- 4 Give focus to the `orditm.scr` client screen.
- 5 Choose Create→Single Line Text and click inside the grid widget.



A new, default-sized grid member is added at the rightmost position of the grid widget (next to the Price grid member).

- 6 With the new grid member selected, set these properties:
 - Name property: `item_total`
 - Column Title property: `Total`

This property displays a column title in the grid widget's first row.

- Length property (under Geometry): 8

This changes the onscreen size of the widget, so it is the same length as the Price grid member.

Note: Use the grid's horizontal scroll bar to view offscreen columns. Or resize the grid to display all seven columns: select the grid widget and under Geometry, set the Onscreen Columns property to 7.

Define a currency format

To display totals in currency format, set the Data Formatting property.

- 7 With the `item_total` grid member still selected, under Format/Display, set its Data Formatting property to Numeric.

Numeric format subproperties are displayed. The Format Type property specifies Local currency. This specifies to display the data in the form \$0.00.

More About Data Formatting Options

A variety of formatting options let you control how widget data appears. You can choose from ten predefined date/time formats and ten numeric formats. You can also create custom formats for both data types. The format is automatically applied when data is entered into fields that have their format properties set accordingly.

Default formats are defined in the Panther message file. You can define your own set of format standards by editing the message file. For more information about the message file and custom formats, [click here](#).

Define a math expression (for server processing)

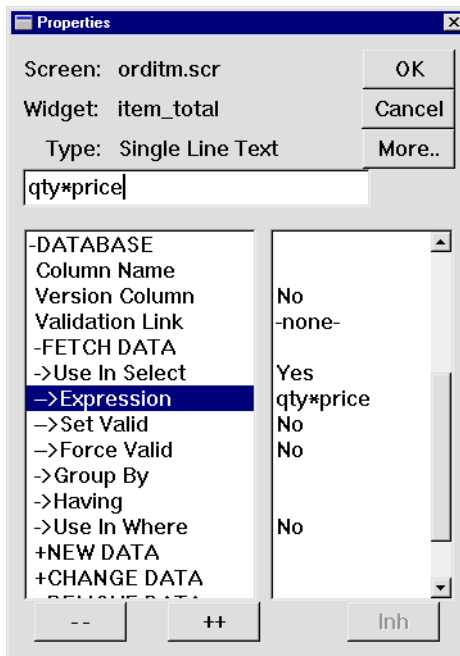
You want the new `item_total` widget to display the total value of each order. This value can be calculated by multiplying values in two other widgets: `qty*price`. You can direct the transaction manager to perform this calculation via the SQL that it generates. To do this, you must set `item_total`'s Use In Select property so it is included in the select list of the generated SQL `SELECT` statement, and provide the appropriate math expression.

Because the server actually performs this processing, the necessary settings for `item_total` must be defined on the service component, where they are accessible to the transaction manager. However, you can set all the properties on the client screen and copy the widget to the service component later. In most cases, redundant property settings are ignored.

- 8 Under Database, in the Fetch Data subcategory, set the Use In Select property to Yes.

Related subproperties are displayed.

- 9 In the Expression subproperty, enter: `qty*price`.



The expression uses the values from both widgets belonging to the `order_items` table to yield a calculated result.

Add the widget to a table view

The transaction manager includes `item_total` in the SQL generation only if the widget is part of the appropriate table view—in this case, `order_items`. Widgets that are outside a table view are excluded from SQL generation.

The next few steps show how to identify widgets that are table view members and how to change table view membership. To do so, you must select the `order_items` table view widget via the DB Interactions window or the [Widget List](#) and access its properties.

- 10 Give focus to the client screen and choose View→Widget List.

The Widget List opens and lists all the components on the current screen: the widget's name in the right-hand column and its type in the left column.

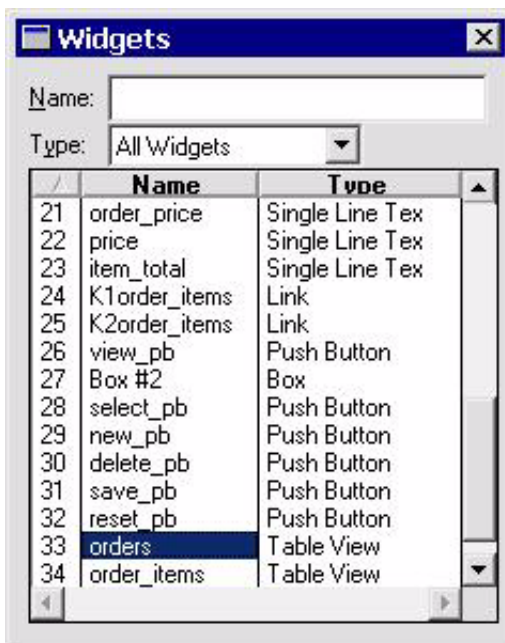
More About the Widget List

You can use the [Widget List](#) as another way to select widgets. All widgets on the current screen are listed in the [Widget List](#), including invisible widgets, such as selection groups, synchronization groups and table views.

When you select an item from the list, the widget on the screen is also selected. The [Properties](#) window displays the properties common to the widgets that are currently selected, or of the screen if no widgets are selected.

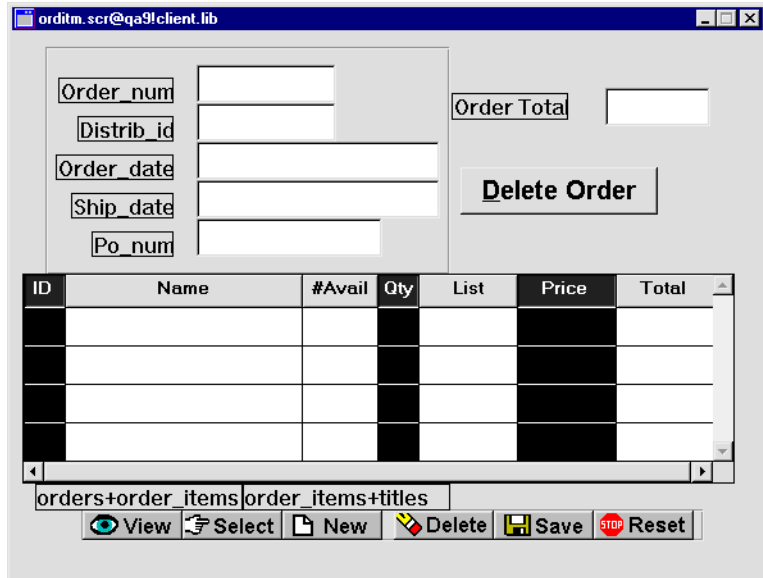
You can select multiple contiguous widgets in the list with a click+drag or Shift+click; Ctrl+click to toggle membership in the selection set or to select non-contiguous items.

- 11 Select the `order_items` table view from the list of names.



If the Properties window displays table view properties, it confirms the table view is selected.

- 12 Choose Edit→Group→Select Members.



All members of the `order_items` table view are selected: ID (`title_id`), Qty (`qty`), and Price (`price`).

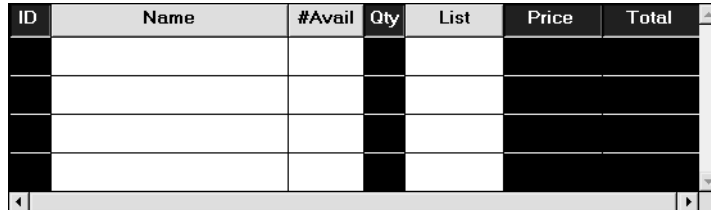
More About Groups and Group Membership

Widgets can belong to several types of groups. Each group type has its own set of properties that control group behavior:

- **Synchronization**—Controls how widgets scroll together. By default, all members of a grid widget belong to a synchronization group and therefore scroll together. Synchronized group properties can specify, for example, scroll increment and scroll behavior when the last item has focus.
- **Table view**—Contains one or more related widgets, usually associated with and named for a single database table. Table view members can also include widgets that are not part of the database table in order to display derived data. Table view properties give the transaction manager the information it needs to generate SQL statements—for example, sort order, or whether the table view is updatable.
- **Selection**—Comprises specific widget types (multiple radio buttons, toggle buttons, or check boxes, and single list boxes) that enhance the user interface by providing users with visual choices. You can define selection group properties such as the number of selections that a user can make, the tabbing order, and the group's entry, exit and validation functions.

13 Add the `item_total` (Total) grid member to the selection set:

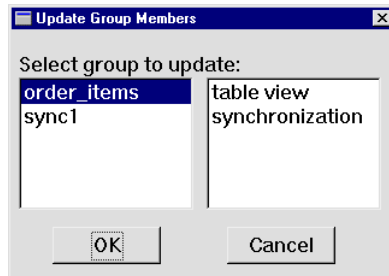
- *Windows* Shift+click on the column's title.
- *Motif* Shift+click within a cell in the column.



ID	Name	#Avail	Qty	List	Price	Total

14 With all four members selected, choose Edit→Group→Update Group Members.

The Update Group Members dialog opens.



The Update Group Members dialog lists all groups to which the selected widgets belong.

15 Select `order_items` as the group to update and choose OK.

All members are deselected.

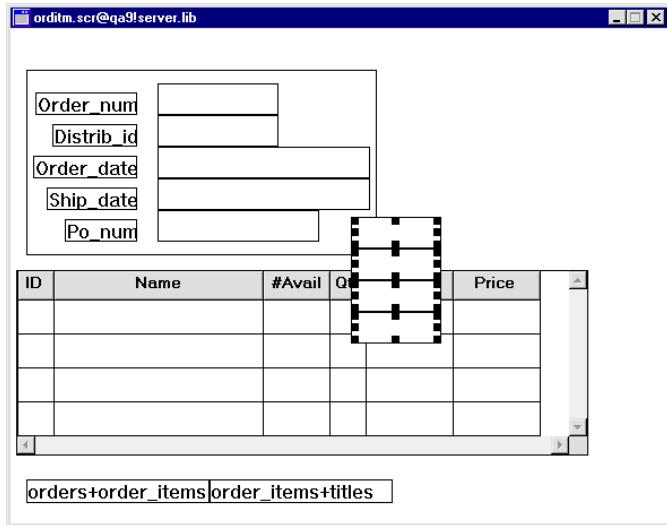
Note: To confirm the new membership, repeat steps 11 and 12. The Total grid member should be selected along with `title_id`, `qty`, and `price`. If it is not, repeat steps 13-15.

16 Save the screen.

Calculate results on the server

The transaction manager generates SQL statements on the server. Therefore, you need to copy the `item_total` widget to the `orditm.scr` service component so its `SELECT` expression is used. Maintaining the same property settings on both the client and its service component ensures that the client screen and its service component remain synchronized.

- 17 Select the Total (`item_total`) grid member on the `orditm.scr` client screen.
- 18 Choose Edit→Copy, then paste it (Edit→Paste) on the `orditm.scr` service component.



An array of single line text widgets is pasted onto the screen.

- 19 Drag the array to the grid widget.

The Total column appears as the rightmost column in the grid widget.

Name	#Avail	Qty	List	Price	Total

-
- 20 Save and close the `orditm.scr` service component.

Calculate results on the client

Order entry screens often include a grand total as well as item totals. In order to display grand totals, the `orditm.scr` client screen has a single line text widget `order_total` and a corresponding Order Total label. The value in `order_total` is calculated from the sum of all values in the `item_total` column. The procedure that performs this calculation must be called on three occasions:

- When data is selected from or saved to the database.
- A row of order item data is deleted from the grid widget.
- A value in the `qty` or `price` columns changes.

Because the values required to calculate a total are already retrieved from the database, the results can be calculated solely on the client—no service call is needed. Therefore, the procedure that perform this operation should be in the screen's JPL Procedures property so it is accessible to all other client procedures.

- 21 Select the `orditm.scr` client screen (deselect all widgets).

- 22 Under Focus, select the JPL Procedures property.

The JPL edit window opens. It currently contains the screen entry procedure `enter_screen`, which behaves like the screen entry procedure that you implemented on the `dstord.scr` client screen. It receives the order identification number (`order_num`) from the calling screen (`dstord.scr`) and executes a `sm_tm_command("SELECT")` to fetch the specified order.

- 23 Scroll to the bottom of the JPL edit window and insert `upd_total.jpl` from `tutorial.lib`. The `upd_order_totals` procedure is read into the JPL edit window:

```
proc upd_order_total()  
{  
    order_total = @sum(item_total)  
    return 0  
}
```

This procedure calculates the order's total with the aggregate function `@sum`.

Update totals on transaction manager events

The grand total in `order_total` needs to be updated whenever the transaction manager performs a `SELECT` or `SAVE` command. To do this, attach a transaction manager hook function to the client screen's root table view.

- 24** Scroll to the bottom of the JPL edit window and insert `evnt_ord_clt.jpl` from `tutorial.lib`. The `tm_events_clt` function is read into the JPL edit window:

```
proc tm_events_clt(event_id)
{
  if (((event_id == TM_POST_SELECT) || (event_id == \
    TM_POST_SAVE)))
  {
    call upd_order_total()
  }
  return TM_PROCEED
}
```

The `tm_events_clt` procedure determines whether a `SELECT` or `SAVE` transaction manager command has executed. If either condition is true, it calls the `upd_order_total` procedure. The `TM_PROCEED` return value tells the transaction manager to resume processing.

- 25** Choose OK.
- 26** Return to the `orditm.scr` client screen and select its root table view `orders`, via the DB Interactions window or [Widget List](#).
- 27** Under Transaction, enter `tm_events_clt` in the Function property.

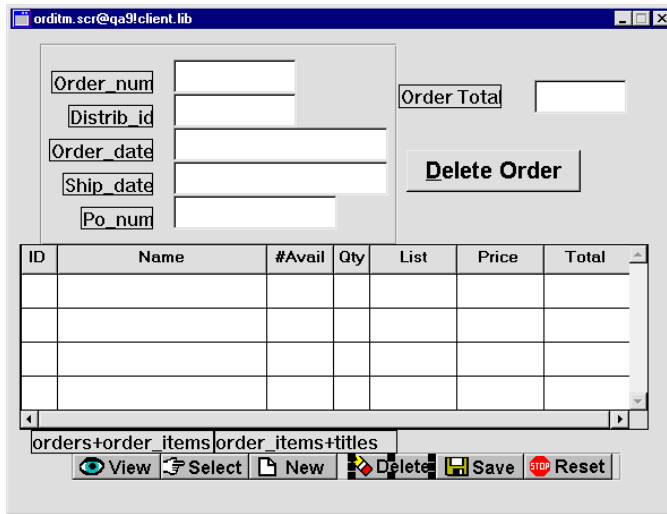
The root table view now has `tm_events_clt` set as its hook function. The transaction manager executes this function when it starts traversing the screen's table views.

Delete a detail record

The screen wizard-generated Delete button was copied and renamed `delete_order_pb` on the `lesson15.clt` screen. Its label was changed to Delete Order and its Pixmap properties were removed. However, its behavior remains the same: it calls a wizard-generated procedure that deletes the master and related details.

To allow a user to delete a single order item instead of the entire order, modify the Delete button at the bottom of the `orditm.scr` client screen: rename the button and assign a new control string to invoke the appropriate procedure.

28 Select the Delete button at the bottom of the `orditm.scr` client screen.



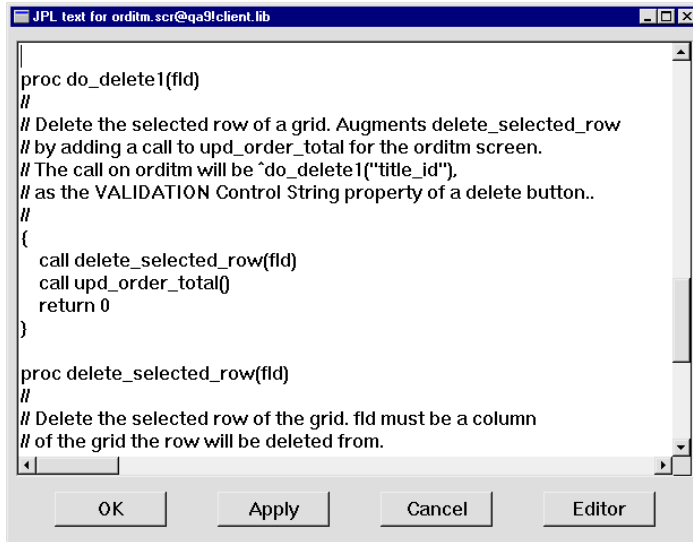
29 Change the widget's name to `delete1_pb`.

30 Under Validation, set the Control String property to `^do_delete1("title_id")`.

When a user chooses the Delete push button, `^do_delete1` is called and is passed the argument `title_id`, the name of a widget to use in the procedure.

31 Return to the JPL edit window for `orditm.scr` client screen.

32 Scroll to the bottom and insert `delete1.jpl` from the `tutorial.lib` library.



```
JPL text for orditm.scr@qa9!client.lib

proc do_delete1(fld)
//
// Delete the selected row of a grid. Augments delete_selected_row
// by adding a call to upd_order_total for the orditm screen.
// The call on orditm will be `do_delete1("title_id")`.
// as the VALIDATION Control String property of a delete button..
//
{
  call delete_selected_row(fld)
  call upd_order_total()
  return 0
}

proc delete_selected_row(fld)
//
// Delete the selected row of the grid. fld must be a column
// of the grid the row will be deleted from.
```

The `delete1.jpl` library member is read into the JPL edit window. It contains two procedures: `do_delete1`, which calls `delete_selected_row`:

```
proc do_delete1(fld)
{
  call delete_selected_row(fld)
  call upd_order_total()
  return 0
}

proc delete_selected_row(fld)
vars grid_name occ
{
  grid_name = @widget(fld)->grid
  occ = @widget(grid_name)->grid_current_occ
  call sm_i_doccure(fld, occ, 1)
  return 0
}
```

`do_delete1` first calls `delete_selected_row`.

`delete_selected_row` deletes the selected row from the detail grid as follows:

- Gets `title_id`'s grid property, which returns the name of the grid in which `title_id` is member.

-
- Gets the grid's `grid_current_occ` property, which returns the occurrence number of the current grid row selection.
 - Calls `sm_i_doccure` to delete the grid row selection.

After the grid row is deleted, `delete_selected_row` returns to `do_delete1`, which next calls `upd_order_total`. This procedure recalculates the value in `order_total`.

When the user saves changes to the database by choosing Save, the record in `order_items` that corresponds to the deleted grid row is deleted.

33 Choose OK.

Validate client data

Item totals and the grand total must be recalculated whenever a value in quantity or price changes. To detect changes in either column, you need to set their Validation Func property. The function that this property specifies executes whenever an occurrence in either column loses focus—for example, the user presses TAB.

More About Widget Validation

When a widget loses focus at runtime (the user presses TAB for example), Panther calls the widget's validation function, then its exit function, and finally the automatic field function.

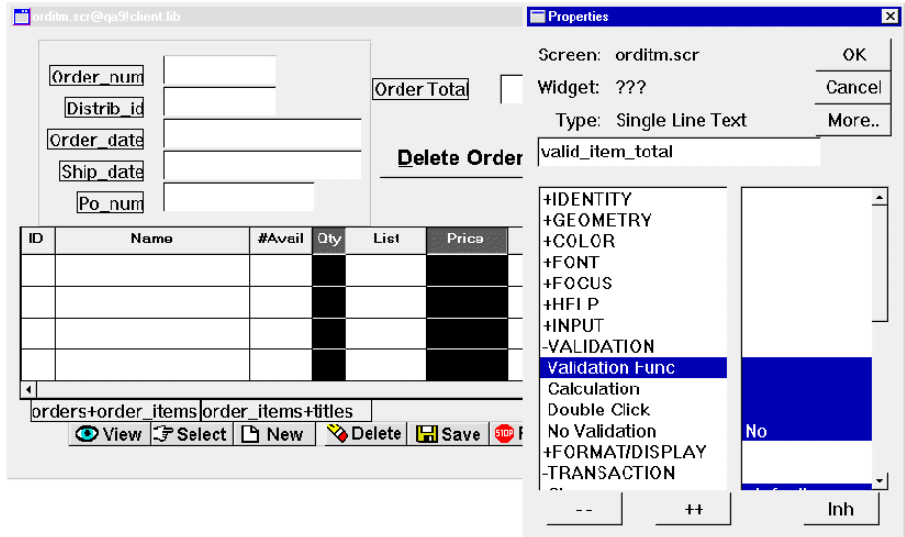
Validation functions are also called under the following conditions:

- As part of screen validation. Screen validation occurs when the XMIT key (for example, an OK button) is pressed or when the screen closes. At that time, all fields on the screen are validated via the function `sm_s_val`.
- When the application code calls library functions for field validation or screen validation.

[CLICK HERE](#) for more information about screen and field validation, and group and grid validation

34 Select the Qty (`qty`) and Price (`price`) grid members.

35 Under Validation, enter `valid_item_total` in the Validation Func property.



36 Return to the JPL edit window.

37 Scroll to the bottom of the window and insert `order_valid.jpl` from the `tutorial.lib` library.

The `order_valid.jpl` library member is read into the JPL edit window and includes the procedure `valid_item_total`:

```
proc valid_item_total(field_no, data, occ, context)
{
  item_total[occ]=price[occ] * qty[occ]
  if (!(context & K_SVAL) || \
      occ == @widget("Detail")->num_occurrences))
  {
    call upd_order_total()
  }
  return 0
}
```

The `valid_item_total` procedure updates `item_total` for the selected item using the expression `price[occ] * qty[occ]`. The `if` command checks the context in which the procedure is invoked. It also specifies two conditions, one of which must be satisfied to execute the `if` statement block, which calls `upd_order_total`:

- The first condition `!(context & K_SVAL)`—the negation of the screen validation bit `K_SVAL`) states that if the procedure is called on widget validation, update the order total by calling `upd_order_total`.
- On the other hand, if the widget is being validated as part of the screen's validation, the order total will only be updated after validating a grid member in the last row of the grid.

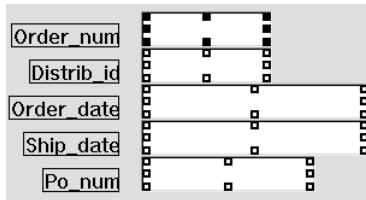
38 Choose OK to save all changes in the JPL edit window.

Clearing data in a virtual field

The `order_total` widget is not derived from a database table so it is not included in transaction manager transactions. Therefore, when you add a new order, delete an existing one, or choose the Reset button, the content of the `order_total` widget doesn't clear. To clear `order_total` when these transaction manager events occur, you must add this widget to the screen's root table view orders.

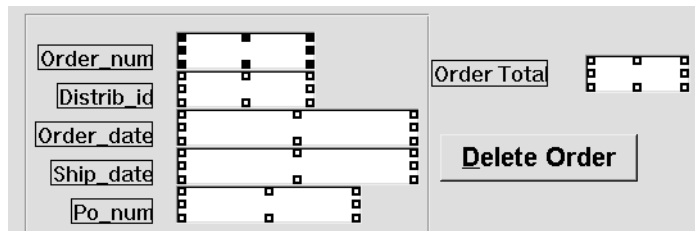
39 Select the orders table view with the [Widget List](#) or the DB Interactions window.

40 Choose Edit→Group→Select Members.



All members on the screen that belong to the orders table view are selected.

41 Add the `order_total` single line text widget to the selection group (Shift+click).



- 42 Choose Edit→Group→Update Group Members.

The `order_total` widget is now controlled by the same display styles and transaction behavior as other widgets that belong to the orders table view.

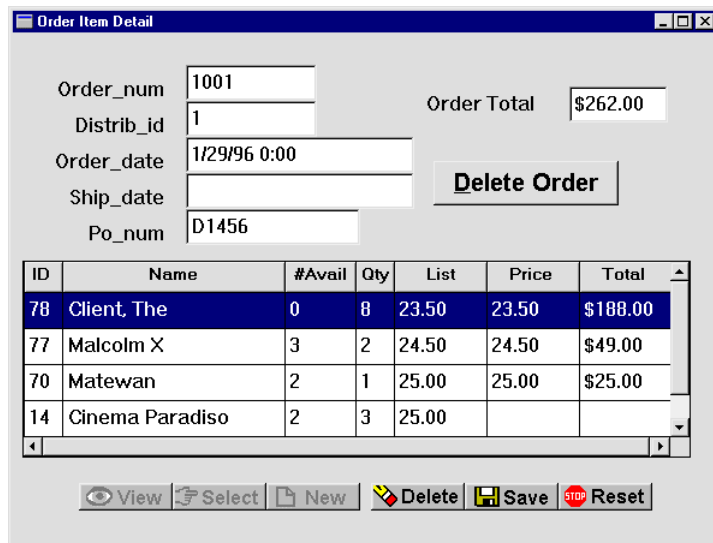
- 43 Save `orditm.scr` to `client.lib` on the application server.

Update a detail record

Test it out! When you go into test mode, the `orditm.scr` screen entry procedure executes a `SELECT` command and displays the first order record in the database.

- 44 Choose File→Test Mode (press F2) or  .

The first order record is displayed.



The screenshot shows a window titled "Order Item Detail" with the following fields and values:

- Order_num: 1001
- Distrib_id: 1
- Order_date: 1/29/96 0:00
- Ship_date: (empty)
- Po_num: D1456
- Order Total: \$262.00

A "Delete Order" button is located to the right of the Ship_date field.

ID	Name	#Avail	Qty	List	Price	Total
78	Client, The	0	8	23.50	23.50	\$188.00
77	Malcolm X	3	2	24.50	24.50	\$49.00
70	Matewan	2	1	25.00	25.00	\$25.00
14	Cinema Paradiso	2	3	25.00		

At the bottom of the window, there is a toolbar with buttons: View, Select, New, Delete, Save, and Reset.


Item totals and the order total are calculated on screen. Rows that lack quantity or price data also omit total data.

- 45 Click in the price field for Cinema Paradiso. Enter 20.00 and press TAB.


The totals are immediately updated when you tab out of the field.

- 46 Click in the row with the ID 70 and choose  .


The row data clears and the order's total is adjusted. The total is recalculated from the client screen's current values and so does not require any database transaction.

- 47 Choose  .

The database is updated with the changed data in order 1001.

- 48 Choose  .

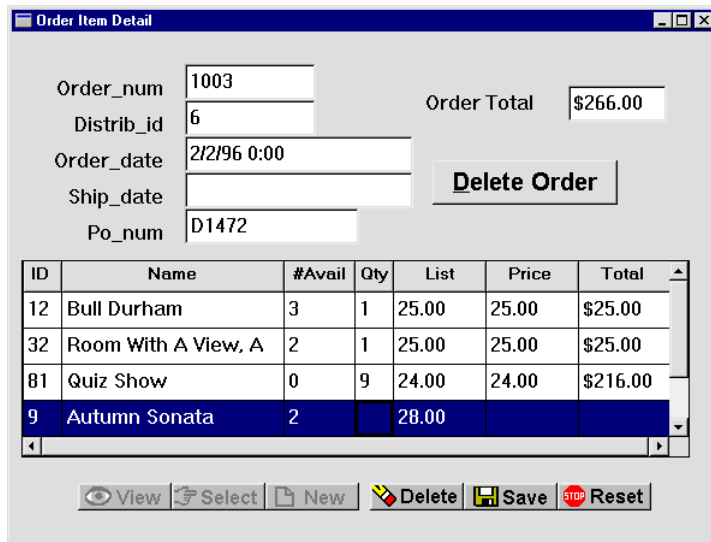
All fields including `order_total` are cleared of data.

- 49 Type 1003 in the `order_num` field and choose  .

The order associated with distributor 6 displays.

- 50 Click into the ID field of the first empty row. Enter 9 and press TAB.


The video displays and the cursor advances to the `qty` field.



ID	Name	#Avail	Qty	List	Price	Total
12	Bull Durham	3	1	25.00	25.00	\$25.00
32	Room With A View, A	2	1	25.00	25.00	\$25.00
81	Quiz Show	0	9	24.00	24.00	\$216.00
9	Autumn Sonata	2	28.00			

- 51 Enter 2 for the quantity, press TAB, and enter 25.00 in the price field. Press TAB again.

The totals are immediately recalculated.

- 52 Choose  Save.
- 53 Return to the editor to add some final touches.

Connect two screens

To connect the `orditm.scr` (Order Item Detail) screen with the `dstord.scr` (Distributor Orders) screen created in Module 3, you must include the `send_order_data` procedure on the `dstord.scr` client screen. The `send_order_data` procedure calls the `orditm.scr` screen.

- 54 Open the `dstord.scr` client screen from `client.lib`.
- 55 With the screen selected, under Focus, select the JPL Procedures property.
- 56 Scroll to the bottom of the JPL edit window and insert `send_order.jpl` from `tutorial.lib`.

```
proc send_order_data()
{
    vars occ
    occ = Detail->grid_current_occ
    if (order_num[occ] == "")
    {
        msg emsg "First select an order."
        return 1
    }
    send DATA order_num[occ]
    call sm_jwindow("(+5,+5)orditm.scr")
    return 0
}
```

The `send_order_data` procedure is called when you double-click on a specific order on the Distributor Order screen (you implemented this behavior in Lesson 14). The data required to execute the appropriate SQL is sent and the `orditm.scr` screen opens.

- 57 Choose OK.
- 58 Save all open screens and proceed to the tutorial finale.

What did you do?

You enhanced the order entry screen to display totals for each item in the order and a grand total for the entire order. You did this by performing these tasks:

- Added a virtual database column to the `order_items` table view so the transaction manager can build a SQL `SELECT` statement using the appropriate math expression on the service component. As a result, when the Order Item Detail screen first opens, the total for each item in the order is tallied and displayed in the grid widget.
- Implemented field validation and screen validation to ensure that totals, for both individual order items and for the order as whole, are recalculated when an `order_items` record is updated, inserted, or deleted.
- Added a virtual database column to the orders table view so `order_total` clears when the screen is cleared of data.

What did you learn?

You learned:

- Adding a widget to a table view group allows a “virtual” widget to behave as though it belongs to a database table. It can be included in transaction manager events and behave as other database-derived widgets from the same table view behave.
- The screen-wizard Delete procedure deletes the master and its detail items. You can replace this with a delete procedure that deletes only one detail item.
- Processing can take place on the server and on the client. By using data that is already displayed on the client screen, your application can handle simple processing to give end users immediate results. On the other hand, if database information is needed to compute particular events, a service is required to invoke the request for data, and business logic is applied on the application server. The results are then returned to the client.



17 The Finale

Congratulations! You have created a fully functional, three-tier Panther application. So, now take it from the top!

- 1 If you have not started up your application, boot your application now, invoke the editor and open a middleware session.

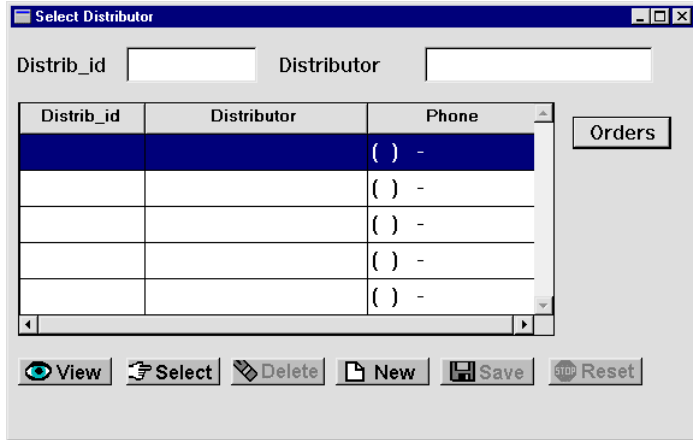
- 2 Choose File→Test Mode (press F2) or  .

- 3 Choose Options→Open Screen. The Enter Screen Name dialog box opens.



- 4 Enter `dstselect` and choose OK.

Any screen that might have been open in test mode closes, and the Select Distributor screen opens.



- 5 Choose  .

All distributors in the vidsales database are displayed.

- 6 Scroll down the list and double-click on the row for distributor ID.

The Distributor Orders screen opens and displays the information associated with the distributor Video Signs, Inc.—this is the record you added in Lesson 9.

Select Distributor

Distributor Orders

Distrib_id

Distrib_id
3
4
5
6
7
8

View

Distrib_id: 7
 Distributor: Video Signs, Inc.
 Address1:
 Address2:
 City:
 State_prov:
 Postal_code:
 Country:
 Phone: (555) 123-4567

Order_num	Order_date	Ship_date	Po_num
1211	1/5/97 0:00		A123

View Select New Delete Save Reset

7 Double-click on the Order_num 1211.

Order Item Detail

Order_num: 1211 Order Total: \$0.00
 Distrib_id: 7
 Order_date: 1/5/97 0:00
 Ship_date:
 Po_num: A123

Delete Order

ID	Name	#Avail	Qty	List	Price	Total

View Select New Delete Save Reset

There are no order items associated with the order, so now you can add some.

- 8 Enter the following order items, pressing TAB between each entry. The totals are updated appropriately.

ID	Qty	Price
15	3	20.00
30	2	29.00

Order Item Detail

Order_num: 1211
Distrib_id: 7
Order_date: 1/5/97 0:00
Ship_date:
Po_num: A123

Order Total: \$118.00

Delete Order

ID	Name	#Avail	Qty	List	Price	Total
15	Dances With Wolves	4	3	24.50	20.00	\$60.00
30	Rashomon	2	2	29.00	29.00	\$58.00

View Select New Delete Save Reset

- 9 You can add, update, or delete order items. Or delete the entire order by choosing the Delete Order button.

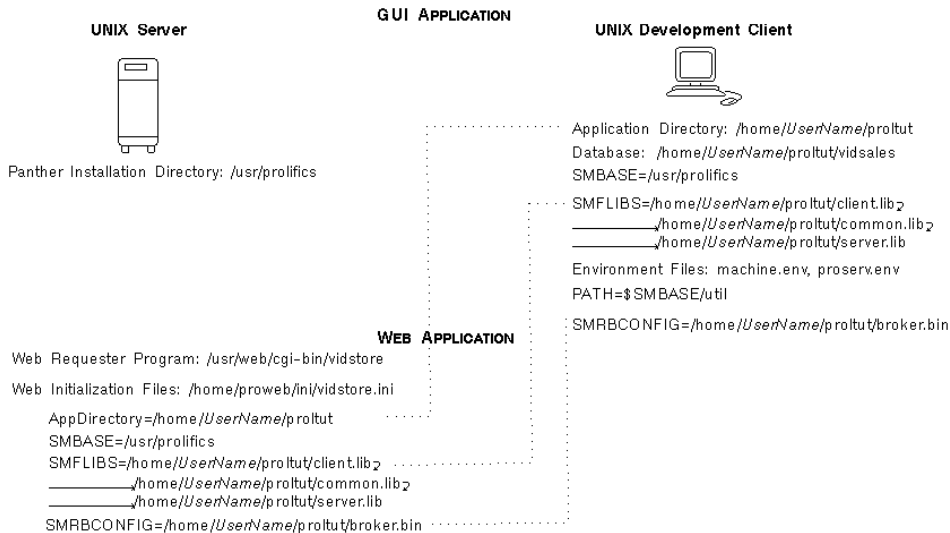
You can always find something more to do that will improve an application. Continue to enhance the screens—for instance, add a Details push button to invoke the `send_data` procedure from the Distributor Orders screen, or add a Done button to the Order Item Detail screen that will take the user back to the Select Distributor screen.

You have successfully completed the tutorial!

A Setting Up the Tutorial

The tutorial steps assume that the Panther development client, Panther application server, and Panther web application server are installed on the same machine. The following diagrams illustrate typical configurations for the tutorial using a Panther web application named vidstore.

For a three-tier JetNet application on a UNIX server using UNIX development clients, a typical configuration for the tutorial would be:



The tutorial directory created in your home directory serves as the application directory and contains a setup file, the application libraries, the environment files, and the database.

For a three-tier JetNet application on Windows, a typical configuration would be:

NT Server



GUI APPLICATION

Panther Installation Directory: C:\Prolifics\Panther
Application Directory: C:\Prolifics\Panther\samples\proltut
Database: C:\Prolifics\Panther\samples\proltut\vidsales
NT Environment: SMBASE=C:\Prolifics\Panther
Tutorial Initialization File: C:\Winnt\tutorstd.ini
SMFLIBS=\$SMBASE\samples\proltut\client.lib
_____ /home/UserName/proltut/common.lib
_____ /home/UserName/proltut/server.lib

WEB APPLICATION

Web Requester Program: C:\netPub\scripts\vidstore.exe
Web Initialization Files: C:\Winnt\vidstore.ini
AppDirectory=\$SMBASE\samples\proltut
SMBASE=C:\Prolifics\Panther
SMFLIBS=\$SMBASE\samples\proltut\client.lib
_____ /home/UserName/proltut/common.lib
_____ /home/UserName/proltut/server.lib

The JetNet application can use a Windows development client with a UNIX web application server if the application libraries and database are available on the UNIX HTTP server. The following diagram illustrates the Panther application server and web application server installed on the UNIX server:

Windows Development Client



Panther Installation Directory: C:\Prolifics\Panther
Database: C:\Prolifics\Panther\samples\proltut\vidsales
(use to import the database objects to the repository)
SMRBHOST=*UnixMachineName*
SMRBPOR=*UnixPortNumber*
Tutorial Initialization File: C:\Winnt\tutorstd.ini
SMBASE=C:\Prolifics\Panther
SMFLIBS=\$SMBASE\samples\proltut\client.lib
_____home/*UserName*\proltut\common.lib
_____home/*UserName*\proltut\server.lib

UNIX Server



Panther Installation Directory: /usr/prolifics
Web Requester Program: /usr/web/cgi-bin/vidstore
Web Initialization Files: /home/proweb/init/vidstore.ini
AppDirectory=/home/*UserName*/proltut
SMBASE=/usr/prolifics
SMFLIBS=/home/*UserName*/proltut/client.lib
_____home/*UserName*/proltut/common.lib
_____home/*UserName*/proltut/server.lib

UNIX Development Directory

Application Directory: /home/*UserName*/proltut
Libraries: /home/*UserName*/proltut/client.lib
_____home/*UserName*/proltut/common.lib
_____home/*UserName*/proltut/server.lib
Database: /home/*UserName*/proltut/vidsales
Environment Files: machine.env, proserv.env
PATH=\$SMBASE/util
SMRBCONFIG=/home/*UserName*/proltut/broker.bin



B Troubleshooting

This section describes problems you might encounter when setting up the Panther environment, or when running Panther, and tells you where to look for more information.

Error Files

The following files are used to record errors in the various Panther components. If you have problems during the setup procedure, check the directory from which you run the tutorial (`pro1tut`) for the existence of any of the files below and check its contents.

- `stderr`—Errors from the Panther application server are written to this file and include errors about environment variables and licensing problems.
- `stdout`—Includes errors specific to the application server.
- `ULOG.*`—Lists all middleware session activity and errors, particularly IPC resource errors. A new `ULOG.*` is started each day, with the date contained in the file name, in the format `ULOG.ddmmyy`.
- `error.log`—Contains errors generated by the Web application server.

If these files provide no clear direction, contact Prolifics Technical Support Services.

Setup and Connection Problems

This section describes some of the more common areas where you might encounter problems while setting up the Panther environment or using Panther, and provides hints for resolving them. If you need more assistance, contact Prolifics Customer Support.

Starting the application or servers

Booting the application

If you are unable to boot your application with `rbboot` (indicated by a `FAILURE` message), check the `ULOG.*` file in the `proltut` directory. Usually, a failure at this point indicates inadequate IPC resources. If the `ULOG` indicates IPC resource errors, refer to the *JetNet Guide/Oracle Tuxedo Guide* for more information.

If only the server running the `proserv` executable fails to start, check the `stderr` and `stdout` files.

Before attempting to restart the application, you need to shut down the servers, even if you received errors, by typing (in the window in which you ran `rbboot`):

```
rbshutdown
```

This utility shuts down the Panther application, including all active servers.

Activating a server

If you are unable to activate an application server using the JetNet Manager, check the following files in your `proltut` directory on the server, in the order in which they are listed:

```
stderr, stdout, and ULOG.*.
```


Setting up the Web application server

If you are unable to start the Web application server from your Web browser, check the files `ULOG.*` and `error.log` in the `proltut` directory.

If you have any problems that require changes to the `proltut.ini` file, or if `error.log` contains an error message indicating that you failed to connect to the server, you must stop and restart your Web application server to have the changes take effect. Shut down the server by typing:

```
$SMBASE/util/monitor -stop proltut
```

Starting the client

Starting a UNIX client

If you are unable to start a Panther client workstation when typing `prodev` at the command line, make sure you have applied the client environment settings to the current window by typing:

```
.. /setup.sh
```

Connecting the client

Connecting to the server remotely

If you receive an error message when you attempt to connect to the middleware from a PC client and cancel out of the Connect dialog, the editor workspace opens but you are not connected to the server. Check that the application and its servers are running and if not, start them up using `rbboot`. Once they have successfully started, choose `File→Open→Middleware Session` in the editor. You will also need to open the remote libraries and remote repositories.

Connecting to the server locally

If you attempt to open a middleware session on a local UNIX client and receive the message that the `TUXCONFIG` file does not exist, most likely your `SMRBCONFIG` setting does not exactly match the Local JetNet Configuration File value in JetMan. The usual cause of this is that the value of `SMRBCONFIG` in `setup.sh` contains a symbolic link as part of your home directory location, and that JetMan used the actual disk location.

Accessing remote libraries

- If the Remote button is grayed out in the Library Table of Contents dialog box, and therefore you cannot access remote libraries, you are probably not connected to the middleware. Make sure the application and servers are running—see the section above on connecting to the server.
- The Remote button will also be grayed out if there is no file access server (`devserv`) running for the application. Check the status in JetMan of the file access server.
- If you attempt to open a remote library and the Library Table of Contents dialog box displays an error next to the library name, or if you get a library format error when you try to save a library member, the client machine may have timed out.

If a client sits idle for a default period of 60 minutes, it is disconnected from the server; check the `ULOG.*` error file to verify that this is the case. If so, save any open modified screens to a local library, close the remote libraries using `File→Close→Library`, and reconnect to the application by choosing `File→Open→Middleware Session`. Once you reconnect, you can open the remote libraries again; you can then open your modified screens and choose `Save As` to save them to the remote libraries.

Index

Symbols

% (percent sign) [11-15](#)

\ (backslash) [12-2](#)

A

Additional table [15-1](#), [15-3](#)

Application

booting [1-11](#), [1-25](#)

connecting to [1-13](#), [1-26](#)

naming [1-8](#), [1-21](#)

restarting [1-13](#), [1-26](#)

Application architecture

specifying [7-8](#)

Application directory

creating [1-2](#), [1-16](#)

Auto Advertised Services option [2-3](#)

B

binherit [12-5](#)

Boot

application [1-11](#), [1-25](#)

unable to [B-2](#)

Bourne shell [3-10](#)

broker.bin

creating [1-7](#), [1-20](#)

specifying location of [1-11](#), [1-24](#)

C

Calculation expression [16-4](#)

CGI (Common Gateway Interface) [5-1](#)

Clearing data [16-17](#)

Client screens

testing [9-3](#)

Client setup [3-1](#), [B-3](#)

under UNIX [3-2](#)

Client/server

comparison [3-2](#)

Column Title property [12-4](#)

Configuration

for the tutorial [A-1](#)

Control string

property [16-13](#)

Currency format [16-4](#)

D

Data Formatting property [16-4](#)

Database

adding data to [9-6](#)

connecting directly to [6-6](#)

connecting via server initialization [2-4](#)

importing from [6-7](#)

saving changes to [9-6](#)

updating [9-5](#)

viewing data [9-4](#)

Database connections [3-14](#)

- Database properties 10-5
- DB Interactions window 13-14
- Debugger
 - description 3-11
- Default/Cancel property 12-10
- Delete Order push button 16-12
- Delete Service property 11-7
- delete_selected_row procedure 16-14
- delete1.jpl 16-14
- Detail section
 - defining contents of 7-5
- Development access server 2-5
- devserv 1-17
 - setting the environment 1-5
- do_delete1 procedure 16-14
- Dominant widget 11-12
- Double Click property 14-2
- Double-click event 14-2
- Dynamic label widget 12-4

E

- Edit Mask property 12-2
- Editor
 - description 3-6, 3-8
 - invoking from command line 5-19
- enter_screen procedure
 - on dstord client screen 13-9
 - on dstslect client screen 14-4
 - on orditm client screen 16-11
- Environment setup 1-4
- Error files B-1
- error.log file B-1
- evnt_ord_clt.jpl 16-11, 16-12
- Executables
 - for development access server 1-5, 1-17
 - for standard server 1-5, 1-17
- Expression property 16-5

F

- Format Type property 16-4
- Function property 13-14, 16-12

G

- Generate unique ID 13-12, 13-16
- Grid widget
 - adding member to 16-3
 - copying from one grid to another 16-10
 - delete row in 16-14
 - selecting 14-2
 - viewing offscreen columns in 16-4
- Group widgets
 - confirming membership 16-9

H

- Hidden property 13-16
- Hook functions
 - invoking 13-14, 13-17
 - on dstord client screen 13-16
 - on dstord service container 13-11
 - on orditm client screen 16-12

I

- Import 6-7
- Inherit From property 12-12
- Initialization file
 - for Web 5-3
- Insert procedure 13-13
- Insert Service property 11-7

J

- JetNet 1-6, 1-19
 - features 3-16
- JetNet configuration file
 - naming application 1-8, 1-21

specifying location 1-9, 1-22

JetNet manager

- activate server 2-6

JIF 4-1, 8-1

- description 3-11

JIF editor

- invoking 8-2

JPL 13-1, 13-23

- delete procedures 16-14
- enter_screen procedure
 - on dstord client screen 13-9
 - on dstslect client screen 14-4
 - on orditm client screen 16-11
- hook functions 13-11
- insert procedure 13-13
- send_data procedure 13-4, 16-20
- tm_events_clt hook function 13-17, 16-11, 16-12
- tm_events_svr hook function 13-12
- validation function 16-16

JPL edit window 13-22

JPL Procedures property 13-3

K

K_EXPOSE flag 14-4

K_SVAL 16-17

Keystroke Filter property 12-2

Korn shell 3-10

L

Label property 12-4

Layout specifications 7-7

LD_LIBRARY_PATH 1-3

Length property 16-4

Library

- accessing remote B-4
- application 1-4, 1-16
- defined 3-6
- opening from JPL Program Text dialog 13-4

- opening from TOC 11-2

Library TOC 11-2

Link service 15-9

LM_LICENSE_FILE 1-3

Local currency 16-4

Local JetNet Configuration File property 1-9, 1-22

M

Master section

- specifying contents of 7-5

Math expression

- calculating on the server 16-4

Menus

- description 3-10

Middleware

- configuring 1-6
- in three-tier architecture 3-16

Minimum Instances 2-3, 2-5

N

Name property 11-14

Naming conventions

- for selection screens 15-7
- for services 8-4

New command 9-6

newapp directory 1-4, 1-16

O

Onscreen Columns property 16-4

Operator property 11-15

Order total 16-11

order_valid.jpl 16-16

P

Panther

- components of 3-6

- overview 3-1
- Password 1-14, 1-27
- Permissions
 - changing for shared files 5-21
- Primary keys 9-5
- Programming
 - in Panther applications 3-12
- proltut
 - creating 1-2, 1-16
- Properties
 - setting 10-1
- Properties window 10-3
- proserv 1-17
 - setting the environment 1-5
- Push button widget 12-9

R

- rbconfig 1-6
- rbshutdown 3-7, 3-14, 4-7
- Reading path 3-10
- Rearrange database columns 15-4
- Remote library B-4
 - configuring server to access 2-5
- Repository
 - creating 6-5
 - description 3-6, 3-9
 - opening 7-2
 - opening by default 6-5
 - opening screen in 11-10
 - propagating changes from 12-5
 - remote 6-5
 - table of contents 6-9
- Repository entry
 - creating 12-11
- Request broker
 - connecting to on the Web 5-18
- Requirements 3-10
- Reservation 4-5, 8-6
- Resize screen 11-9
- Resize widget 12-10, 16-4

S

- Save command 9-6
- Screen entry 13-6
- Screen entry procedure 14-3
- Screen wizard 7-1
 - and additional tables 15-1
 - generated push buttons 9-4
 - specifying service routine name in 7-9
- Select command 9-5
- SELECT expression 16-4
- Select Service property 11-7
- Selection screen 15-5, 15-14
 - testing 15-9
- Selection service container 15-1
- send_data procedure
 - invoking by double-click event 14-2
 - invoking from push button 12-8
 - listStep 13-4
- send_ord.jsp 16-20
- Server
 - activating 2-6
 - configuring 2-1
 - development access 2-5
 - instantiations of 2-3, 2-5
 - shutdown 1-13, 1-26
 - standard 2-2
 - unable to activate B-2
 - Web application 5-1
- Server environment 1-4, 1-16
 - defining 1-5, 1-17
- Service
 - advertising 4-5, 8-6
 - creating 8-4
 - defining 4-3
- Service container
 - appearance of 7-14
 - editing 10-8
- Service name 4-4, 8-4, 11-7
- Service operations 7-9
- Service properties 11-7

Setup errors [B-2](#)

setup.sh

 copying [1-3](#)

 editing [1-3](#)

Size to Contents property [12-4](#)

SMDICNAME [6-5](#), [7-2](#)

SMTERM [1-3](#)

SQL generation [16-10](#)

Standard server [2-2](#)

 properties of [2-3](#)

Styles [9-5](#)

 description [3-10](#)

T

Table of contents

 of repository [6-9](#)

Table view widget [11-5](#), [13-16](#)

Table views

 adding widget to [16-9](#)

 selecting members of [16-6](#)

Test

 connection [4-6](#), [5-22](#)

 screens [9-1](#)

 validation [16-18](#)

Test mode

 exiting [9-8](#)

Three-tier architecture [3-4](#)

tm_events_clt procedure [13-17](#), [16-11](#), [16-12](#)

tm_events_svr procedure

 on dstord service container [13-12](#)

Toolbar

 Tool box

 Create [12-9](#)

Transaction manager [13-23](#)

 description [3-14](#)

Transport Methods [4-4](#)

Tutorial

 configuring the [A-1](#)

 tutorial directory

 contents [1-5](#), [1-17](#)

Tutorial requirements [3-10](#)

TUXEDO

 features [3-17](#)

Two-tier architecture [3-3](#)

U

ULOG file [B-1](#)

UNIX client

 setting up [3-2](#)

upd_order_total [16-16](#)

Update Service property [11-7](#)

Use In Select property [16-4](#)

Use In Where property [10-6](#)

V

valid_item_total [16-15](#), [16-16](#)

Validation Func property [16-15](#)

Validation service [15-9](#), [15-12](#)

Validation Service property [15-1](#)

vidsales database [6-6](#)

 importing [6-8](#)

View command [9-4](#)

Virtual field [16-1](#)

 including in SQL generation [16-6](#)

W

Web application architecture [3-5](#)

 in the tutorial [A-1](#)

Web application server [5-1](#)

 errors [B-1](#)

 shutting down [5-23](#)

 unable to start [B-3](#)

web_shutdown [5-18](#)

web_startup [5-18](#)

Widget List [11-5](#), [16-6](#)

Widget Type property [12-4](#)

Widgets

 copying [11-11](#)

determining dominant 11-12
moving 11-10
naming 11-14