TABLE OF CONTENTS

Contents:

Panther

Getting Started 2-Tier

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.

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

Contents:

About This Document

Introducing Panther

Introducing the Tutorial

Module 1: Setting Up

1. Setting Up the Client

2. Setting Up the Web Application Server

Module 2: Creating and Testing Screens

3. Creating a Repository

4. Using the Screen Wizard

5. Testing Screens

6. Setting Properties to Query the Database

Module 3: Connecting the Screens

7. Enhancing the Screen

8. Inheriting from the Repository

9. Writing and Executing JPL

10. Customizing Screen Behavior

Module 4: Extending the Application

11. Implementing Selection Screens

12. Calculating Data from Database Values

13. The Finale

A. Setting Up the Tutorial

B. Troubleshooting

Index

PREFACE

About This Document

Getting Started serves as an introduction to the Panther toolset. It offers an introduction to the software and to the development process. It describes, in step-by-step instructions, the features of the Panther development environment for building a two-tier application. It also includes directions for setting up clients and web application servers 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.
italics	Indicates emphasis or book titles.
UPPERCASE TEXT	Indicates Panther logical keys. *Example*: XMIT
boldface text	Indicates terms defined in the glossary.

Convention	Item

`monospace text`	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.
	Examples:
	`#include <smdefs.h>`
	`chmod u+w *`
	`/usr/prolifics`
	`prolifics.ini`

`monospace italic text`	Identifies variables in code representing the information you supply. *Example*: `String` *`expr`*

`MONOSPACE UPPERCASE TEXT`	Indicates environment variables, logical operators, SQL keywords, mnemonics, or Panther constants.
	Examples:
	`CLASSPATH`
	`OR`

`{ }`	Indicates a set of choices in a syntax line. One of the items should be selected. The braces themselves should never be typed.
`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `formlib [-v]` *`library-name`* `[`*`file-list`*`]...`
`...`	Indicates one of the following in a command line: ■ 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 The ellipsis itself should never be typed. *Example*: `formlib [-v]` *`library-name`* `[`*`file-list`*`]...`

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.

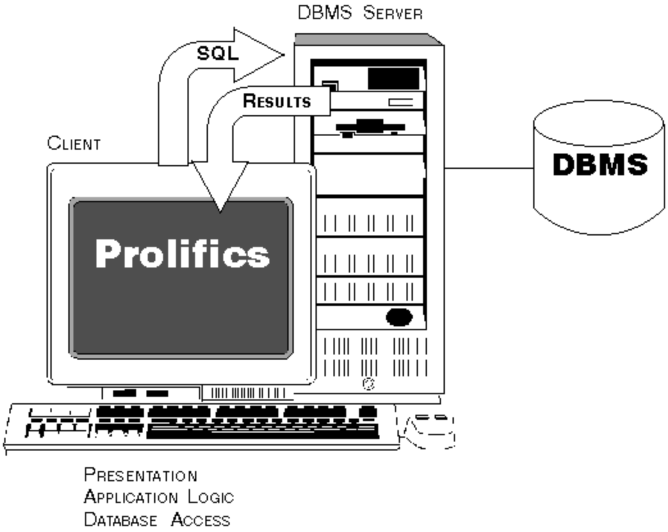


Figure 1 In two-tier architecture, each client has direct connection to the database server.

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.

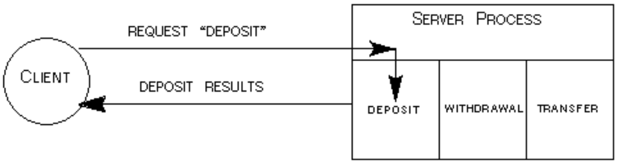


Figure 2 The client requests a service and the appropriate server responds.

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.

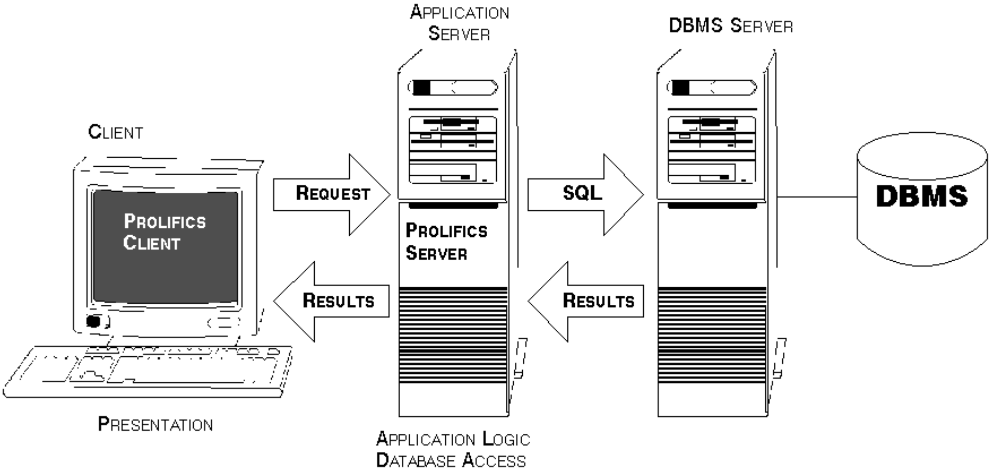


Figure 3 Three-tier clients have a connection to the database by way of the Panther application server.

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.

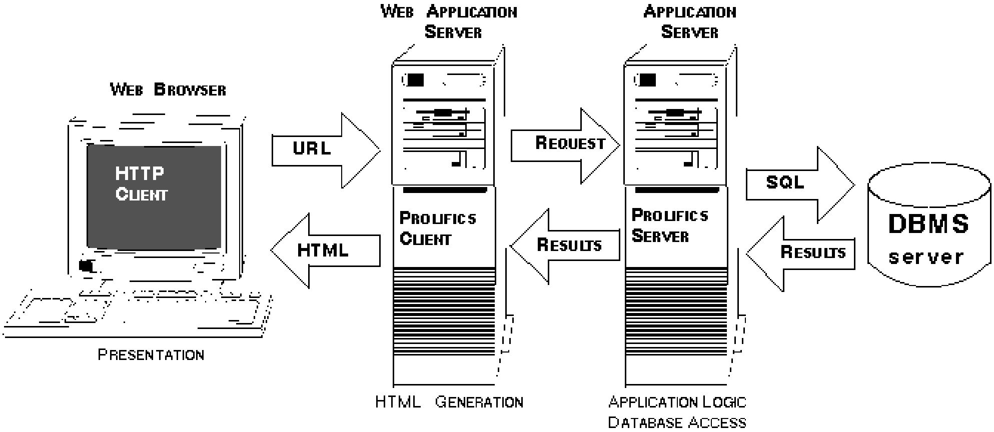


Figure 4 In Panther web applications, the web application server generates HTML for your web client screens.

Product Components

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

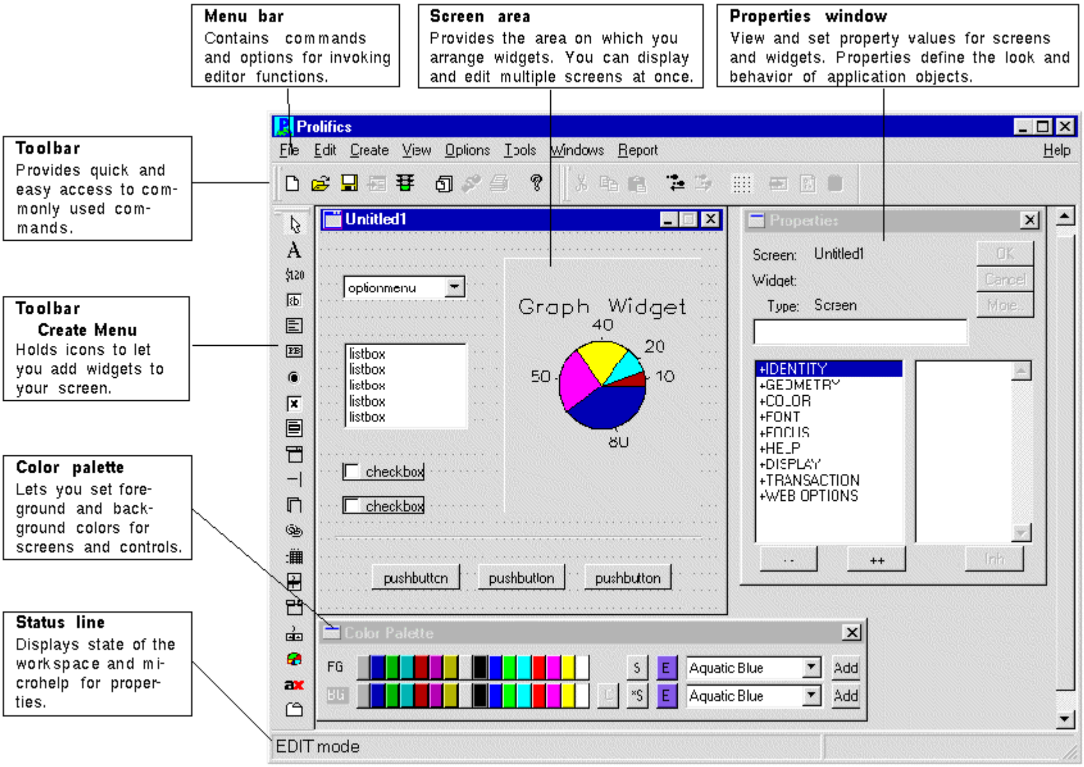
- Panther development tools:
  - Editor—A graphical environment for creating screens, reports, and service components, using widgets such as push buttons and data entry fields. Wizards are available to guide you through the process of creating screens or reports that access database information.
  - 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.



Menu bar
Contains commands and options for invoking editor functions.

Screen area
Provides the area on which you arrange widgets. You can display and edit multiple screens at once.

Properties window
View and set property values for screens and widgets. Properties define the look and behavior of application objects.

Toolbar
Provides quick and easy access to commonly used commands.

Toolbar
Create Menu
Holds icons to let you add widgets to your screen.

Color palette
Lets you set foreground and background colors for screens and controls.

Status line
Displays state of the workspace and microhelp for properties.

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 pulldowns and submenus) which can be attached to your screens as menu bars and/or toolbars. Pulldown 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.

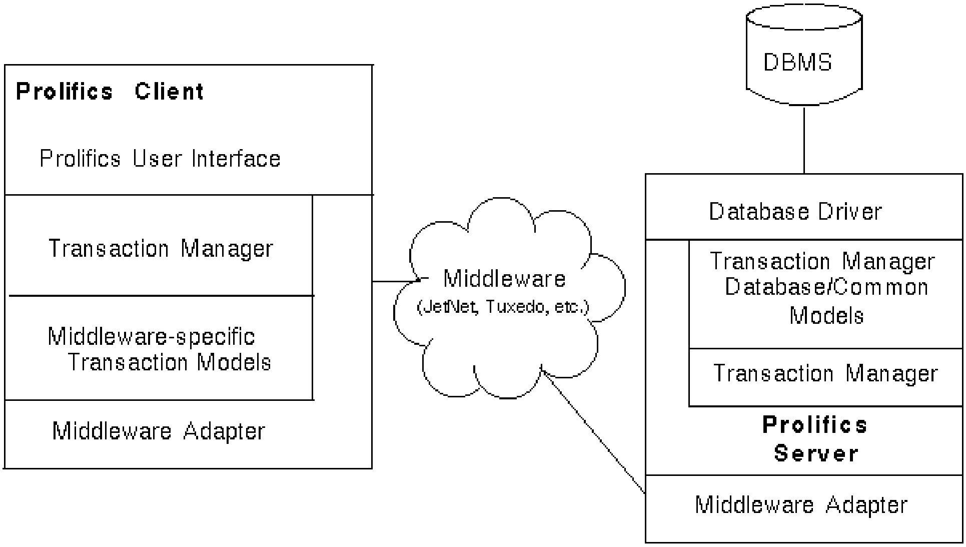


Figure 6 Application built for three-tier architecture with the screen wizard takes advantage of the transaction manager to generate service requests.

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:

- `rbboot` 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 32 bit 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 Panther COM components, 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*.

OVERVIEW

Introducing the Tutorial

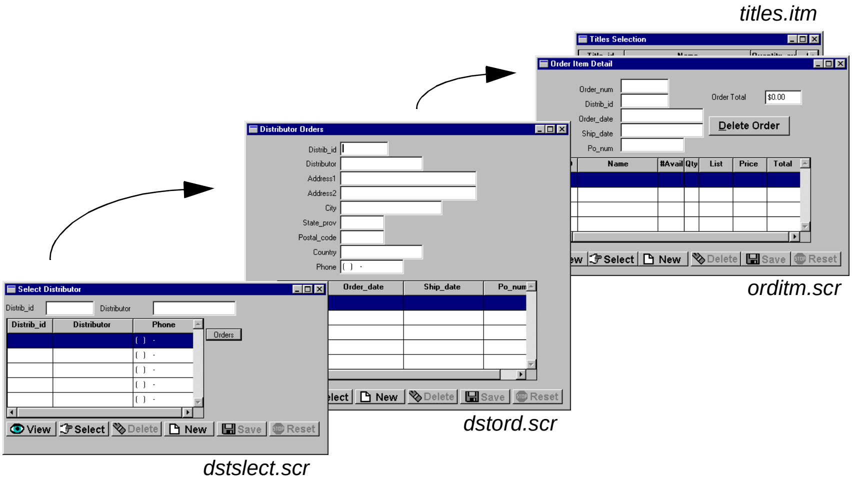
This tutorial gives an overview of two-tier and web application development, and offers a general methodology for enhanced client/server application development. Whether you are new to Panther software or are already a Panther user, a novice or experienced programmer, you can build this mini-application to learn about Panther's application development process. You can also can use the tutorial's examples, principles, code, and concepts as a template for your own applications.

About this Tutorial

The tutorial is organized into four modules plus a wrap-up and is designed to be followed in sequence. Each module begins with an introduction that describes the basic concepts to be learned in it and what you can expect to accomplish on completion. Each module has two or more lessons, for a total of 12 lessons.

You start by setting up the client/server environment. In practice, a system administrator 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 development environment.

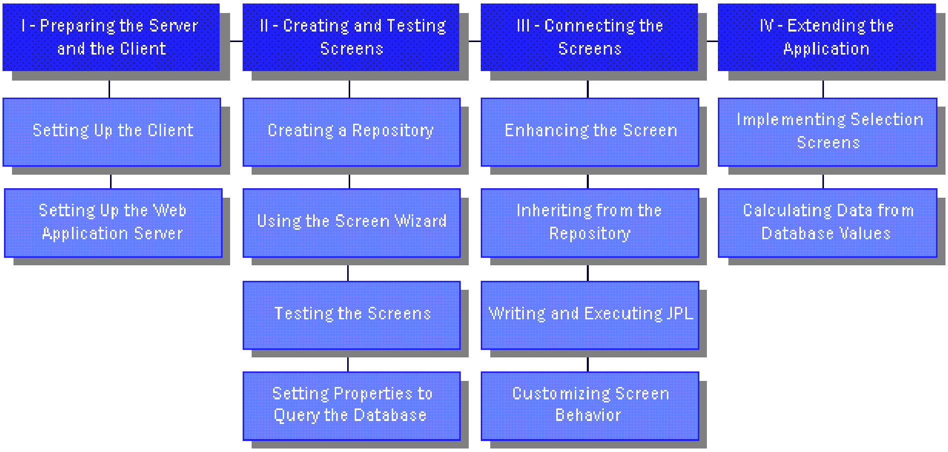
After the environment is set up, you start building an application that, on completion, consists of three screens that can be used to view and update video distributors and their orders.



The tutorial uses a JDB database, provided with the distribution, that contains distributor and order data. It also accesses a tutorial library, which contains the client screens and JPL modules that are used in the lessons.

Accessing the Tutorial Lessons

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


I - Preparing the Server and the Client
Setting Up the Client
Setting Up the Web Application Server
II - Creating and Testing Screens
Creating a Repository
Using the Screen Wizard
Testing the Screens
Setting Properties to Query the Database
III - Connecting the Screens
Enhancing the Screen
Inheriting from the Repository
Writing and Executing JPL
Customizing Screen Behavior
IV - Extending the Application
Implementing Selection Screens
Calculating Data from Database Values

About Each Module

Module 1: Setting Up

The lessons in Module 1 show how to set up the Panther environment to develop an application and to run a client-server application on the web. Typically, a system administrator sets up the web server environment for the development team. The purpose of this module is to acquaint you with this process.

The first lesson introduces you to the Panther development environment. In the second lesson, you set up a Panther web application server.

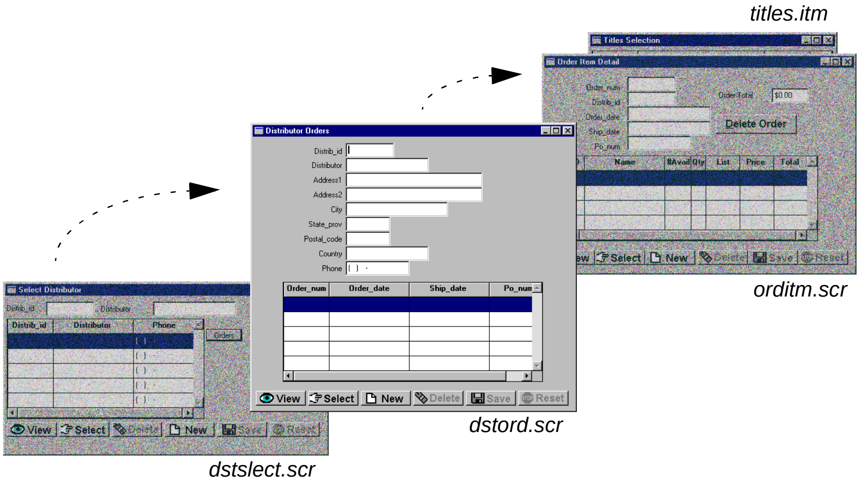
In the process of setting up the environment, you are introduced to the following concepts:

- *Setup Variables*—Panther applications depend on a number of 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.
- *Web Setup Manager*—The graphical Web Setup Manager guides you through the settings needed to configure your web application server.

Module 2: Creating and Testing Screens

Module 2 contains five lessons that lay the groundwork for your application—one that implements the functions used to view and update a list of video distributors and their orders. In these lessons, you perform these tasks:

- 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.
- 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 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.
- 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 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.

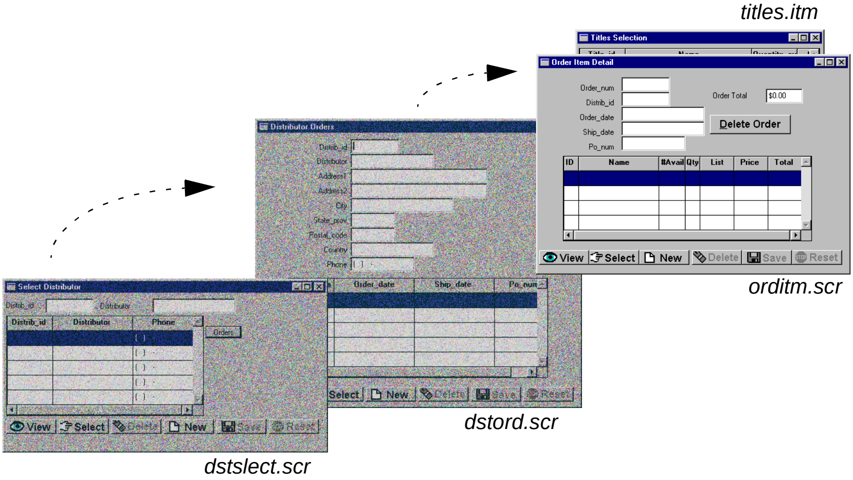
In Lesson 9, you add JPL procedures to your screens. These procedures are provided in `tutorstd.lib`, and their contents are described in the lesson. If you have additional questions about the code, you can refer to the comments in the actual JPL procedure, since the comments are not included in the 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 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 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.
- 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—each lesson builds on the results of the previous one. If you jump ahead, you might not have the data or screens you need to perform the tasks being covered in that lesson.
- When 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 web application. When you want to resume, make sure you're in the appropriate directory, set up the environment by applying the setup file (in UNIX), and restart the web application.

Before You Start

Before you start the tutorial, be aware of the following requirements:

- Know the directory and path name where Panther software is installed.
- Set up the Panther client to run either on a UNIX workstation or under Windows; instructions are given for both platforms. Illustrations represent the Windows version.
- If you run the client under Windows, make sure the Panther `Samples\Tutorial` directory is installed on your PC.
- To set up a web application server (Lesson 2), you need to have a web browser installed on your PC. You also need to find out the location of your HTTP server and its program directory (CGI, NSAPI, or ISAPI).

  If you do not have an HTTP server, you can skip Lesson 2. Follow the instructions for testing the screens in GUI.

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

Starting the Tutorial From the Beginning

If you have already started the tutorial, and wish to start it over from the first lesson, delete the files from the tutorial working directory, `proltut`, and copy the files over from the tutorial directory, `tutorial`. Both directories are in the `samples` directory of the Panther software installation.

For More Information...

As you progress through the tutorial or after you complete it, you might have questions about a specific area of Panther. Listed below are places you can look for more information.

For information on:	**Refer to:**
Using the editor and screen and report wizards	*Using the Editors*
Developing your application; using Panther components; using two- and three-tier architectures; using the JPL scripting language	*Application Development Guide*
Developing your application for the web	*Web Development Guide*

Module 1: Setting Up

LESSSON

1 Setting Up the Client

A Panther client can run either on UNIX or Windows; instructions are given for both. Illustrations in this tutorial show the Windows version.

In this lesson you learn how to:

- Start the client and specify which libraries automatically open.
- Open tutorial-specific libraries.

Setting up on Windows

In order to start the Panther client on Windows, you must provide setup information in the client's initialization file. A tutorial-specific initialization file `tutorstd.ini` is in the Windows directory. Variables in this file must be set as follows:

Note: Before beginning any of the steps below, be certain that you have installed the Panther software according to the steps listed in the *Installation Guide*.

1. Set `SMBASE` to the full path name of your client's Panther installation on Windows. For example:

   Windows `SMBASE=C:\prolifics\panther`

2. The client initialization file specifies through the application variable `SMFLIBS` which libraries open automatically when Panther (or any of its editors) starts up.

To run this tutorial as a web application, all screens that you design must be stored in a library located in the web application directory (refer to page 2-10, “General environment settings”).

In order to open the client library when the editor starts up, set `SMFLIBS` to its full path name:

```
SMFLIBS=ApplicationDirectory\client.lib
```

Note: If the web application directory is on a remote host, your client must have access to that host's file system.

3 Skip to step 9 in “Start up the Panther editor.”

Setting up on UNIX

If you are setting up the client on UNIX, it is assumed that you are running under either the Korn or Bourne shell. Before setting up, you need to know the following:

- The location of your Panther 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

This tutorial recommends that you create a local directory under your home directory and change to it. You copy all necessary files to the new directory and perform all tasks in it. The tutorial uses this directory as the Panther application directory; in practice, the application directory is located in a central location that is accessible to the development team.

Note: Before beginning any of the steps below, be certain that you have installed the Panther software according to the steps listed in the *Installation Guide*.

1 Log onto your UNIX 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 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 PantherInstallDir/config/setup.sh .
```

where *PantherInstallDir* is the full path name to the Panther installation (`/usr/prolifics` by default).

Edit the environment setup file

You set the Panther environment on UNIX with the `setup.sh` file. Edit your copy of this file to include information needed by the Panther software 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 software is installed at another location, then set `SMBASE` to the full path name of the correct directory (as noted above).
- 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 above).
- 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.
- Add `SMFLIBS` and export it:

```
SMFLIBS=client.lib
export SMFLIBS
```

6. Save and close the `setup.sh` file.

7. At the command line, type:

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

   This applies the settings in `setup.sh`.

8. Copy all files from the Panther tutorial directory to your application directory (`proltut`):

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

   This copies the libraries (`client.lib, tutorstd.lib`) and the sample database to the `proltut` directory.

Restarting the tutorial

Each time you log onto your UNIX machine, you must run `setup.sh` again before running the tutorial:

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

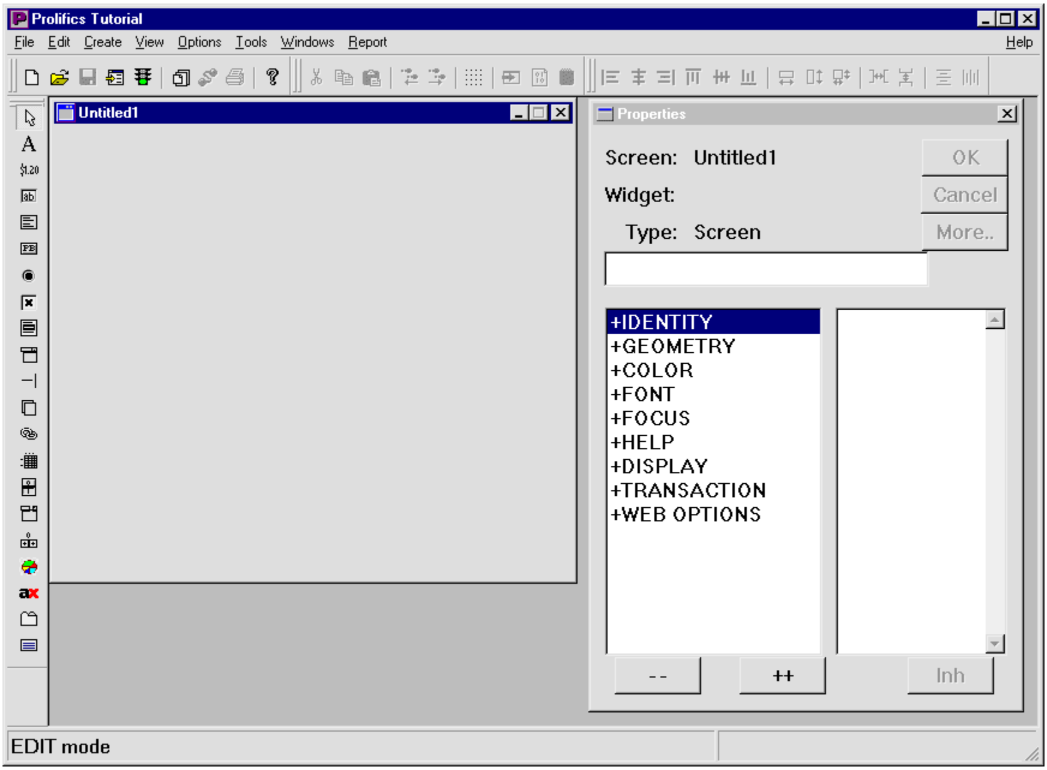
Start up the Panther editor

After you provide the required setup information, you can start the Panther editor and begin building your application's components.

9. Depending on whether you invoke the Panther client from Windows or UNIX, use one of these procedures:

   - *Windows*—Choose Tutorial from the Start menu or choose the Tutorial icon.
   - *UNIX* – Type `prodev` & at the command line.

   The editor workspace opens:

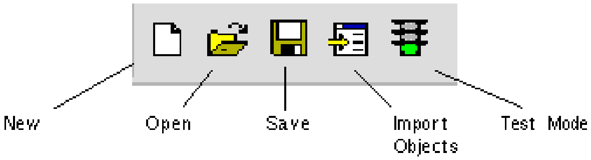


Screen: Untitled1

Widget:

Type: Screen

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.

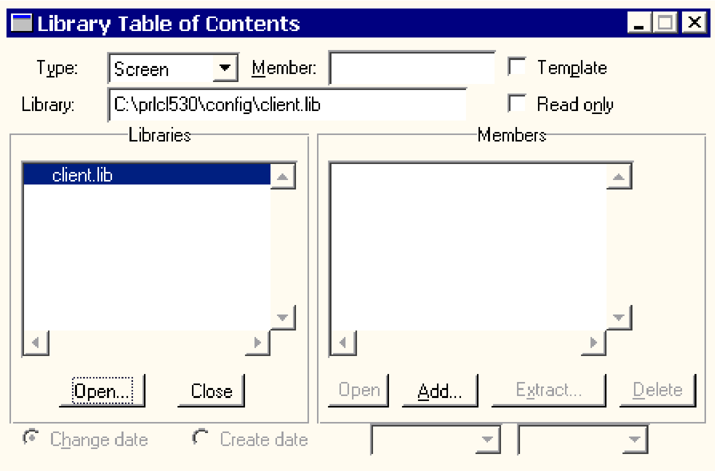
You can also use the Panther menu bar to create menus and toolbars to attach to individual screens or to the application as a whole.

Open a library and access its members

A tutorial library is provided with the Panther installation and is stored locally on your workstation or PC.

10 Choose View→Library TOC.

The Library Table of Contents opens:



Library Table of Contents

Type: Screen

Member:

☐ Template

Library: C:\prlcl530\config\client.lib

☐ Read only

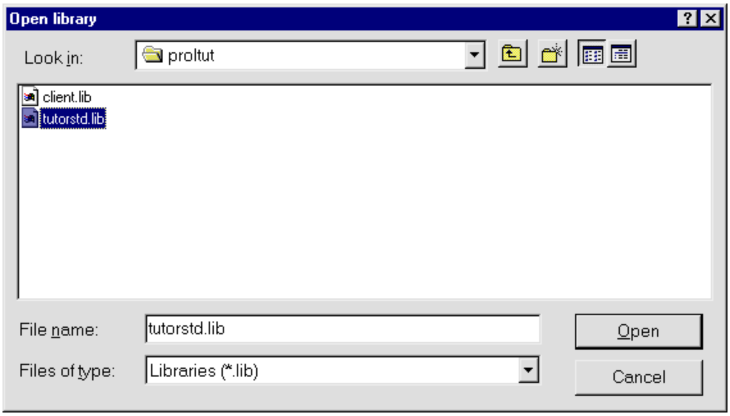
Libraries

client.lib

Members

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

The Open Library dialog box opens:



12 Select `tutorstd.lib` and choose Open (or OK on Motif).

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

What did you do?

In this lesson, you performed these tasks:

- On Windows, edited the Panther initialization file `tutorstd.ini`. On UNIX, you applied the settings in the `setup.sh` file. In practice, you edit all of the initialization files provided with your installation. You also started the editor and opened a library.

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 which libraries to open on startup.

- During development, you save screens to local libraries. However, to make them available to the web application server, you must save them to the appropriate server library.

LESSSON

2 Setting Up the Web Application Server

This lesson shows you how to set up a Panther web application server and test the server on 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, does any processing on the screen, generates the HTML to display the screen, and returns the HTML to your HTTP server, which transmits the HTML back to the browser that requested it. For more information, refer to the *Web Developerment Guide*.

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.

In this lesson you learn how to:

- Use the Web Setup Manager to create the web application initialization file and start up the server.
- Include a JPL module that handles server startup and shutdown.
- Start up the web application server and test the connection.

Most of this lesson assumes that:

- You can access the Panther web application server from your development environment.
- You are testing the screen on a graphical web browser.
- You are using the CGI version of the Panther web program.

Before starting this lesson

Before you begin this lesson, determine the following:

- The location of your Panther web application server:
  - 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`.
- That the Panther Web Setup Manager is installed on your HTTP server and ready to run via a web browser (if it is not, refer to the *Installation*). The default URL location is:

  ```
  http://serverName/programDirectory/websetup
  ```

Make a note of its URL:

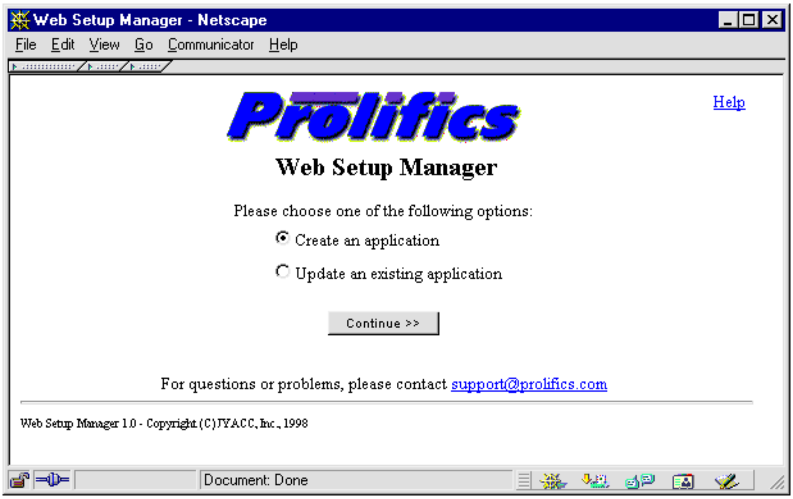
Start the Web Setup Manager

Each Panther web application must have an initialization file, which contains configuration settings for the application. The Web Setup Manager will walk you through the configuration settings and create or update the initialization file accordingly. The initialization file will have the same name as your application, for example `tut_webapp.ini`. The Web Setup Manager also copies over the executables and starts the application when it first creates the initialization file.

1 In your web browser type the URL of the program directory on your HTTP server and run websetup.

(*UNIX server*) `http://`*hostname*`/`*ProgramDiectory*`/websetup`

(*Windows server*) `http://`*hostname*`/`*ProgramDirectory*`/websetup.exe`


Web Setup Manager - Netscape
File Edit View Go Communicator Help
Prolifics
Help
Continue >>
Document: Done

Web Setup Manager

Please choose one of the following options:

● Create an application

○ Update an existing application

For questions or problems, please contact support@prolifics.com

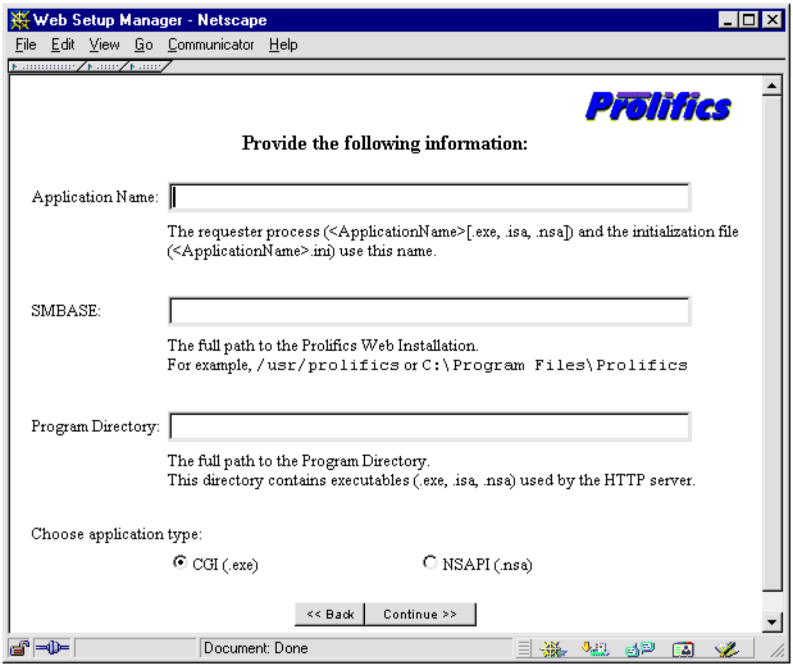
Web Setup Manager 1.0 - Copyright (C) JYACC, Inc., 1998

2 Check that the Create an Application radio button is selected.

3 Choose Continue to set up your application directory and its settings.

Enter the program locations

You assign each web application a unique name which is used to name the application's `CGI/ISAPI/NSAPI` program and the application's initialization (`.ini`) file. The requester executable that is installed as part of the Panther web application server is named `proweb`. This requester executable is copied to have the same name as your web application.


Web Setup Manager - Netscape
File Edit View Go Communicator Help
Prolifics
<< Back
Continue >>
Document: Done

Provide the following information:

Application Name:

The requester process (<ApplicationName>[.exe, .isa, .nsa]) and the initialization file (<ApplicationName>.ini) use this name.

SMBASE:

The full path to the Prolifics Web Installation.
For example, `/usr/prolifics` or `C:\Program Files\Prolifics`

Program Directory:

The full path to the Program Directory.
This directory contains executables (.exe, .isa, .nsa) used by the HTTP server.

Choose application type:

● CGI (.exe) ○ NSAPI (.nsa)

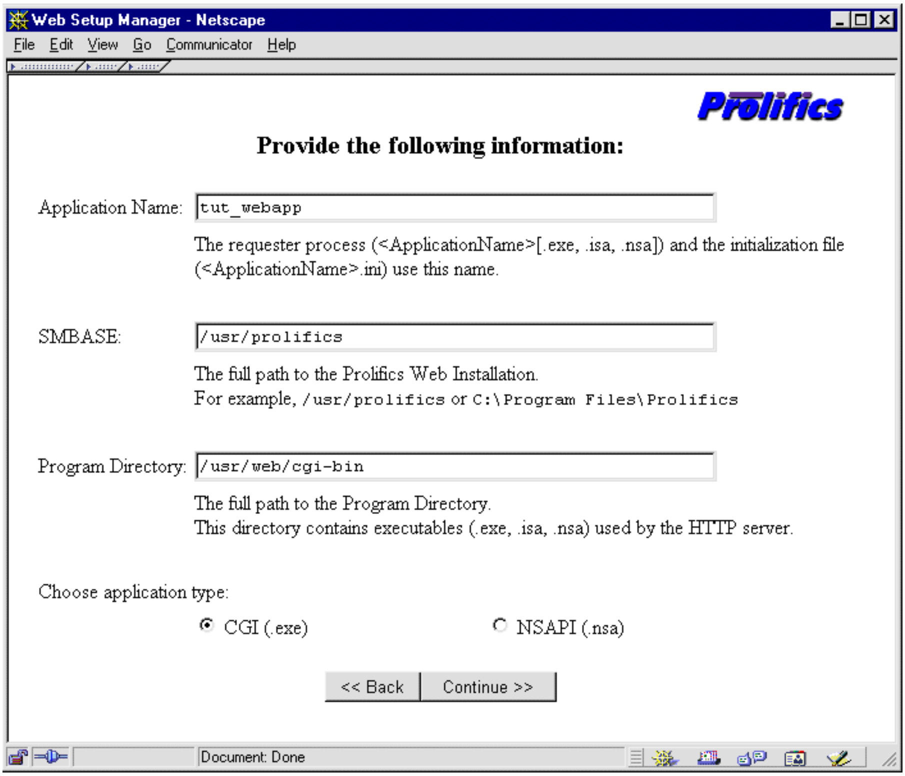
4 For Application Name, type the unique application name you are using to identify your web application, for example, `tut_webapp`.

5 Set `SMBASE` to the full path name of your Panther web installation directory. (Refer to page 2-2, “Before starting this lesson.”)

6 Enter the full path to the program directory that contains the executables used by the HTTP server. The program directory may contain CGI (`.exe`), NSAPI (`.nsa`), or ISAPI (`.isa`) executables depending on the protocols used by your HTTP server. (Refer to page 2-2, “Before starting this lesson.”)

Note: The program directory is determined by your HTTP server. If you are not sure of this directory, or if you run into permission problems accessing it, check with your system administrator. It should be a directory path, not a browser URL.

7 For application type, choose CGI by selecting the radio button.


Web Setup Manager - Netscape
File Edit View Go Communicator Help
Prolifics

Provide the following information:

Application Name: tut_webapp

The requester process (<ApplicationName>[.exe, .isa, .nsa]) and the initialization file (<ApplicationName>.ini) use this name.

SMBASE: /usr/prolifics

The full path to the Prolifics Web Installation.
For example, /usr/prolifics or C:\Program Files\Prolifics

Program Directory: /usr/web/cgi-bin

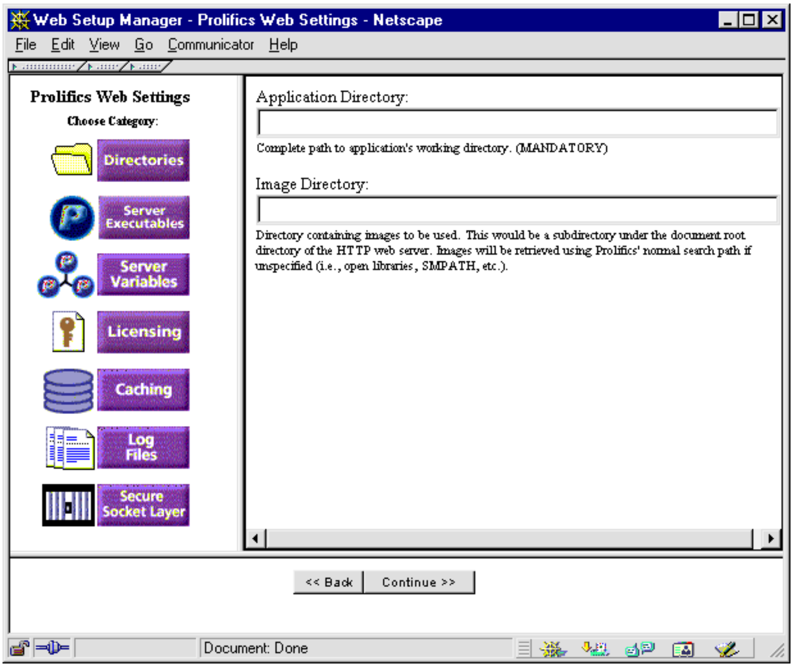
The full path to the Program Directory.
This directory contains executables (.exe, .isa, .nsa) used by the HTTP server.

Choose application type:

● CGI (.exe) ○ NSAPI (.nsa)

8 Choose Continue.

The Web Setup Manager will confirm that both an `.ini` file and an executable specific to your application (for example, `tut_webapp.ini` and `tut_webapp` have been created.



Application Directory:

Complete path to application's working directory. (MANDATORY)

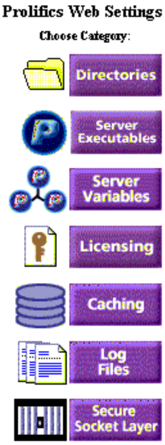
Image Directory:

Directory containing images to be used. This would be a subdirectory under the document root directory of the HTTP web server. Images will be retrieved using Prolifics' normal search path if unspecified (i.e., open libraries, SMPATH, etc.).

9 Choose OK.

Settings for your Web Application Server

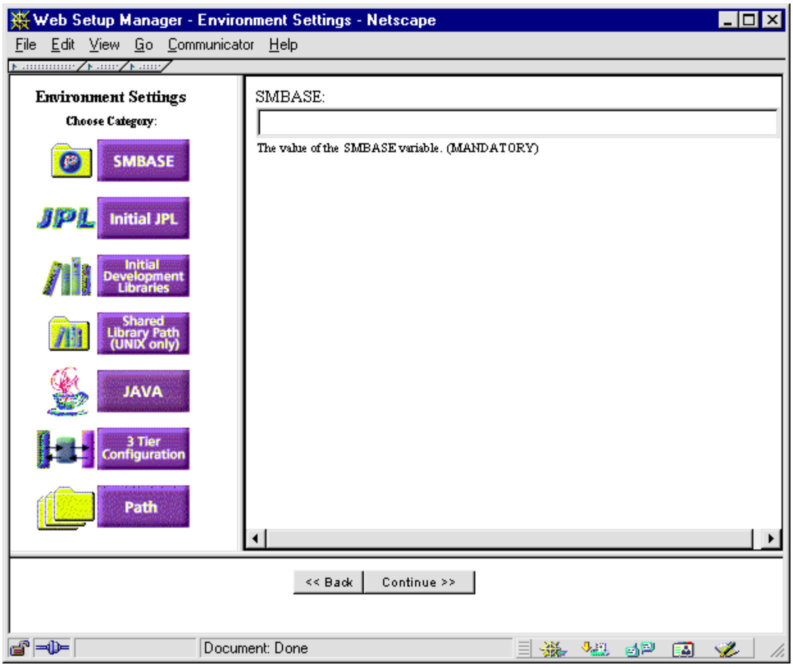
In the left-hand frame, you will see icons for the initialization settings specific to the Panther web application server. When you click on each setting, there will be an accompanying explanation of how to set that variable. Some browsers will display a default; most settings can be left at the default for now. Settings which you need to edit or double-check include:



Prolifics Web Settings

Choose Category:

10 Set the Application Directory to the full path name of your `proltut` directory.



Web Setup Manager - Environment Settings - Netscape
File Edit View Go Communicator Help
Environment Settings
Choose Category:
SMBASE
Initial JPL
Initial Development Libraries
Shared Library Path (UNIX only)
JAVA
3 Tier Configuration
Path
<< Back
Continue >>
Document: Done

SMBASE:

The value of the SMBASE variable. (MANDATORY)

11 Choose Server Executables.

12 In Dispatcher, set or check the full path name of the `dispatcher` executable, generally, the path location to the `util` directory of the web application server installation.

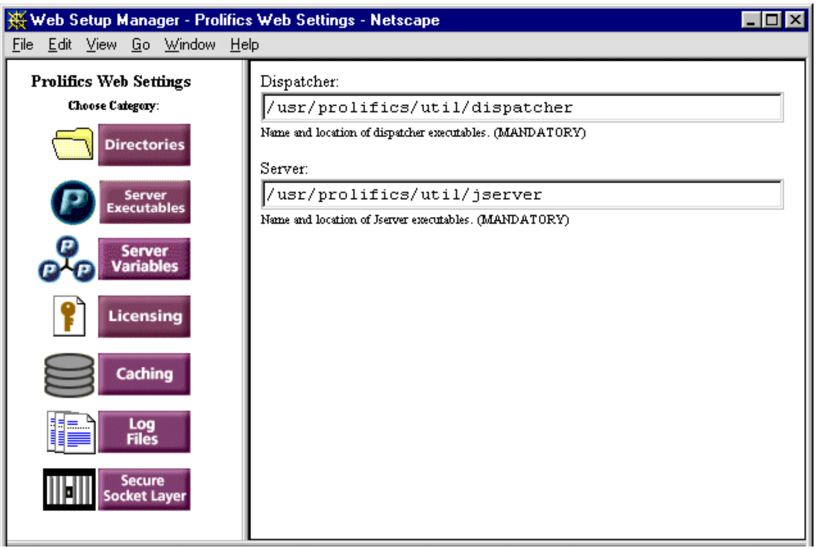
UNIX *WebInstallDirectory*`/util/dispatcher`

Windows *WebInstallDirectory*`\util\dispatcher.exe`

13 In Server, set or check the full path name of the jserver executable.

UNIX *WebInstallDirectory*`/util/jserver`

Windows *WebInstallDirectory*`\util\jserver.exe`

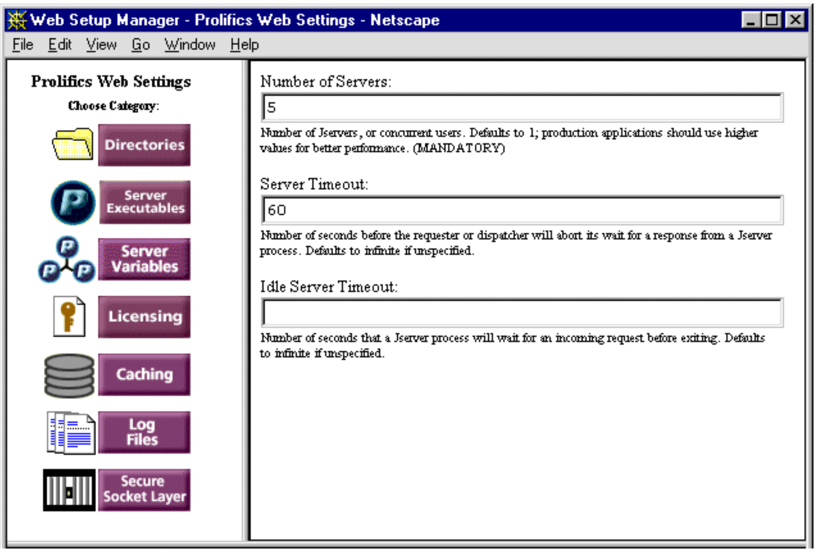


Dispatcher:
`/usr/prolifics/util/dispatcher`
Name and location of dispatcher executables. (MANDATORY)

Server:
`/usr/prolifics/util/jserver`
Name and location of Jserver executables. (MANDATORY)

14 Choose Server Variables to display the server options.

15 Set Number of Servers to 5. This is the number of concurrent users. Since this setting greatly affects performance, you must set it for each web application.



Number of Servers:

5

Number of Jservers, or concurrent users. Defaults to 1; production applications should use higher values for better performance. (MANDATORY)

Server Timeout:

60

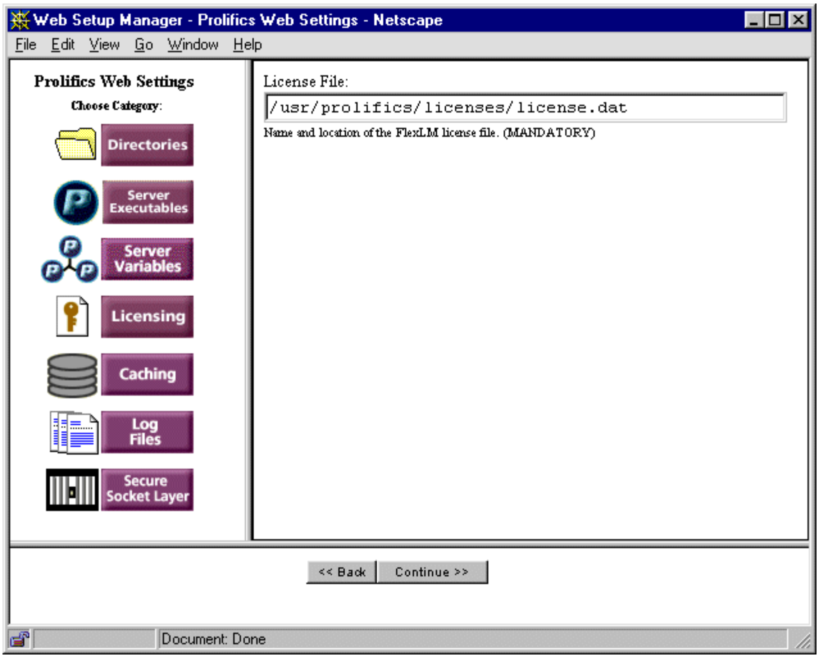
Number of seconds before the requester or dispatcher will abort its wait for a response from a Jserver process. Defaults to infinite if unspecified.

Idle Server Timeout:

Number of seconds that a Jserver process will wait for an incoming request before exiting. Defaults to infinite if unspecified.

16 Choose Licensing to display the licensing options.

17 Set or check the path of your `license.dat` file.



License File:

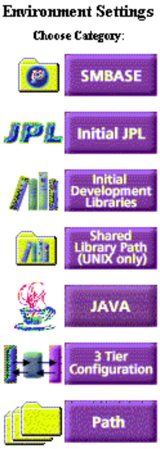
/usr/prolifics/licenses/license.dat

Name and location of the FlexLM license file. (MANDATORY)

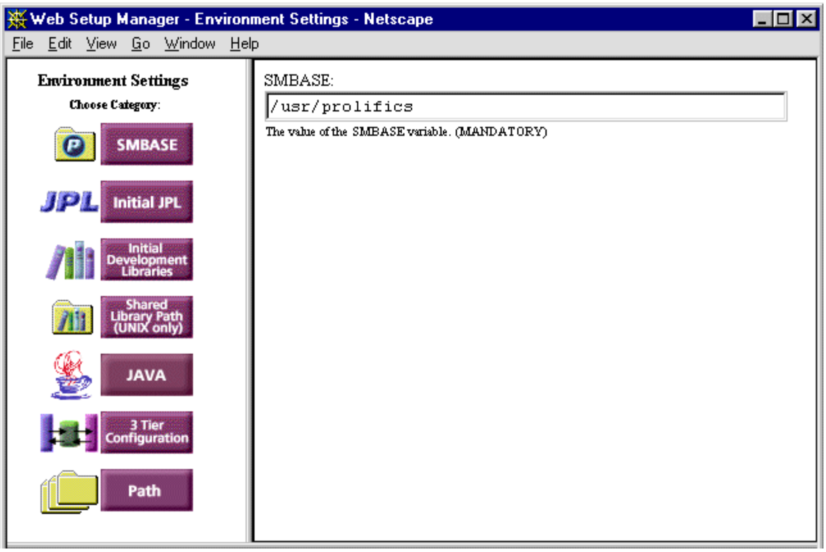
18 Choose Continue.

General environment settings

The next section allows you to create and update `tut_webapp.ini` settings relevant to the general environment. You will see a new set of icons in the left-hand frame. Step through each category, editing or double-checking settings as necessary.



19 In `SMBASE`, set or check the full path name of the Web Application Server installation directory.


Web Setup Manager - Environment Settings - Netscape
File
Edit
View
Go
Window
Help
Environment Settings
Choose Category:
SMBASE
Initial JPL
Initial Development Libraries
Shared Library Path (UNIX only)
JAVA
3 Tier Configuration
Path

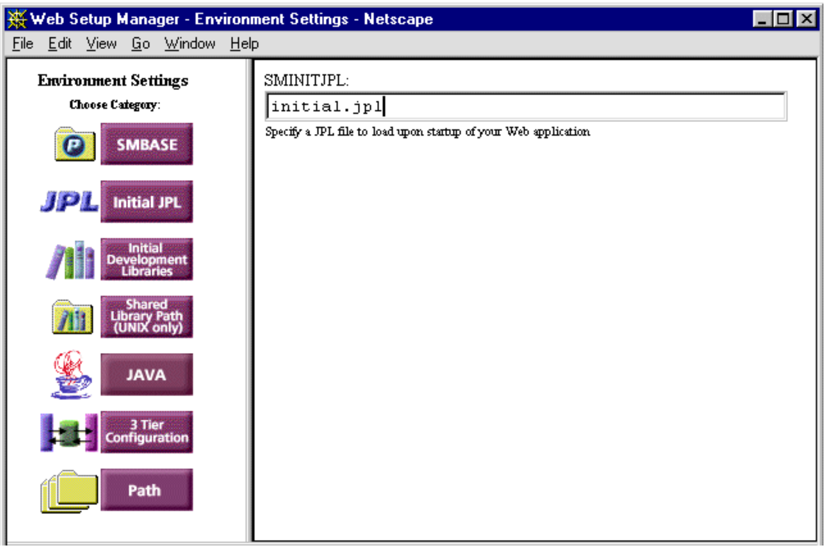
SMBASE:

/usr/prolifics

The value of the SMBASE variable. (MANDATORY)

20 Choose Initial JPL.

21 Set `SMINITJPL` to `initial.jpl`.

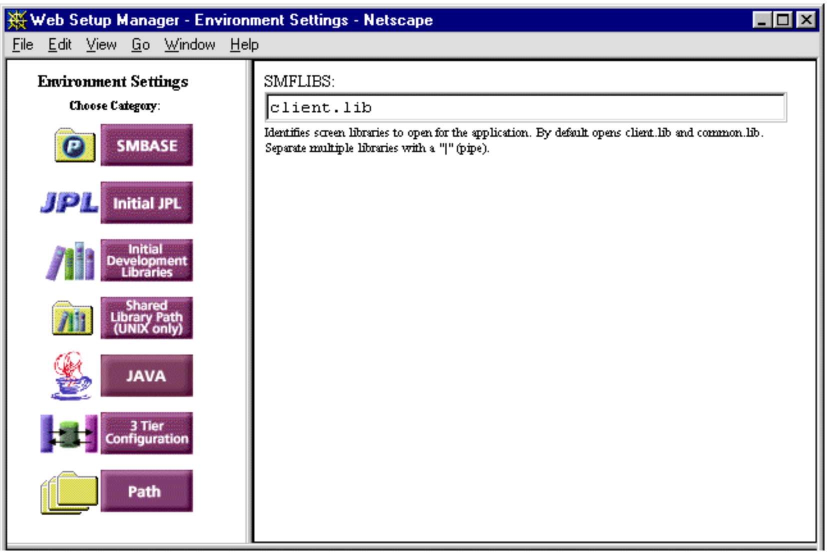

Web Setup Manager - Environment Settings - Netscape
File Edit View Go Window Help
Environment Settings
Choose Category:
SMBASE
Initial JPL
Initial Development Libraries
Shared Library Path (UNIX only)
JAVA
3 Tier Configuration
Path

SMINITJPL: initial.jpl

Specify a JPL file to load upon startup of your Web application

22 Choose Initial Development Libraries.

23 In `SMFLIBS`, set or check `client.lib`.


Web Setup Manager - Environment Settings - Netscape
File Edit View Go Window Help
Environment Settings
Choose Category:
SMBASE
Initial JPL
Initial Development Libraries
Shared Library Path (UNIX only)
JAVA
3 Tier Configuration
Path

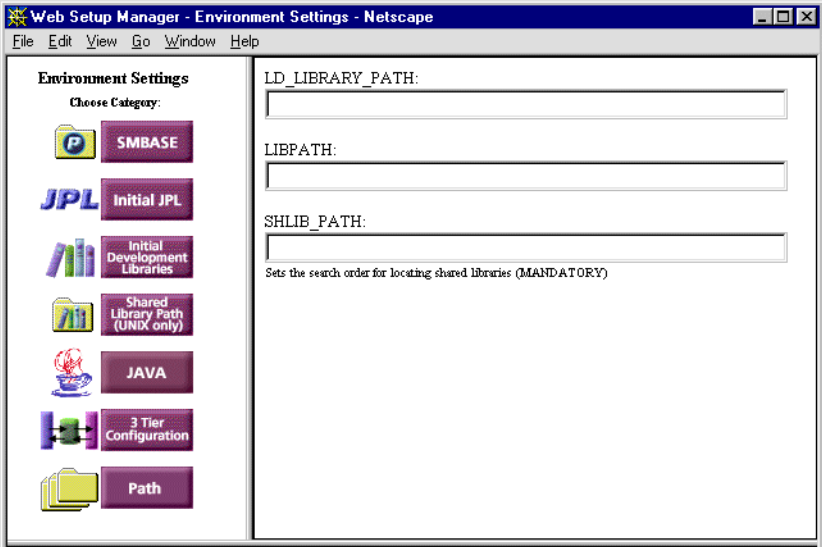
SMFLIBS: client.lib

Identifies screen libraries to open for the application. By default opens client.lib and common.lib. Separate multiple libraries with a "|" (pipe).

24 (*UNIX only*) Select Shared Library Path.

25 (*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`.


Web Setup Manager - Environment Settings - Netscape
File
Edit
View
Go
Window
Help
Environment Settings
Choose Category:
SMBASE
Initial JPL
Initial Development Libraries
Shared Library Path (UNIX only)
JAVA
3 Tier Configuration
Path

LD_LIBRARY_PATH:

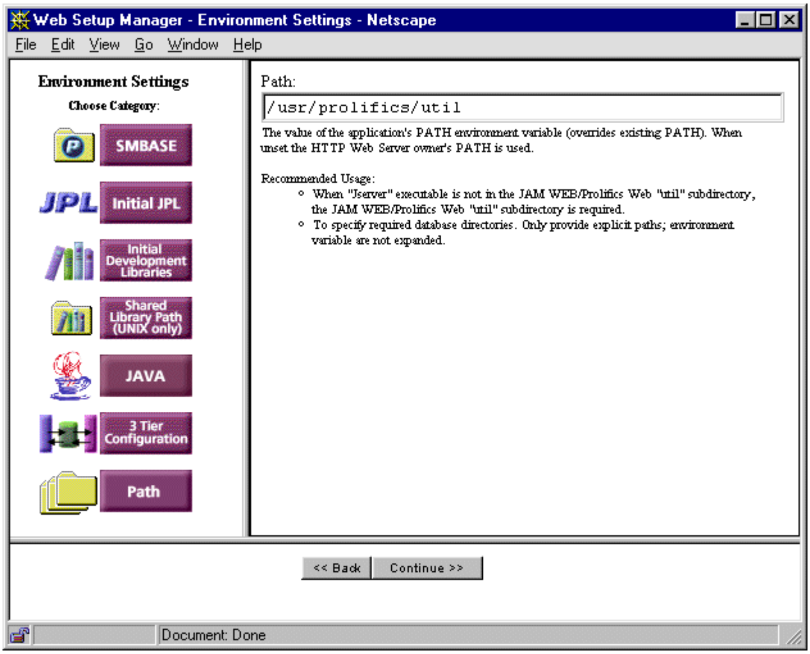
LIBPATH:

SHLIB_PATH:

Sets the search order for locating shared libraries (MANDATORY)

26 Choose Path.

27 Set Path to the full path name of `util` in the *WebInstallDir* directory.


Web Setup Manager - Environment Settings - Netscape
File Edit View Go Window Help
Environment Settings
Choose Category:
SMBASE
Initial JPL
Initial Development Libraries
Shared Library Path (UNIX only)
JAVA
3 Tier Configuration
Path
<< Back
Continue >>
Document: Done

Path:

/usr/prolifics/util

The value of the application's PATH environment variable (overrides existing PATH). When unset the HTTP Web Server owner's PATH is used.

Recommended Usage:

- When "Jserver" executable is not in the JAM WEB/Prolifics Web "util" subdirectory, the JAM WEB/Prolifics Web "util" subdirectory is required.
- To specify required database directories. Only provide explicit paths; environment variable are not expanded.

28 Choose Continue.

29 The Additional Environment Variables screen appears. Choose Continue.

30 The Web Setup Manager will give you a message that your application has been created. Choose Start Application or exit from your browser.

The Web Setup Manager has created your web application `.ini` file and copied the appropriate executables for that application.

31 If you started the application, shut it down from the command line:

UNIX *WebInstallDir*`/util/monitor -stop` *WebAppname*

Windows *WebInstallDir*`\util\monitor -stop` *WebAppname*

Add JPL routines to the client library

When a web application server starts up, it looks in its JPL module—specified by the web initialization file variable `SMINITJPL`—for the `web_startup` procedure to handle startup processing. This processing can include opening database connections and creating global variables. The module distributed for this tutorial—`in itial.jpl`—contains a `web_startup` procedure that initializes the JDB database engine and establishes a connection to the `vidsales` database:

```
proc web_startup
{
   DBMS ENGINE jdb
   DBMS DECLARE tut_conn CONNECTION FOR DATABASE "vidsales"
}
```

When the server shuts down, it looks in the same module for the `web_shutdown` procedure to perform cleanup operations such as closing the database connection. The `web_shutdown` procedure for this tutorial closes the database connection established earlier by `web_startup`:

```
proc web_shutdown
{
   DBMS CLOSE_ALL_CONNECTIONS
}
```

The distributed `tutorstd.lib` library contains a web initialization file. A copy of this module must be saved in the client library.

32 Start the Panther editor:

- Windows—Choose Tutorial from the Start menu or choose the Tutorial icon.

  For more information on setting up the client on Windows, refer to page 1-1, “Setting up on Windows.”

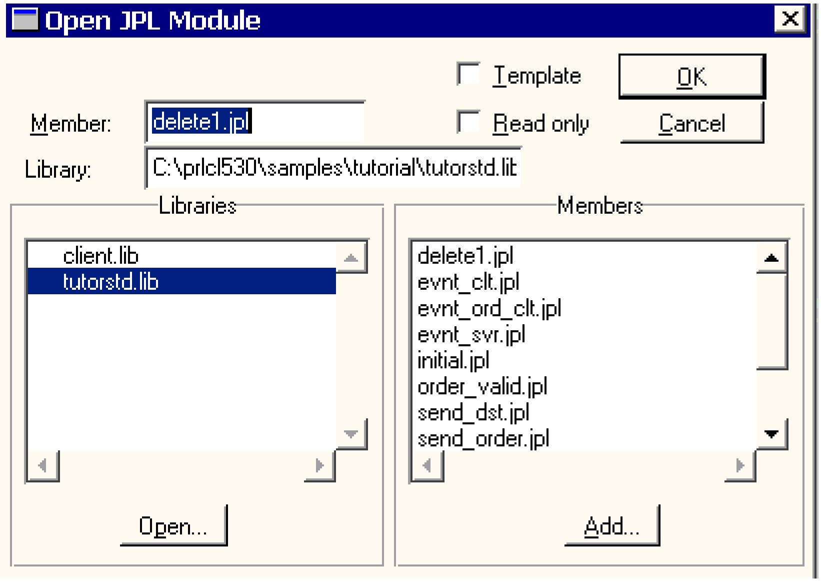
- UNIX—Type `prodev &` at the command line.

  For more information on setting up the client on UNIX, refer to page 1-2, “Setting up on UNIX.”

33 In the editor, choose File→Open→Library and select `tutorstd.lib`.

34 Choose File→Open→JPL.

The Open JPL Module dialog opens.


Open JPL Module

□ Template

□ Read only

OK

Cancel

Member: delete1.jpl

Library: C:\prlcl530\samples\tutorial\tutorstd.lib

Libraries

client.lib
tutorstd.lib

Open...

Members

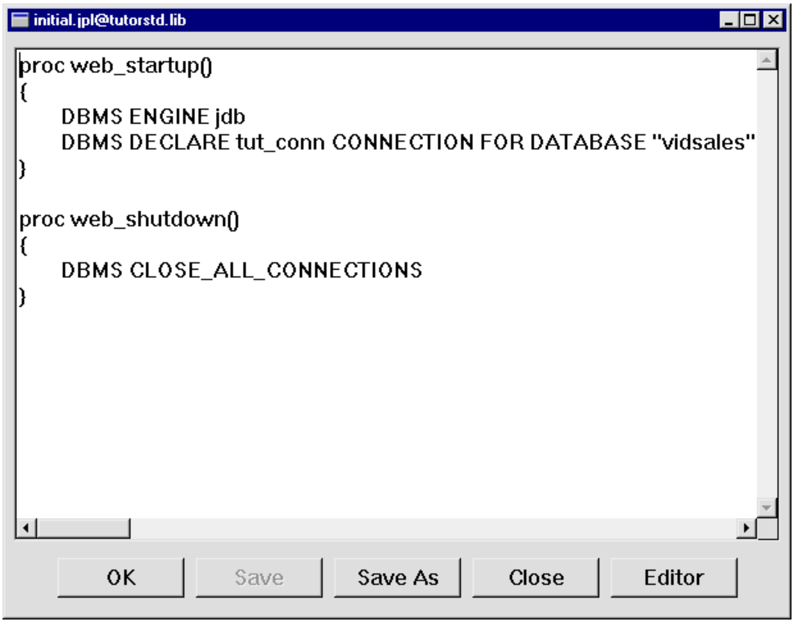
delete1.jpl
evnt_clt.jpl
evnt_ord_clt.jpl
evnt_svr.jpl
initial.jpl
order_valid.jpl
send_dst.jpl
send_order.jpl

Add...

- If `tutorstd.lib` is not open, choose Open and select it from the Open Library dialog box.

35 Select `tutorstd.lib` from the list of open libraries, and double-click on `initial.jpl` in the Members list to open it.

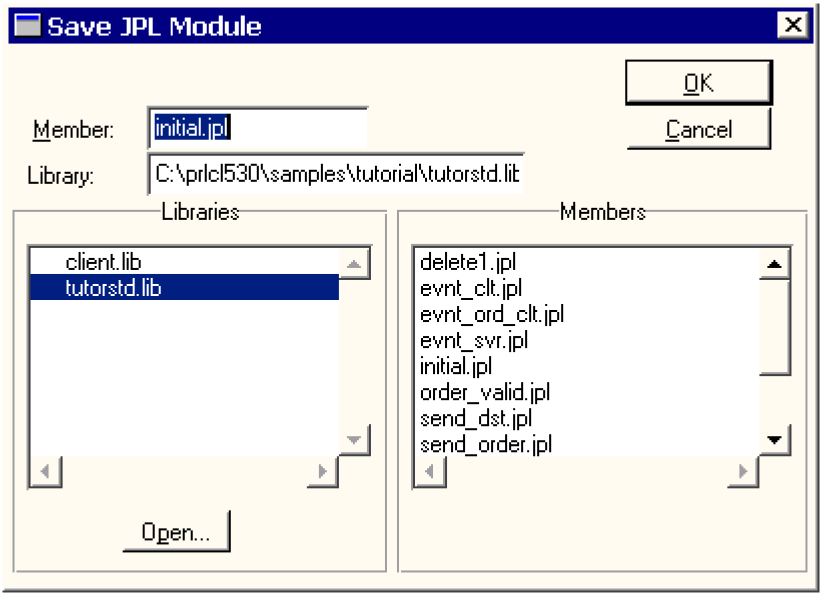
The `initial.jpl` file opens in the JPL Program Text dialog box.



```
proc web_startup()
{
	DBMS ENGINE jdb
	DBMS DECLARE tut_conn CONNECTION FOR DATABASE "vidsales"
}

proc web_shutdown()
{
	DBMS CLOSE_ALL_CONNECTIONS
}
```

36 Choose Save As. The Save JPL dialog opens.



Member: initial.jpl

Library: C:\prlcl530\samples\tutorial\tutorstd.lib

Libraries:
- client.lib
- tutorstd.lib

Members:
- delete1.jpl
- evnt_clt.jpl
- evnt_ord_clt.jpl
- evnt_svr.jpl
- initial.jpl
- order_valid.jpl
- send_dst.jpl
- send_order.jpl

37 Select `client.lib` in the Libraries list, and `initial.jpl` appears in the Member box. Choose OK.

The `initial.jpl` file is saved to `client.lib`.

38 On the JPL Window, choose Close.

39 Choose File→Exit to end your editor session.

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 9 of this tutorial. For more information on JPL commands and syntax, refer to the *Programming Guide*.

Change permissions

Make sure the application directory can support the web application server started by the httpd program. Because the user that the httpd program runs under is different from your user login, the web application server is normally not allowed to write to your directory.

Test the connection

Now you can start the web application server and test the client/server connection by opening your client screen in a web browser.

40 If your HTTP server is not running, activate it now. Also start the client's web browser on your PC if it is not already running.

41 Start the web application server from the command line:

UNIX `WebInstallDir/util/monitor -start WebAppname`

Windows `WebInstallDir\util\monitor -start WebAppname`

42 Type in the URL in your browser:

`http://hostname/cgi-bin/WebAppname`

At this point, you have started up the Panther web application server. The client screen appears.

If the phrase “No service requested!” is displayed then you have successfully configured the web application server.

If you make any changes to the web initialization (`WebAppname.ini`) file, you must stop and restart the Panther web application server to have the changes take effect. Stopping the web application server is described in the next section.

Shut down the server

After you successfully start the web application server, you can shut it down with the `monitor` utility; the lessons that immediately follow do not require its usage.

43 Shut down the web application server:

UNIX `WebInstallDir/util/monitor -stop WebAppname`

Windows `WebInstallDir\util\monitor -stop WebAppname`

What did you do?

In this lesson, you performed these tasks:

- Used the Web Setup Manager to make a copy of the web application server initialization file, gave it the same name as your web application, and edited it to reflect variable settings for your application. The Web Setup Manager also copied the web executable and started the web application server.
- 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.
- 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 CGI directory.
- Each application must have an initialization file, which contains variables determining the behavior of Panther web, the Panther environment used by the jserver program, and the database environment.
- Certain processing, such as opening a database connection 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.

OVERVIEW

Module 2: Creating and Testing Screens

LESSSON

3 Creating a Repository

When you begin to build a database application, you need to create a repository. A repository lets you take advantage of the contents of the database. You can directly, or via the screen wizard, use the database-derived objects in the repository to create screens.

In this lesson, you learn how to create a repository 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:

- Create a repository.
- Import the `vidsales` database tables into the repository.

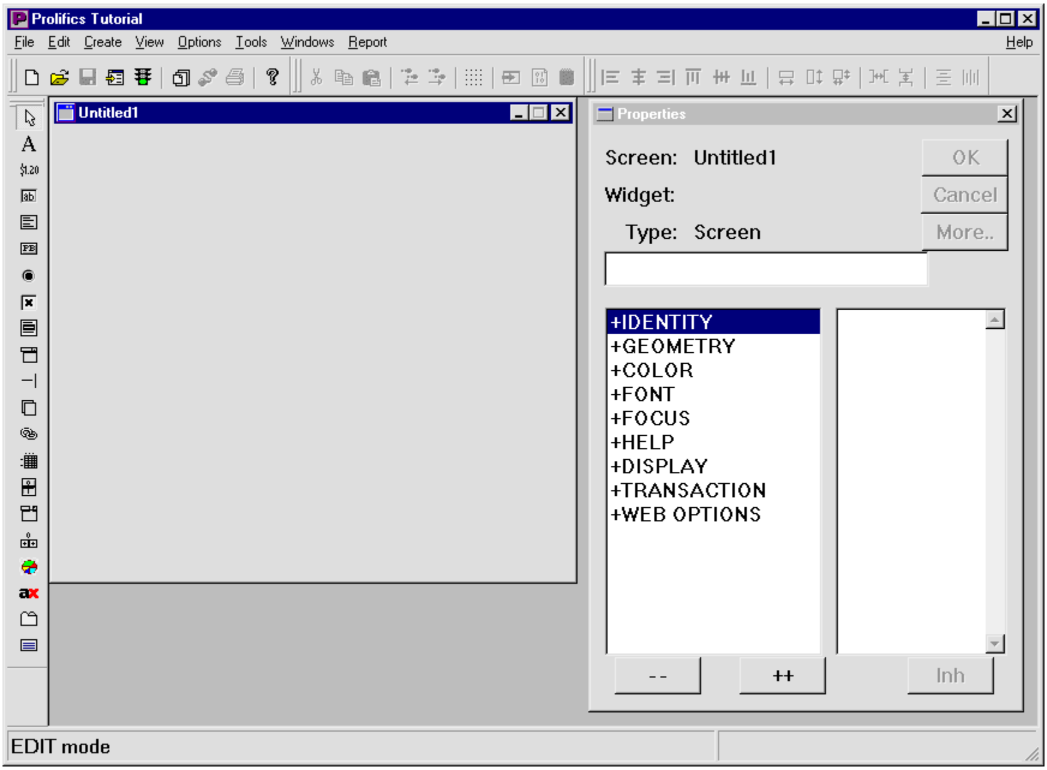
Start the editor

Depending on whether you invoke the Panther editor from Windows or UNIX, use one of these procedures:

- *Windows*—Choose Panther Client→Tutorial from the Start menu or choose the Tutorial icon.

- *UNIX*—Type `prodev &` at the command line.

The editor workspace opens.



Screen: Untitled1

Widget:

Type: Screen

+IDENTITY
+GEOMETRY
+COLOR
+FONT
+FOCUS
+HELP
+DISPLAY
+TRANSACTION
+WEB OPTIONS

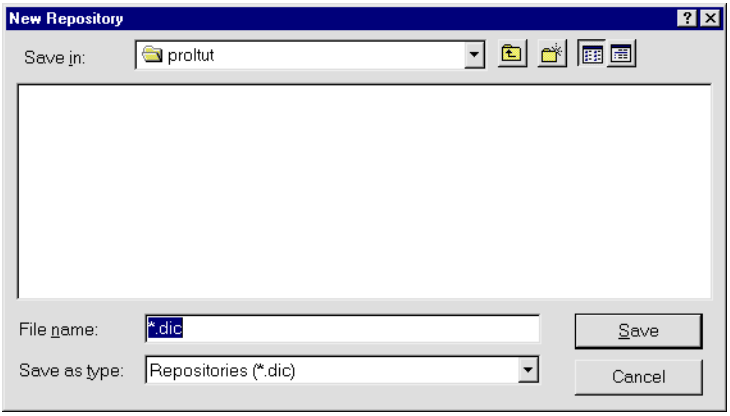
EDIT mode

Create a repository

Create a repository where you can store imported database objects.

1 Choose File→New→Repository.

The New Repository dialog opens.



2 Enter `data.dic` as the name of a repository in the File Name field.

On startup, Panther automatically opens any local repository with the name `data.dic`. To open a repository with a different name, edit your initialization file and set the application variable `SMDICNAME` to the repository's path.

3 Choose Save.

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

More About Repositories

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.

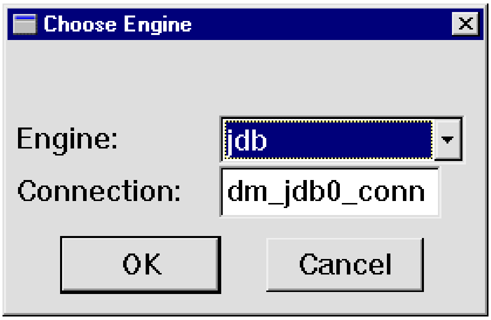
Connect to the database

In order to import database definitions, the editor must be directly connected to one of the database engines that Panther supports, including its own database engine JDB.

A Panther client can establish a connection to the database engine directly and through Panther's web application server. In this lesson, you will connect to JDB directly; in Lesson 2, you set up the web application server so it can connect to a database engine and let you test application screens from your browser.

4 Choose File→Open→Database.

The Choose Engine dialog opens.

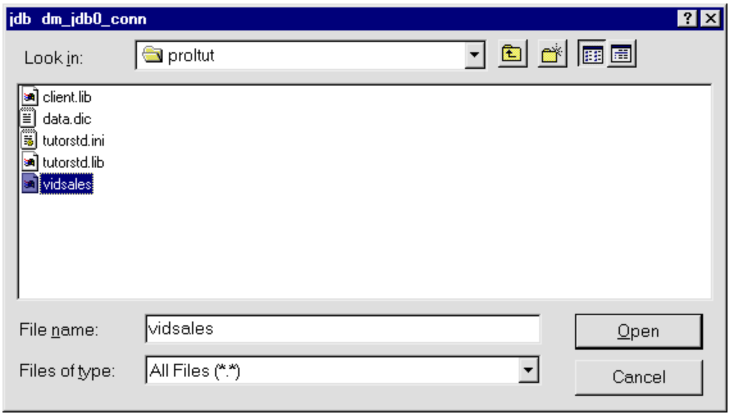


Choose Engine

Engine: jdb

Connection: dm_jdb0_conn

5 Choose OK. A database-specific dialog appears.



jdb dm_jdb0_conn

Look in: proltut

client.lib
data.dic
tutorstd.ini
tutorstd.lib
vidsales

File name: vidsales

Files of type: All Files (*.*)

6 Select or specify `vidsales` in the File Name field, then choose Open.

You are now connected to the database.

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

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.

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 the Panther editor.

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.

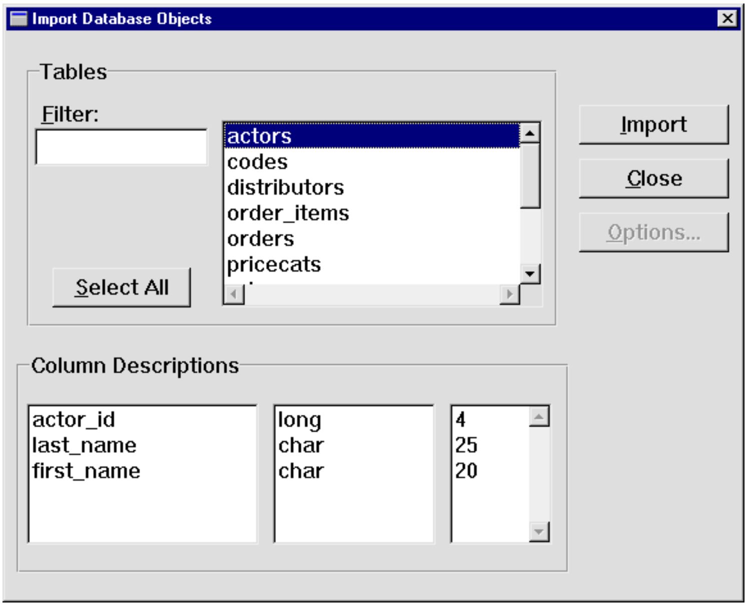
Also, if your database engine supports other database object types, such as views and synonyms, you can import those as well.

All application objects that inherit from repository objects are maintained without having to access the database or manually edit screens. Development can continue even when the database is inaccessible or unavailable.

All application objects that inherit from repository objects are maintained without having to access the database or manually edit screens. Development can continue

7 Choose Tools→Import Database Objects or ▤.

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



actors
codes
distributors
order_items
orders
pricecats

actor_id	long	4
last_name	char	25
first_name	char	20

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.

8 Choose Select All to select all of the tables in the database.

9 Choose Import.

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

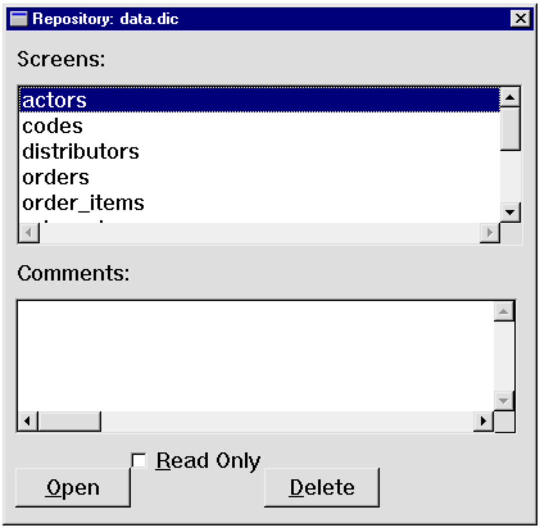
10 When the import is complete, choose Close to return to the editor.

View repository contents

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

11 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 screens in the open repository.



12 (Optional) Leave the Repository TOC open so you can easily access repository screens. 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 performed these tasks:

- Connected to the `vidsales` database so that you could import its definitions into a repository.
- Created a repository for developing your database application.
- Imported the `vidsales` database tables into the repository.
- Viewed the contents of the repository with 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 can establish a direct database connection from your workstation. In this case, you did so in order to import database definitions to a repository.
- The import process creates an entry in the open repository for each database table that you import.

LESSSON

4 Using the Screen Wizard

The screen wizard provides an easy way to create transactional database application components, including:

- Client screens for a client/server application for two-tier processing.
- Screens that use a web-compatible interface.

In this lesson, you create a master-detail client screen. The client screen lets a user 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 creates screens by using objects and database columns from the open repository (for the tutorial, the repository `data.dic`). After the screens are created, you save them to their appropriate library.

In this lesson you learn how to:

- Use Panther's screen wizard to create a client screen.
- Save wizard output to the client library.

1 If necessary, invoke the editor, and connect to the `vidsales` database.

Open the repository

To use the screen wizard, a repository that contains imported database objects must be open. If the `SMDICNAME` variable is set in Panther's Windows initialization file (`tutorstd.ini`), the repository that you created in Lesson 3 opens automatically when you invoke the editor, and you can skip to step 3; otherwise, open the repository manually (step 2).

Note: (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 If the `data.dic` repository is not open, choose File→Open→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 it.

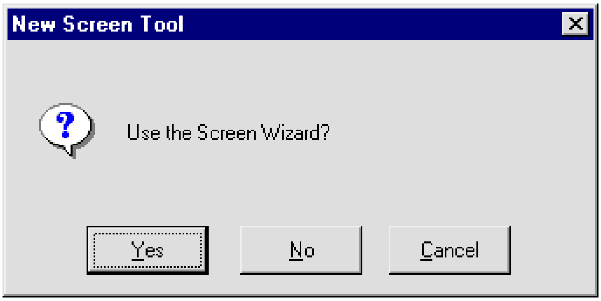
More About the Screen Wizard

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.

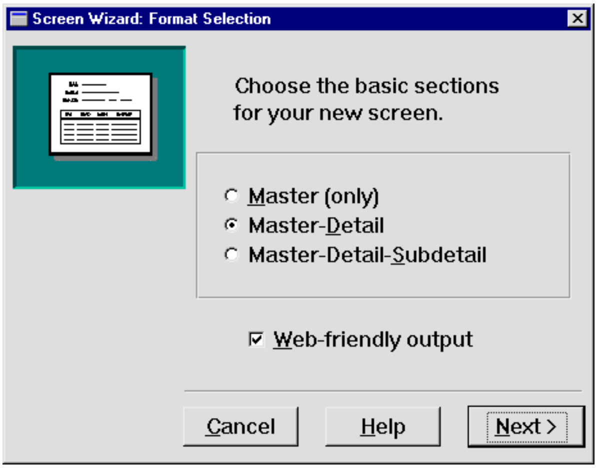


New Screen Tool

Use the Screen Wizard?

4 Choose Yes to use the screen wizard.

The Format Selection dialog opens.



Screen Wizard: Format Selection

Choose the basic sections for your new screen.

○ Master (only)
● Master-Detail
○ Master-Detail-Subdetail

☑ Web-friendly output

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.

5 Select the Master-Detail (default) option to define the sections.

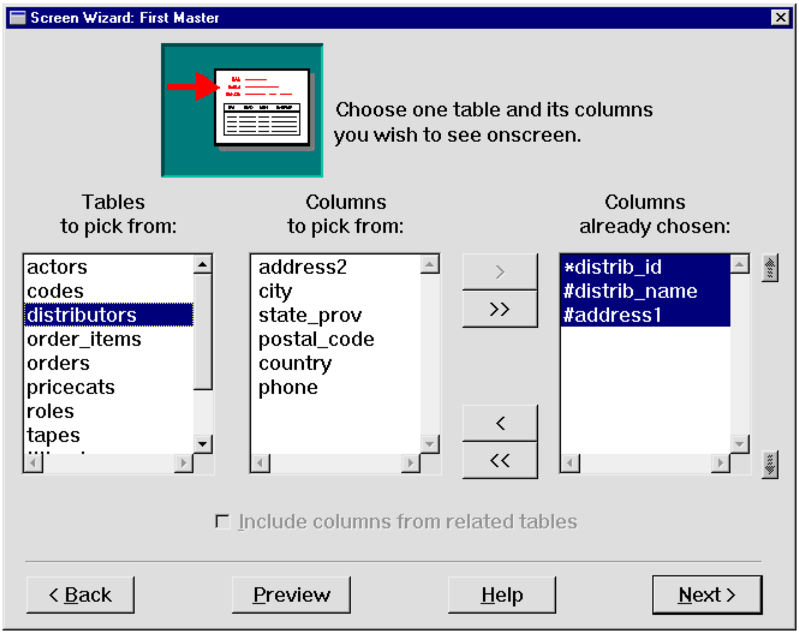
6 Leave the Web-Friendly Output check box selected.

More about Web-Friendly Output

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.

7 Choose Next.

The First Master dialog opens.



Screen Wizard: First Master

Choose one table and its columns you wish to see onscreen.

Tables to pick from:

actors
codes
distributors
order_items
orders
pricecats
roles
tapes

Columns to pick from:

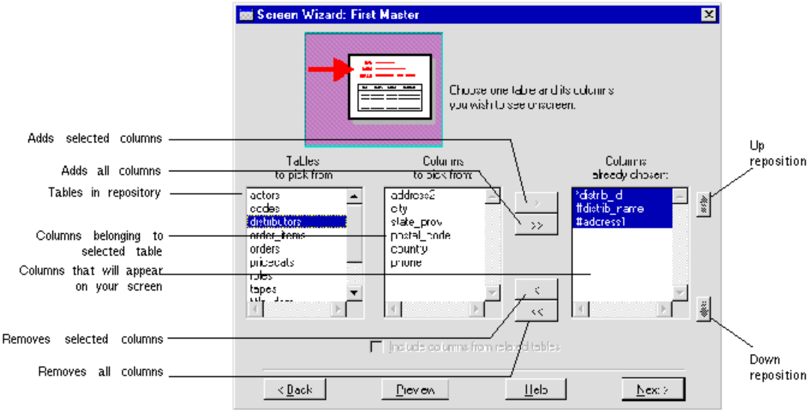
address2
city
state_prov
postal_code
country
phone

Columns already chosen:

*distrib_id
#distrib_name
#address1

☐ Include columns from related tables

The list displays 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. Also included are the “not null” keys (indicated with the pound sign).

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.

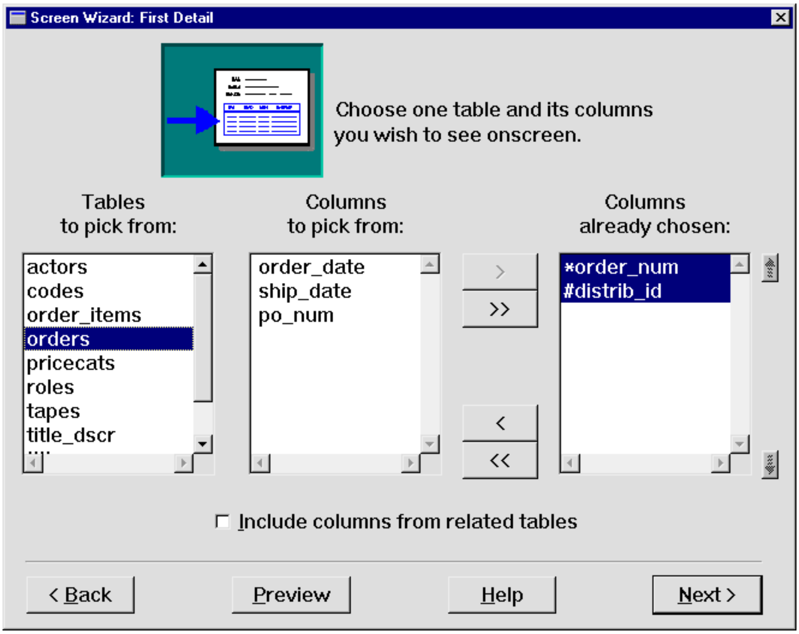
10 Choose Next.

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.



Screen Wizard: First Detail

Choose one table and its columns you wish to see onscreen.

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 client screen to display all orders associated with a given distributor.

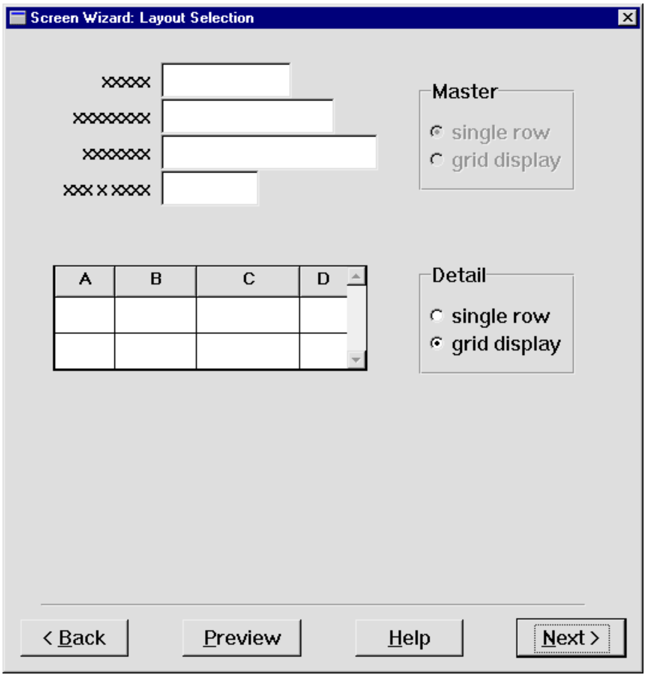
More About Links

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.



Master
● single row
○ grid display

Detail
○ single row
● grid display

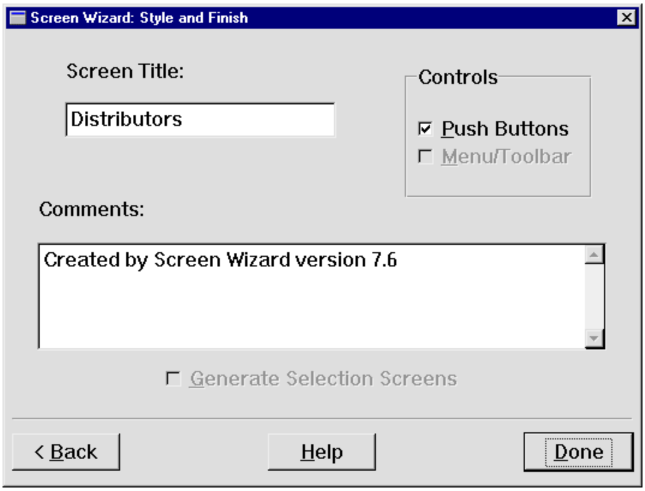
More About Screen Wizard Layout Options

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 Style and Finish dialog opens.



Screen Wizard: Style and Finish

Screen Title: Distributors

Controls

☑ Push Buttons

□ Menu/Toolbar

Comments: Created by Screen Wizard version 7.6

□ Generate Selection Screens

Customize the output screen

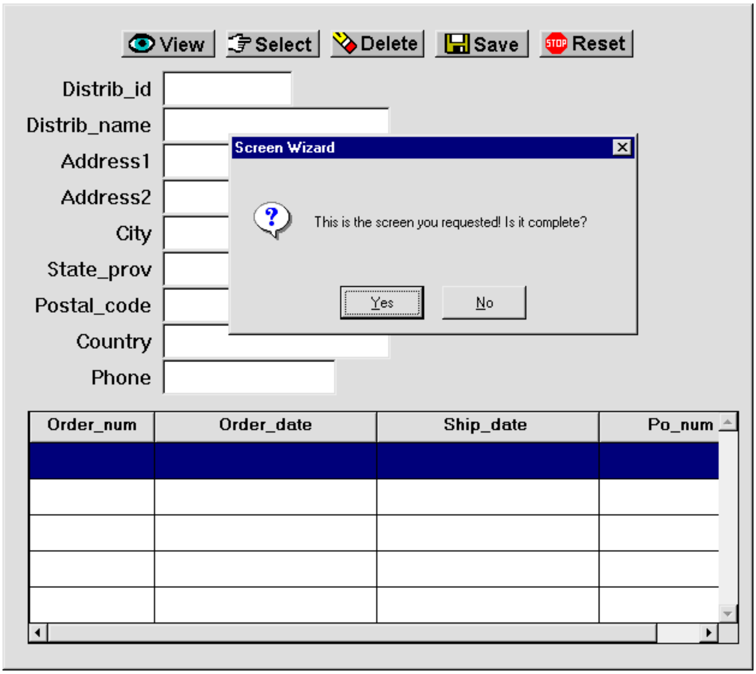
The Style and Finish dialog lets you can customize the client screen. Because you chose the web-friendly option, the Menu/Toolbar option is greyed out; web browsers do not support menus or toolbars.

15 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).

16 (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.

17 Choose Done. Panther displays the finished screen with this prompt:



Screen Wizard

This is the screen you requested! Is it complete?

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

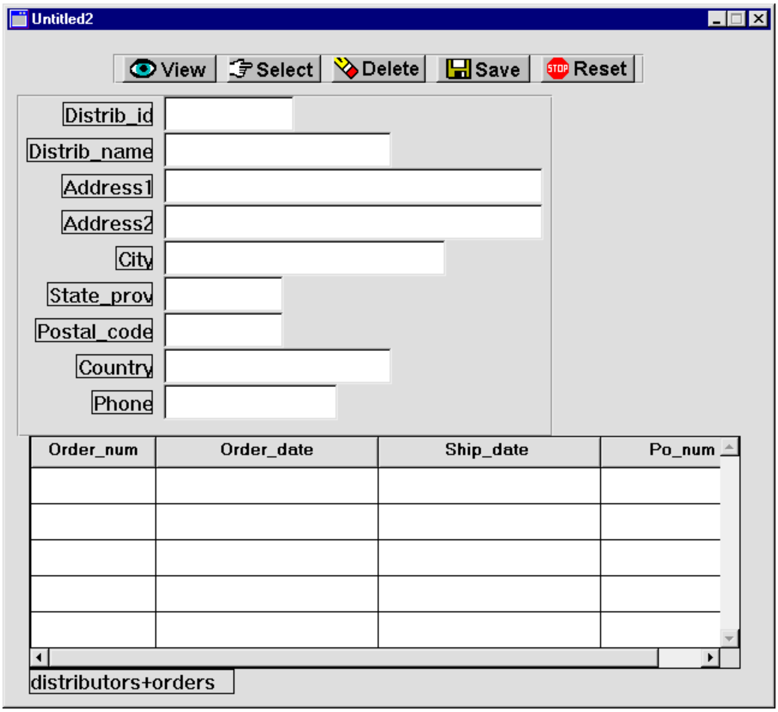
If you choose No, you return to the screen wizard's Style and Finish dialog. You can make further changes there and in previous wizard dialogs.

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

Save the screen

This is a good time to save the screen to client library `client.lib`.

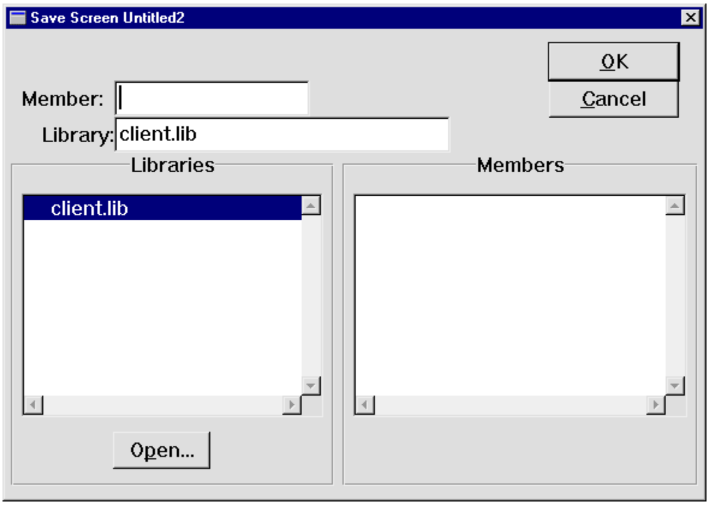
19 If necessary, bring focus to the client screen (Untitled2).



Order_num	Order_date	Ship_date	Po_num

20 Choose File→Save or 💾 .

The Save Screen dialog opens where you specify a library and name for the untitled screen.



Member:

Library: client.lib

Libraries

client.lib

Members

21 Enter `dstord.scr` in the Member field.

22 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. The client screen is the screen a user 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 *.jpl for library JPL modules.

23 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 from the open repository.
- Saved the screen to the client 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 can build a variety of screens according to the format that you choose.

LESSSON

5 Testing Screens

In this lesson, you test the screen that you created in Lesson 4. You can test this screen either within the editor or in the browser. You also learn how the transaction manager controls an application's behavior and appearance.

In this lesson you learn how to:

- Test a screen in GUI mode or a web browser.
- Access the database to view and update data.

Test the screen in the browser

In your web browser, you can test any screen that is stored in the web application library.

1. If necessary, start the Panther web application.

   Type the following from the command line of the *WebInstallDir*/util directory:

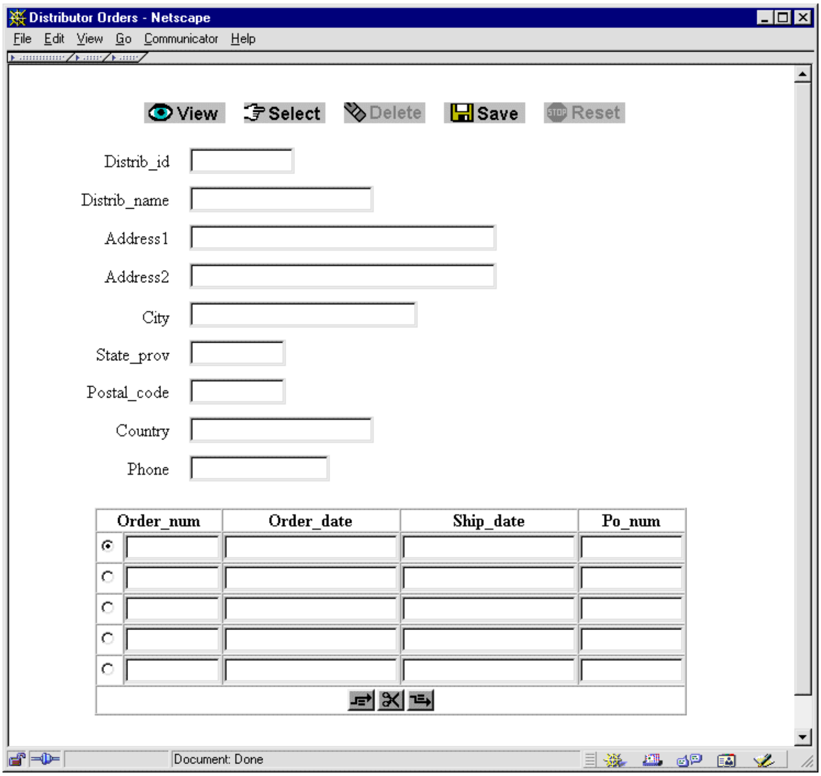
   UNIX `monitor -start` *WebAppname*

   Windows `monitor -start` *WebAppname*

2. To invoke the screen that you created in Lesson 4, supply its URL in the following format:

   `http://`*hostname*`/cgi-bin/`*WebAppname*`/dstord.scr`

   The screen appears in the browser.


Distributor Orders - Netscape
File Edit View Go Communicator Help
View
Select
Delete
Save
Reset
Document: Done

Distrib_id

Distrib_name

Address1

Address2

City

State_prov

Postal_code

Country

Phone

	Order_num	Order_date	Ship_date	Po_num
●				
○				
○				
○				
○				

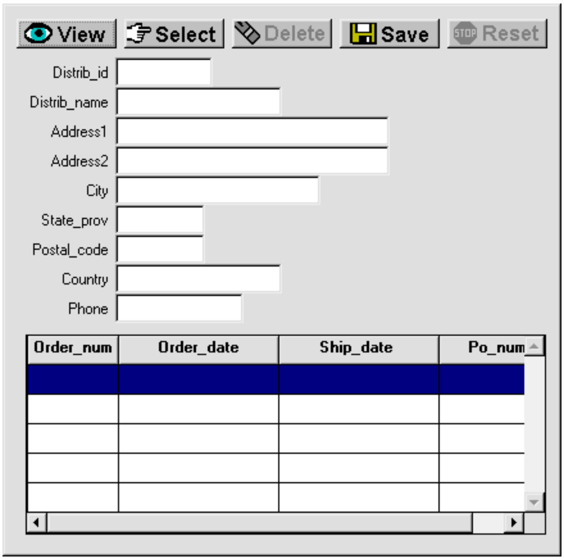
To continue testing the screen in the web browser, skip to Step 5.

Test the screen in GUI

You can also test and debug any screen within the editor. The editor's graphical environment shows how the screen appears and behaves in the host GUI platform. Any changes that you make to a screen can be tested immediately without saving edits, so you can experiment without committing to the changes. For more information about test mode, refer to Chapter 38, “Testing Application Components,” in *Application Development Guide*.

3 If necessary, restart the editor and open `dstord.scr`.

4 Choose File→Test Mode to test the screen.



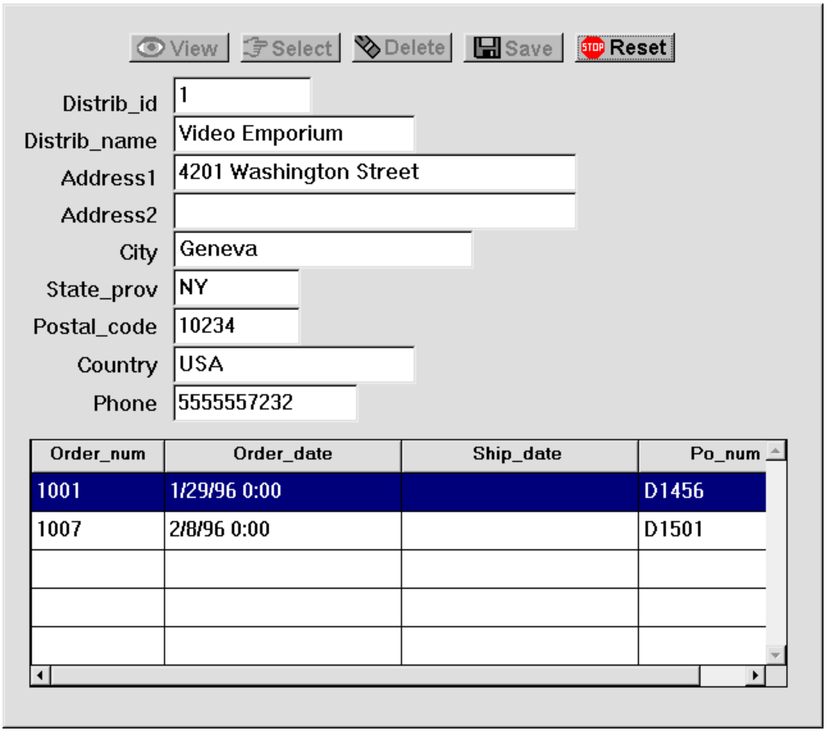
View data

With the screen that you just built, you can access real data from the `vidsales` database. The commands associated with the screen's buttons let you access and maintain data in the database.

If necessary, open the `vidsales` database (Database→Connect).

5 Choose View .

The first record in the `distributors` table displays.



Distrib_id	1
Distrib_name	Video Emporium
Address1	4201 Washington Street
Address2	
City	Geneva
State_prov	NY
Postal_code	10234
Country	USA
Phone	5555557232

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 prevents you from saving or deleting records.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 behavior of the default styles with the styles editor, or change the widget's style assignment in the editor.

Also, because the View command only allows read access, the record data display as literal text and not inside input fields.

6 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.

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

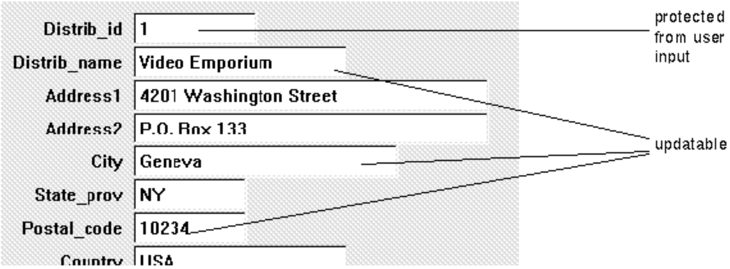
In order to update database records, you must select them. When you select records for update, by default Panther protects primary key fields from data entry. This is a result of Panther's application of a style to each widget.

7 Choose Select.

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

The value in `distrib_id` is shown as read-only because it is a primary key in the `distributors` table; in general, primary key fields in database tables cannot be changed.

8 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
Distrib_name	Video Emporium
Address1	4201 Washington Street
Address2	P.O. Box 133
City	Geneva
State_prov	NY
Postal_code	10234
Country	USA

protected from user input

updatable

Save the changes

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

9 Choose Save .

Panther calls the update procedure to update the database.

10 To close the current transaction, choose STOP Reset .

Note: The following is only true for Windows. In Motif, the system menu is available, and you can choose close.

11 (Windows only) Press Escape twice to exit test mode if viewing in GUI.

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.scr` 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 the test mode offer functionally that is equivalent to the buttons.

What did you do?

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

- For web applications, access the screen in a web browser. For GUI applications, accessed test mode from the editor workspace.
- Executed database commands to view and update 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.

LESSSON

6 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.

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.

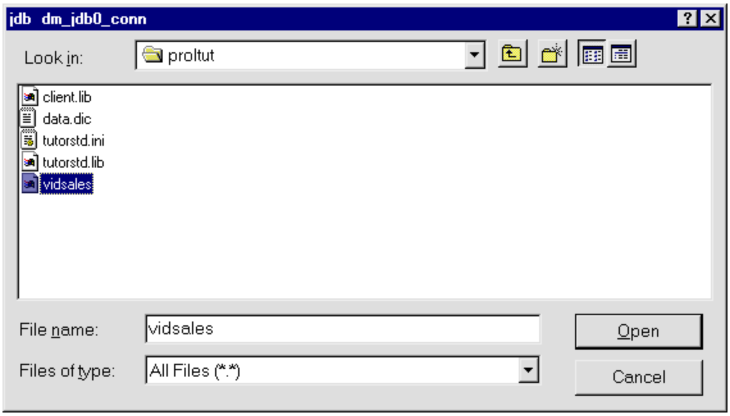
In this lesson, you set properties that let users search the database for a specific distributor by entering an identification number. The transaction manager automatically generates the appropriate SQL query statement based on this input, submits the input to the database engine, and returns the desired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

In this lesson you learn how to:

- Set widget properties on the screen.
- Specify a particular widget to act as a query field.
- Test the resulting query screen by searching for specific distributor records.

1 If necessary:

- Invoke the editor and connect to the database.
- Reopen the `dstord.scr` screen in `client.lib`.
- To start the screen in a web browser, restart the web application.



Look in: proltut

File name: vidsales

Files of type: All Files (*.*)

Use 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.

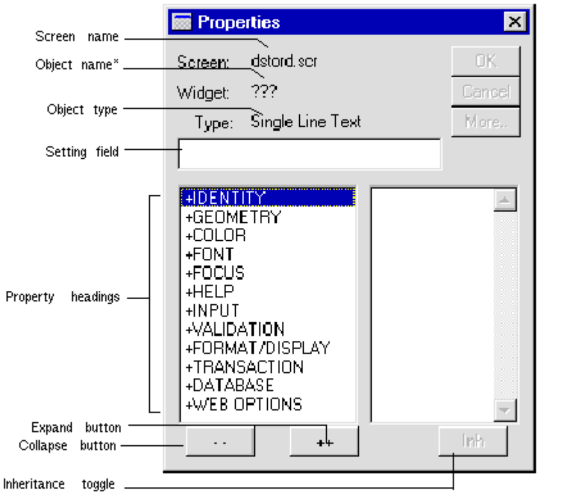
- Choose View→Properties Window.

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.



Screen: dstord.scr

Widget: ???

Type: Single Line Text

*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 - - .

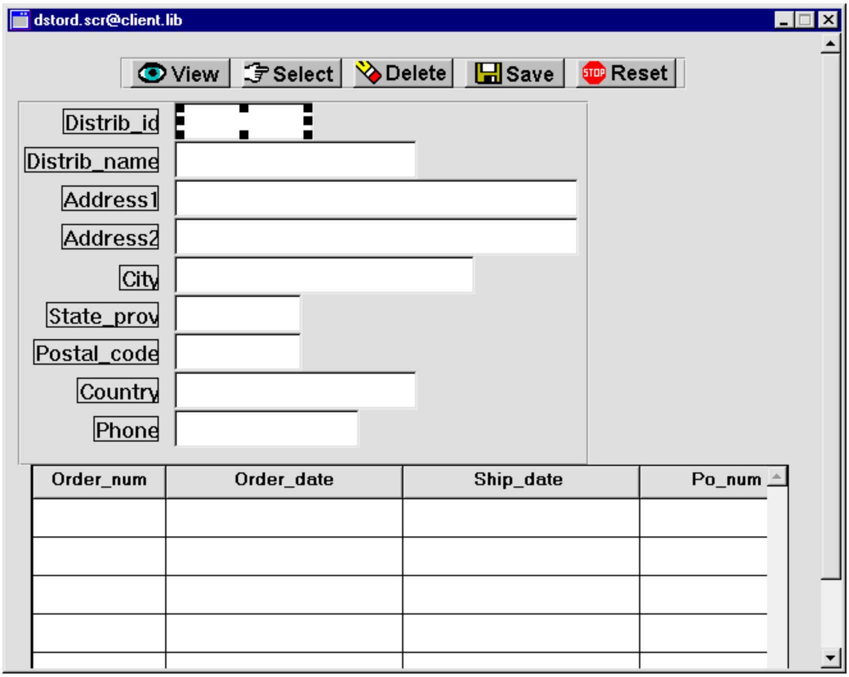
The properties list collapses, displaying only the property headings.

No matter how you set a property, you can reverse any change with Edit→Undo or ↶ .

Change properties

The next several steps show how to set a property on a widget located on 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.

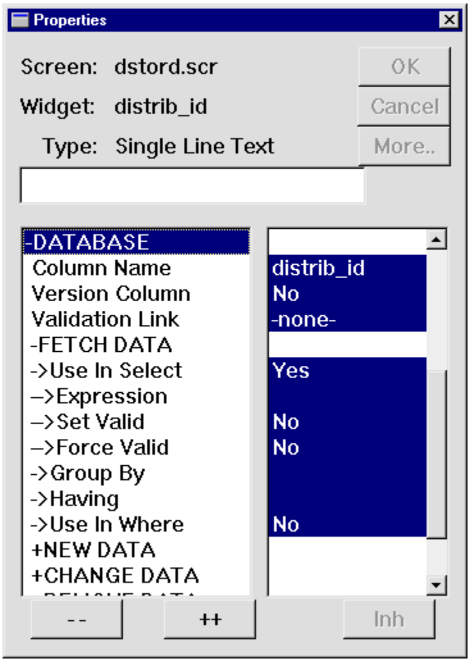
4 On the screen `dstord.scr`, select the single line text (data entry) widget `distrib_id`, which is adjacent to the `Distrib_id` label.



5 In the Properties window, select the Database heading.

Database-specific properties are displayed.

6 Under the Database heading, select Fetch Data. Fetch-specific properties are displayed.



Screen: dstord.scr

Widget: distrib_id

Type: Single Line Text

-DATABASE	
Column Name	distrib_id
Version Column	No
Validation Link	-none-
-FETCH DATA	
->Use In Select	Yes
–>Expression	
–>Set Valid	No
–>Force Valid	No
->Group By	
->Having	
->Use In Where	No
+NEW DATA	
+CHANGE DATA	

7 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

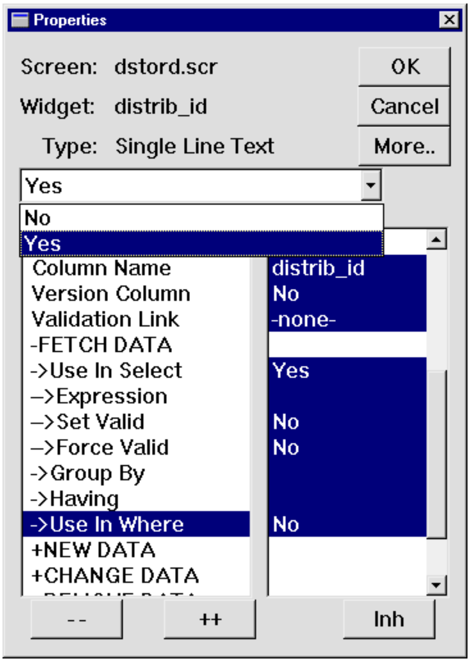
You can set a property with predefined values in several ways after you select it:

- 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.

8 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.



Screen: dstord.scr
Widget: distrib_id
Type: Single Line Text

Yes

Column Name	distrib_id
Version Column	No
Validation Link	-none-
-FETCH DATA	
->Use In Select	Yes
->Expression	
->Set Valid	No
->Force Valid	No
->Group By	
->Having	
->Use In Where	No
+NEW DATA	
+CHANGE DATA	

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.

Save the screen

In order to view your changes in the browser, you must first save the screen.

9 With focus on the `dstord.scr` screen, choose File→Save or 💾.

The screen is saved to `client.lib`.

View specific records

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

10 If the `dstord.scr` screen still displays inside your browser, reload it. Otherwise, invoke the screen by supplying this URL:

http://*hostname*/cgi-bin/*webAppname*/dstord.scr

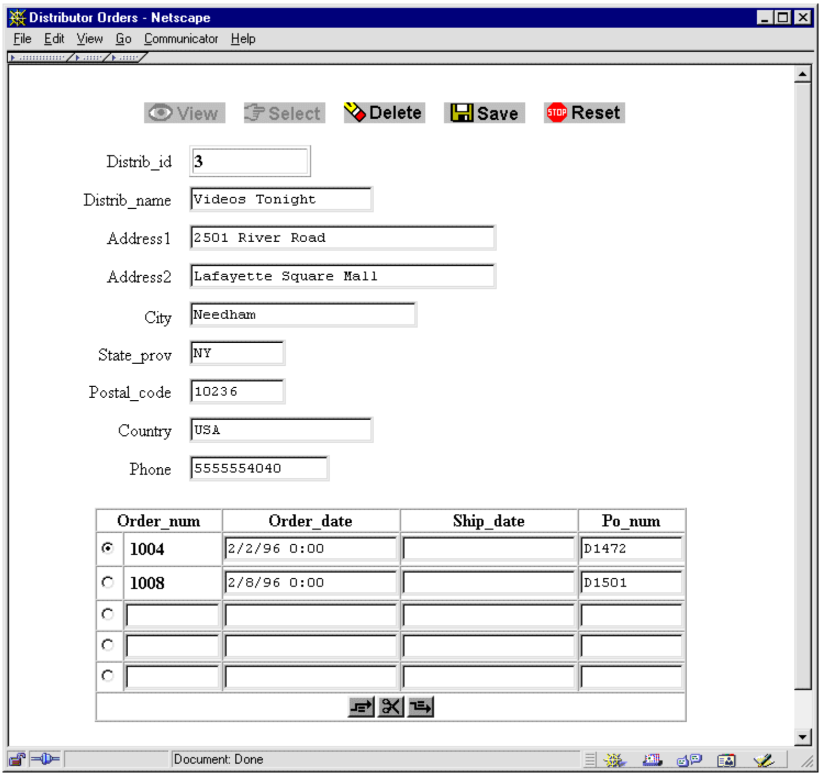
You can also choose File→Test Mode to view the screen in GUI.

The Distributor Orders screen displays.

11 Enter 3 in the `distrib_id` field.

12 Choose ☞ **Select** .

The distributor record for Videos Tonight displays.


Distributor Orders - Netscape
File Edit View Go Communicator Help
View
Select
Delete
Save
Reset
Document: Done

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
○				
○				
○				

Now, modify the phone number.

13 Choose **Save** .

Your changes are written to the database.

14 Choose **Reset** .

The transaction closes. You can try all the buttons on the screen. After each transaction, remember to choose Reset to close 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, do not delete distributors via the `dstord.scr` screen.

15 Close the browser window or press Shift+F5 (Esc twice or choose Options→Editor) to exit GUI 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 the screen by setting its Use In Where property.
- Test the resulting screen by entering search criteria.

What did you learn?

You learned:

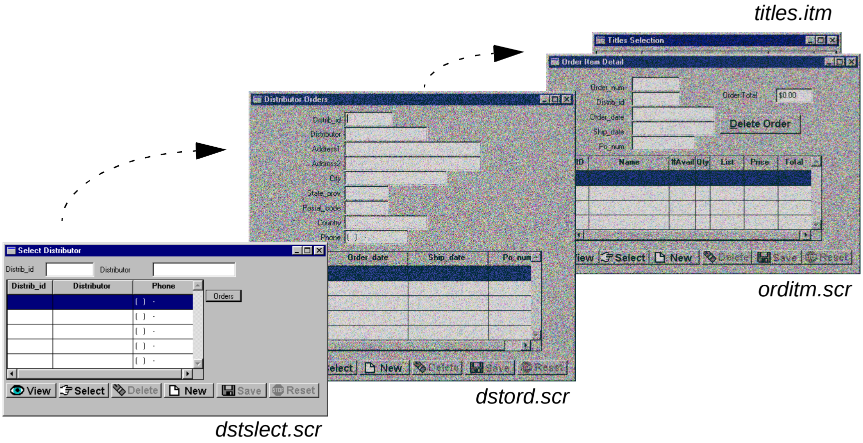
- The Properties window lets you set an object's behavior.
- Inherited property values can be selectively overridden, without affecting inheritance links for other properties.

Module 3: Connecting the Screens

LESSSON

7 Enhancing the Screen

An additional screen is provided in `tutorstd.lib` that uses a grid widget to display a list of distributors. In this lesson, you enhance this screen 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:

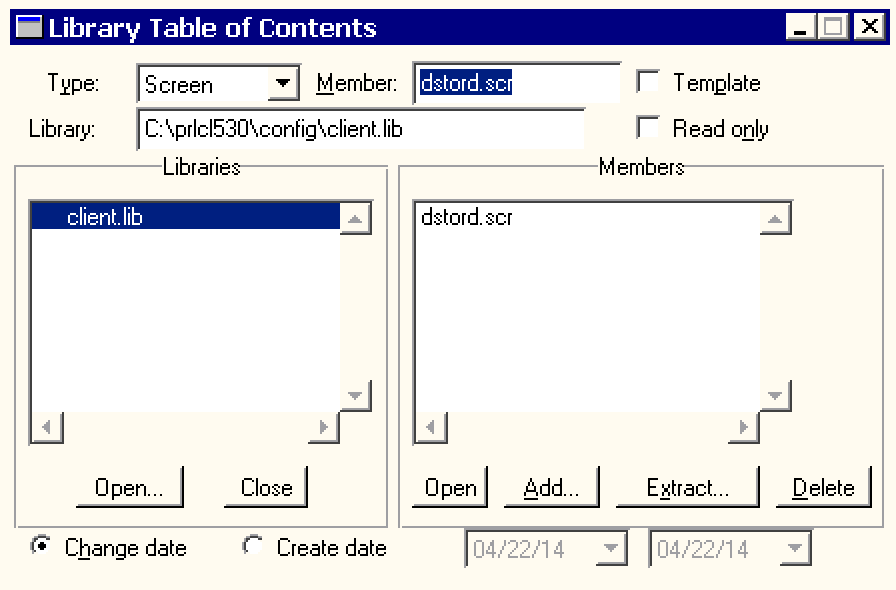
- Resize the screen and copy widgets to it from a repository entry.
- Specify and define more than one widget to act as a query-by-example field, so users can use a variety of search criteria to query the database.

In this lesson, you open a wizard-built screen from the tutorial library. You then save it to `client.lib`, and enhance the user interface by including query fields.

1 If necessary, invoke the editor and connect to the database.

2 Choose View→Library TOC.

The Library TOC opens.



Library Table of Contents

Type: Screen

Member: dstord.scr

□ Template

Library: C:\prlcl530\config\client.lib

□ Read only

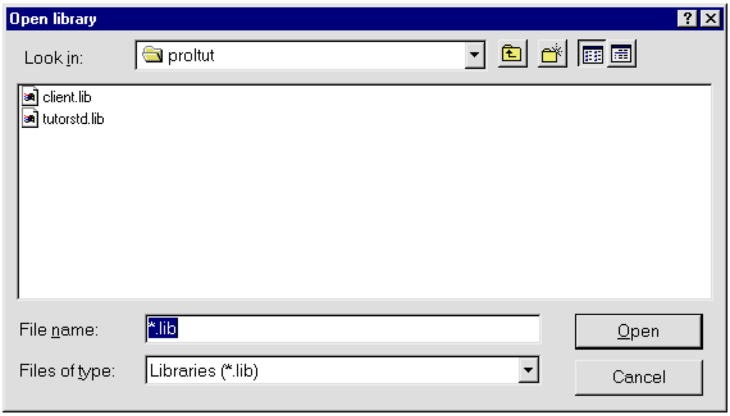
Libraries: client.lib

Members: dstord.scr

● Change date ○ Create date 04/22/14 04/22/14

3 If `tutorstd.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.



Open library

Look in: proltut

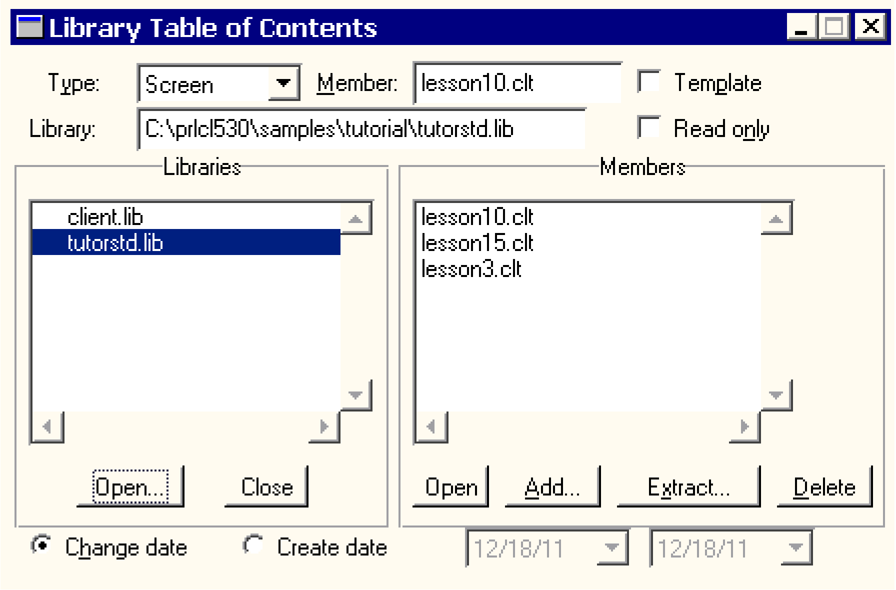
client.lib
tutorstd.lib

File name: *.lib

Files of type: Libraries (*.lib)

- Select `tutorstd.lib`.
- Choose Open. The Library TOC redisplays.

4 Select `tutorstd.lib` from the list of open libraries and `lesson10.clt` from the list of library members. Choose Open.



Library Table of Contents

Type: Screen Member: lesson10.clt ☐ Template

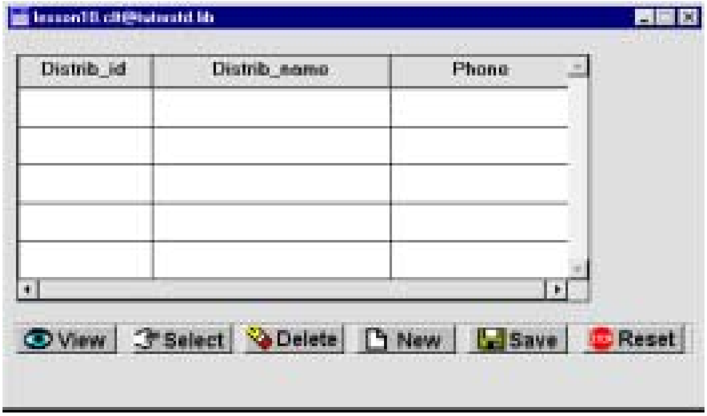
Library: C:\prlcl530\samples\tutorial\tutorstd.lib ☐ Read only

Libraries:
client.lib
tutorstd.lib

Members:
lesson10.clt
lesson15.clt
lesson3.clt

● Change date ○ Create date 12/18/11 12/18/11

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.

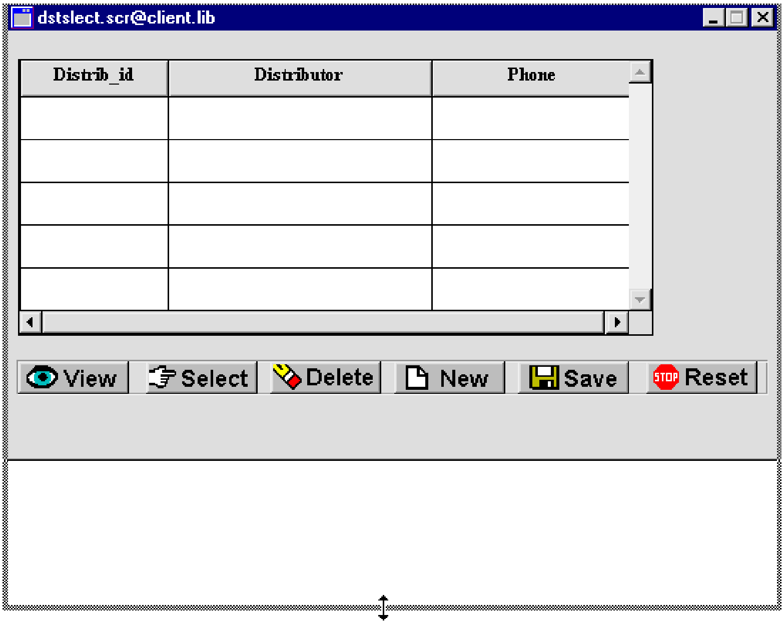
6 Save the screen as `dstslect.scr` in `client.lib`.

Resize the screen

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

7 In the editor workspace, resize the `dstslect.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.



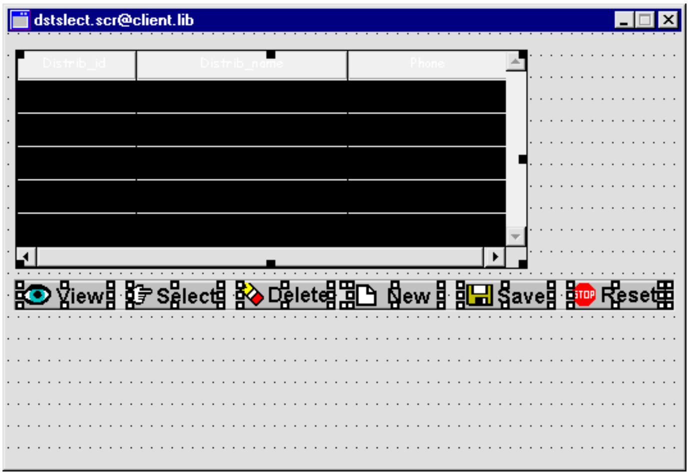
- Click in an empty area of the client screen to deselect all widgets.

  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.

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



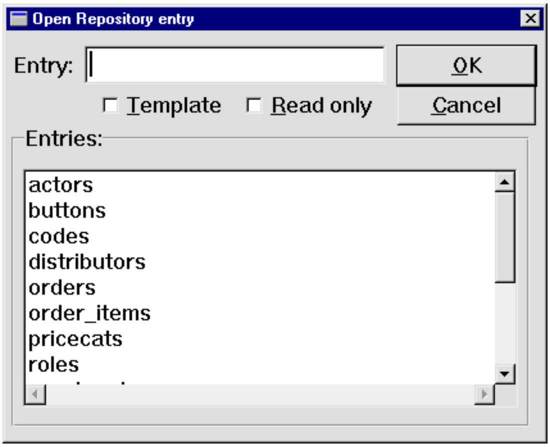
Open a repository entry

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.

9 If the repository is not open, choose File→Open→Repository, and select `data.dic` in the `proltut` directory.

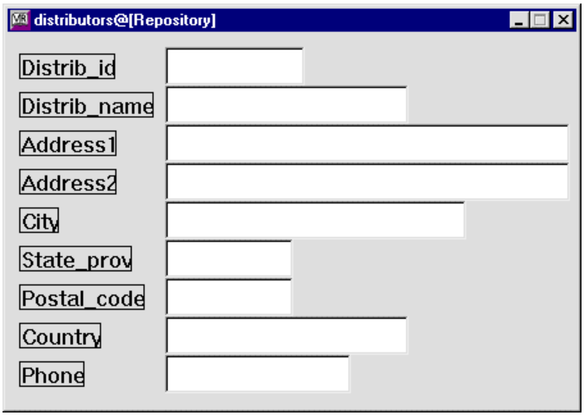
10 Choose File→Open→Repository Entry.

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



11 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 `dstslect.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, as shown in the next lesson.

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

12 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.

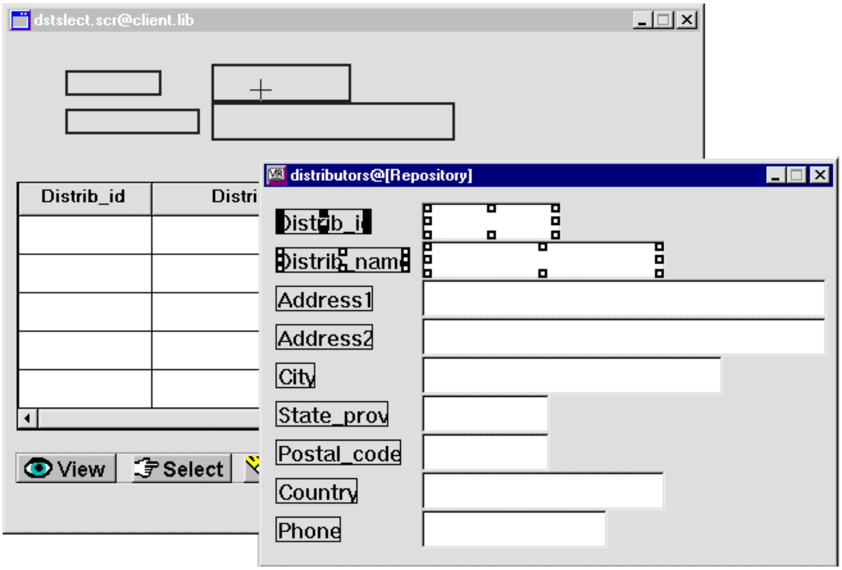
More About Selecting More than One Widget

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.

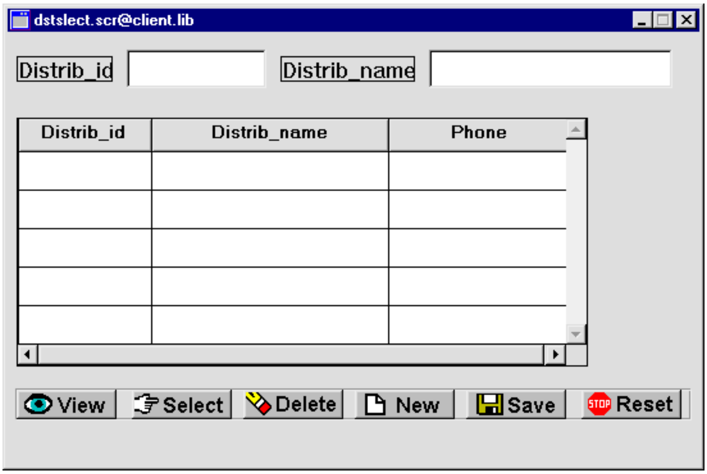
There are a variety of ways to select multiple widgets:

- 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.

13 Drag the widgets from the repository screen `distributors` to the top of the screen `dstslect.scr`.



14 (Optional) Bring focus to the distributors repository entry and choose File→Close→distributors. The repository window closes.



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.

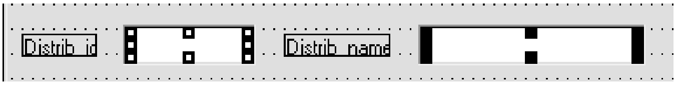
15 Select the copied text widgets on the client screen `dstslect.scr` and set each one's Name property (under Identity) as follows:

- `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.

16 Select the `distid_qbe` and `distname_qbe` widgets.



17 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.

18 Under Database, select `NEW DATA`. Under `NEW DATA`, set the following property for both widgets:

- Use In Insert = No

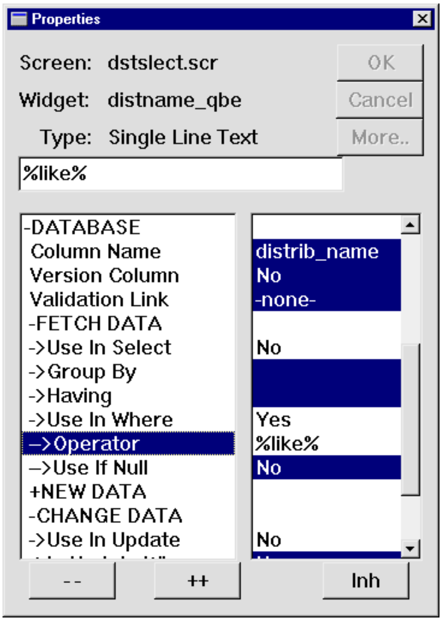
19 Under Database, select `FETCH DATA`. Under `FETCH DATA`, set these properties for both widgets:

- 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.

20 Select the `distname_qbe` widget.

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



Screen: dstslect.scr

Widget: distname_qbe

Type: Single Line Text

%like%

-DATABASE	
Column Name	distrib_name
Version Column	No
Validation Link	-none-
-FETCH DATA	
->Use In Select	No
->Group By	
->Having	
->Use In Where	Yes
–>Operator	%like%
–>Use If Null	No
+NEW DATA	
-CHANGE DATA	
->Use In Update	No

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.

22 Save the screen (press F6).

Query the database

You can try it out!

23 Invoke the screen from your browser by supplying this URL:

`http://`*`hostname`*`/cgi-bin/`*`webAppname`*`/dstslect.scr`

The Select Distributor screen opens.

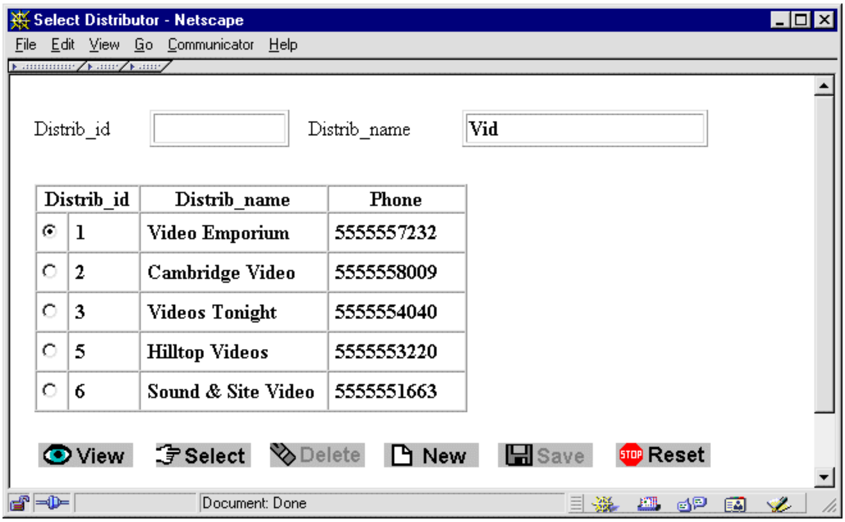
Or choose File⟶Test Mode to view the screen in a GUI environment.

Note: You may need to start the web application server before viewing the screen in your browser.

24 Type `Vid` in the `Distrib_name` field.

25 Choose View.

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


Select Distributor - Netscape
File Edit View Go Communicator Help
View
Select
Delete
New
Save
Reset
Document: Done

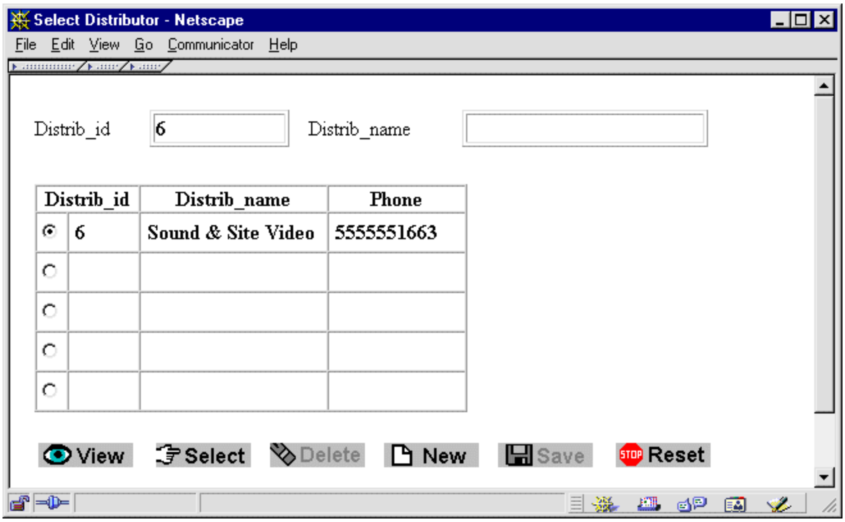
Distrib_id

Distrib_name Vid

	Distrib_id	Distrib_name	Phone
●	1	Video Emporium	5555557232
○	2	Cambridge Video	5555558009
○	3	Videos Tonight	5555554040
○	5	Hilltop Videos	5555553220
○	6	Sound & Site Video	5555551663

26 Choose Reset.

27 Type 6 in the `Distrib_id` field and choose View.



Distrib_id 6 Distrib_name

Distrib_id		Distrib_name	Phone
●	6	Sound & Site Video	5555551663
○			
○			
○			
○			

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.

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.

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.

LESSSON

8 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 to all application screens.

In this lesson you learn how to:

- Control user input by defining an edit mask.
- Edit a repository entry and propagate the changes from the repository to screens.
- Create a widget on the screen and copy it to a repository entry for later use.

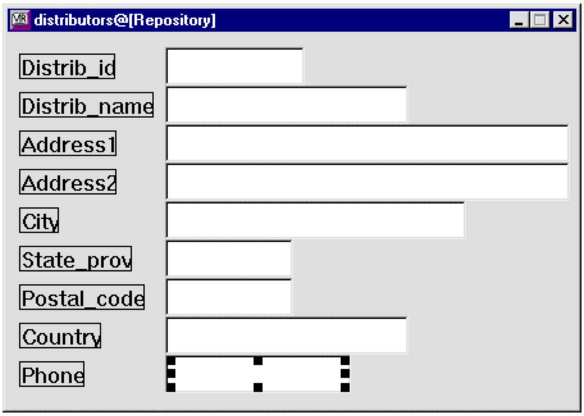
1 If necessary:

- Invoke the editor.
- Reopen the `dstslect.scr` screen from `client.lib`.
- 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.



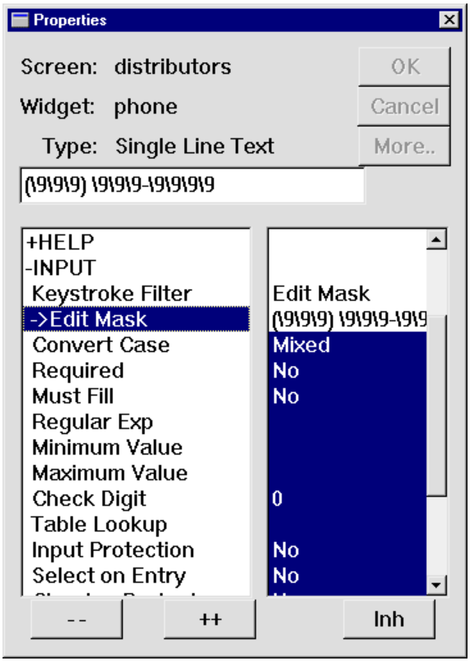
3 Under Input, set the Keystroke Filter property to Edit Mask.

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.



Screen: distributors
Widget: phone
Type: Single Line Text

(\9\9\9) \9\9\9-\9\9\9\9

+HELP	
-INPUT	
Keystroke Filter	Edit Mask
->Edit Mask	(\9\9\9) \9\9\9-\9\9
Convert Case	Mixed
Required	No
Must Fill	No
Regular Exp	
Minimum Value	
Maximum Value	
Check Digit	0
Table Lookup	
Input Protection	No
Select on Entry	No

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:

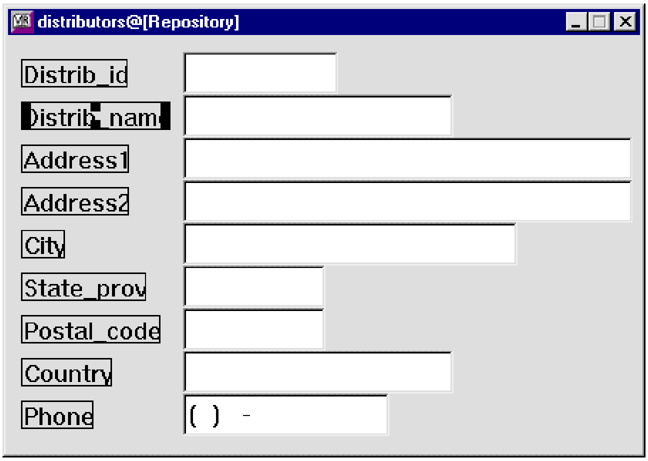
- Digits-only—Allows entry of the digits 0 through 9 only.
- Alphanumeric—Allows entry of any digits, the letters a-z and a-Z, and the space character.
- Yes/No fields—Allows entry of the initial letters of “yes” and “no.”
- Case style—Enforces upper-case, lower-case, mixed case conversion.
- Required 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 example, using more descriptive label text or using a different font for data entry widgets. The following 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 repository changes

Changes made to parent widgets in the repository are visible in child widgets on screens that are open in the editor when the repository entry is saved. To propagate changes to child widgets on other screens, open those screens 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 the `dstslect.scr` screen. 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

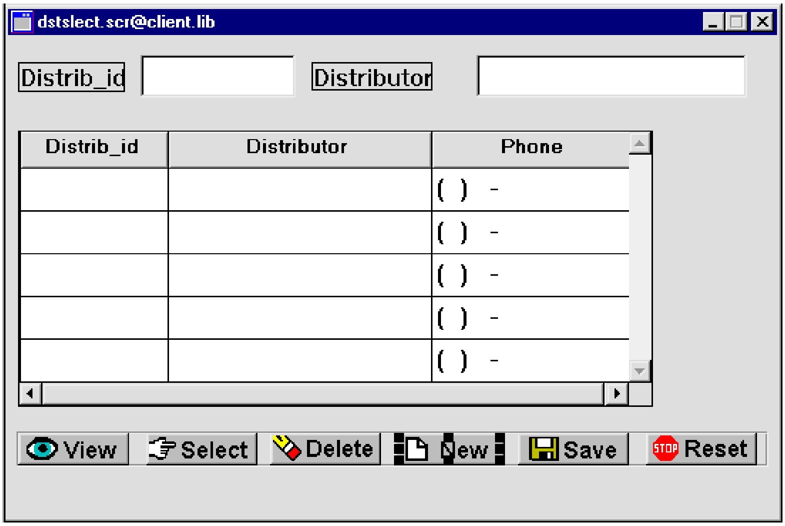
You can modify the behavior established by the screen wizard by writing your own procedure to carry out a particular action. For example, the `dstslect.scr` screen acts as a search or query screen for finding distributor records, so you might consider using a different screen for adding new distributors 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

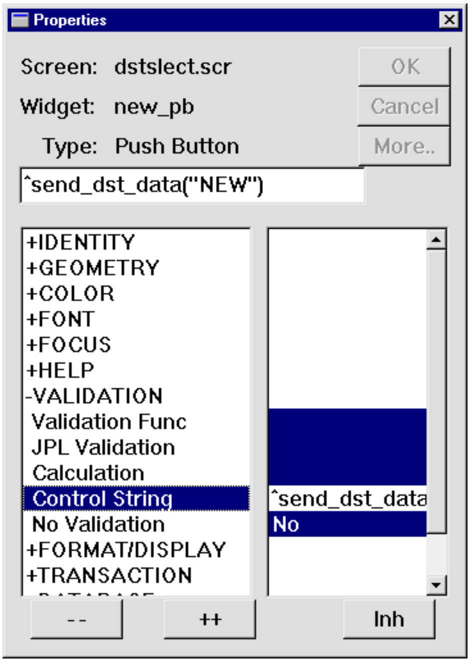
You can attach actions to widgets, menu items, and specific logical keys through control strings. Control strings are a shorthand notation for doing common tasks:

- 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 `dstslect.scr` screen.



15 Under Validation, change the Control String property from its inherited property value `^do_new` to `^send_dst_data("NEW")`.


Properties

Screen: dstslect.scr

Widget: new_pb

Type: Push Button

^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 9 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

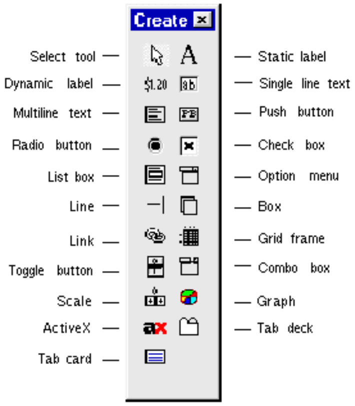
Create a push button that executes a procedure to send data from this screen to the `dstord.scr` screen, which you created in Module 2.

You can create widgets with either the Create menu or the Tool box. The next several steps show how to use the Tool Box.

16 If the Tool Box is not open:

UNIX Choose View→Tool Box

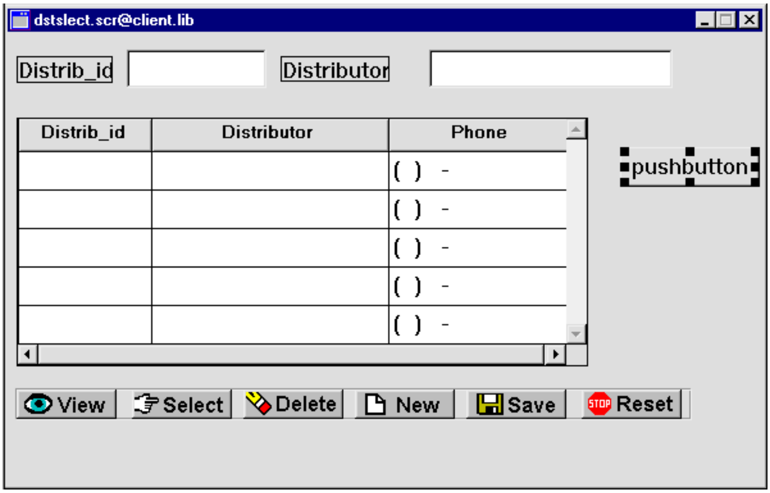
Windows Choose Options→Configure Toolbars. Select Create.



17 Choose Create→Push Button or

18 Click near the right side of the screen.

A default-sized push button appears at the cursor position.



19 With the button selected, under Identity, 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 supplies 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.

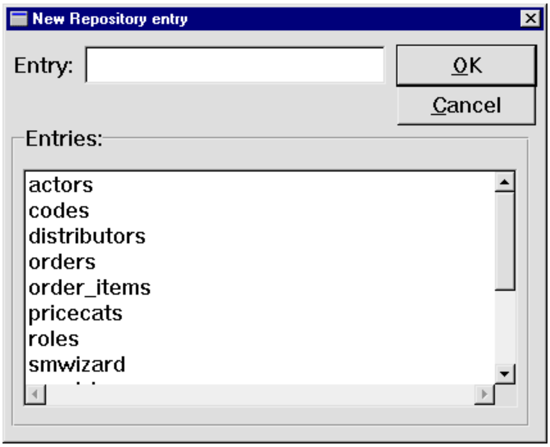
The `dstslect.scr` screen is saved to the `client.lib` library.

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 screen to a repository entry. An inheritance link is automatically established between the object and its parent in the repository.

26 Choose File→New→Repository Entry.

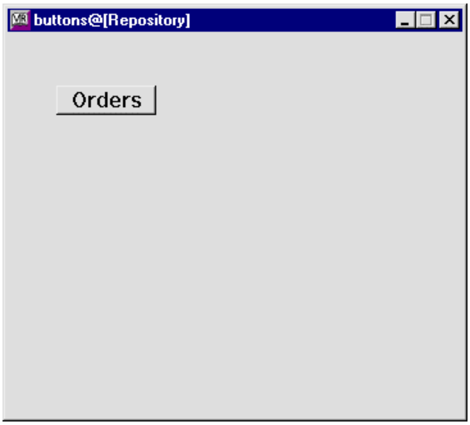
The New Repository Entry dialog box opens.



27 Enter `buttons` as the name of the entry. Choose OK.

An empty window with the name `buttons@[Repository]` opens in the workspace.

28 Select the `order_pb` push button on the `dstslect.scr` screen and drag it to the `buttons` repository entry.



29 Return to the `dstslect.scr` screen 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 entry focus and choose File→Save.

The `buttons` repository entry is saved to the open repository.

31 (Optional) Choose File→Close→buttons.

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 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.

LESSSON

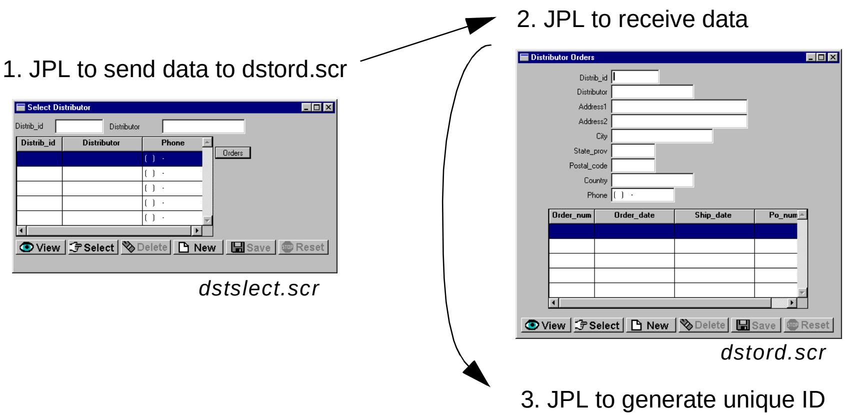
9 Writing and Executing JPL

You can embed special processing in your application screens with Panther's scripting language JPL. Because the JPL is saved with the screen, it is accessible to the screen and its widgets.

In this lesson you learn how to:

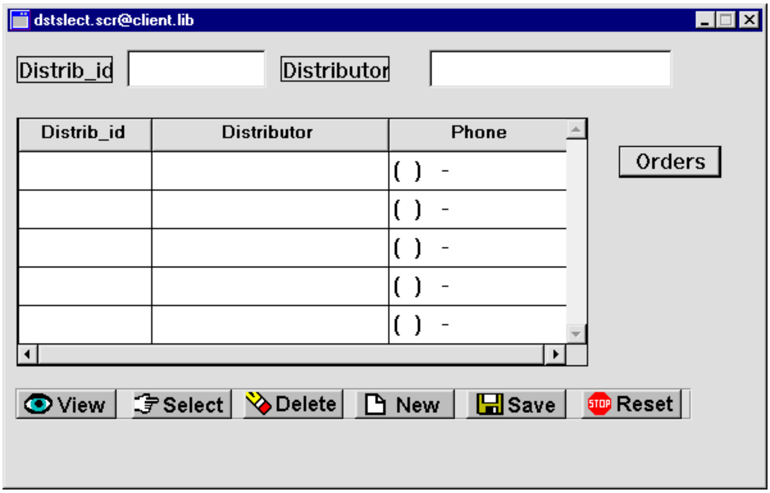
- Attach JPL procedures to screens: one procedure to send data and the other to receive the transmitted data.
- Implement error handling by displaying a message to the user.
- Use the DB Interactions window to view database objects and their relationships on your application screens, and to gain access to their properties.
- Write transaction manager hook functions that automatically generate a distributor ID number when a distributor is added to the database.

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 `tutorstd.lib`. The following diagram illustrates the JPL procedures you will enter in this lesson.



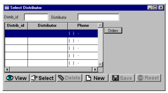
1 If necessary:

- Invoke the editor and connect to the database.
- Reopen client screen `dstslect.scr` from the `client.lib` library.



Write a procedure to access a distributor’s orders

JPL to send data



dstslect.scr

Write a JPL procedure that is called when a user chooses the New or Orders button on the `dstslect.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.

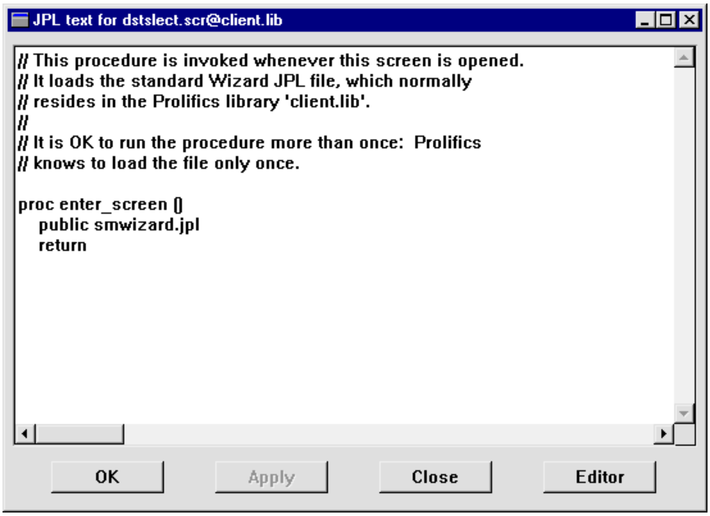
Because the same procedure is called by two different screen objects, make the procedure available to both by attaching the JPL code to the screen's JPL Procedures property.

2 Select the `dstslect.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:



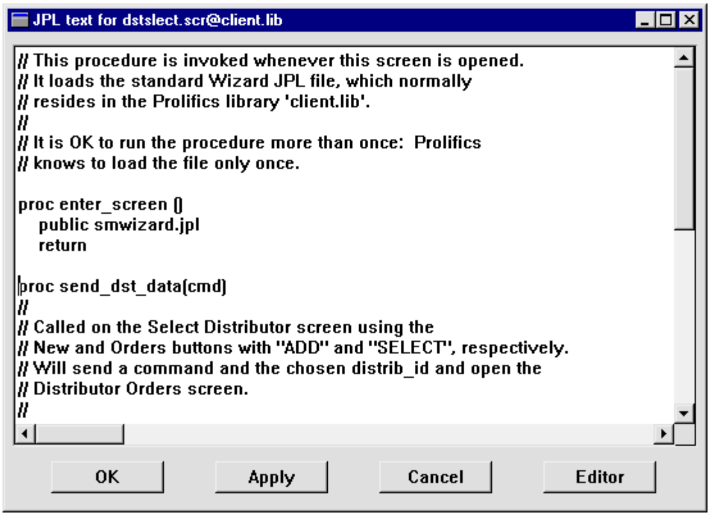
```
// This procedure is invoked whenever this screen is opened.
// It loads the standard Wizard JPL file, which normally
// resides in the Prolifics library 'client.lib'.
//
// It is OK to run the procedure more than once:  Prolifics
// knows to load the file only once.

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 `tutorstd.lib`:

- Choose Edit→Insert From Library. The Open JPL Module dialog opens.
- If necessary, open `tutorstd.lib` by choosing Open and selecting it from the Open Library dialog.
- From the Open JPL Module dialog, select `tutorstd.lib` from the list of open libraries, and select `send_dst.jpl` from the list of members. Choose OK.



```
// 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 (`dstslect.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 Apply to save the procedure.

6 Choose File→Save to save the 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 Editor.

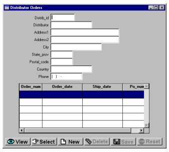
When you exit the system editor, you resume in the JPL edit window, where you save your changes back to its source by choosing Apply or OK. You can also save library JPL modules to a new library member by choosing Save As.

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 as 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

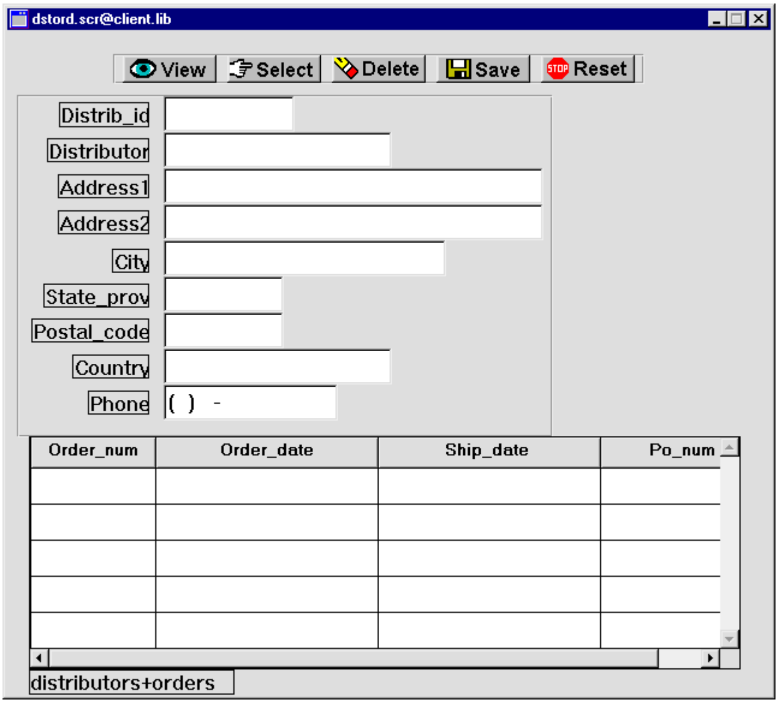
JPL to receive data



dstord.scr

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 `dstslect.scr` screen's New or Orders buttons.

7 If necessary, reopen the `dstord.scr` screen from `client.lib`.

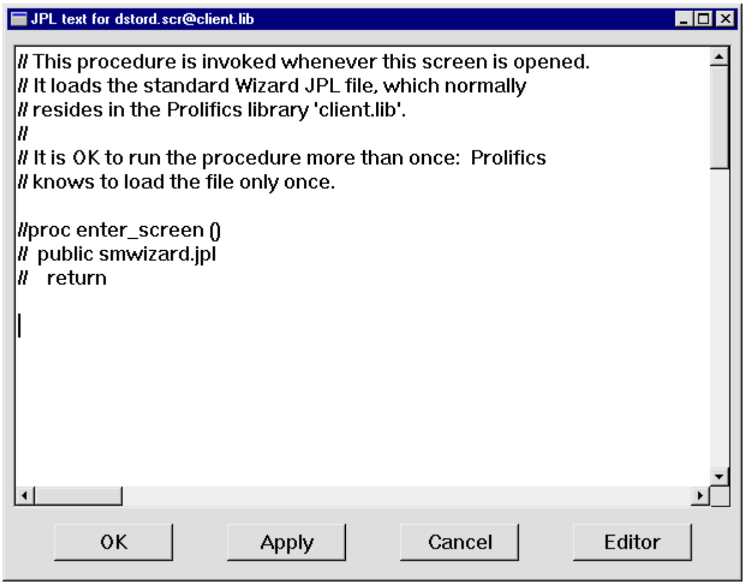


8 Select the `dstord.scr` client screen by clicking in an empty space within its borders, so no widgets are selected.

9 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 `tutorstd.lib`.

10 Comment out the original `enter_screen` procedure by placing // at the beginning of each of the three lines.



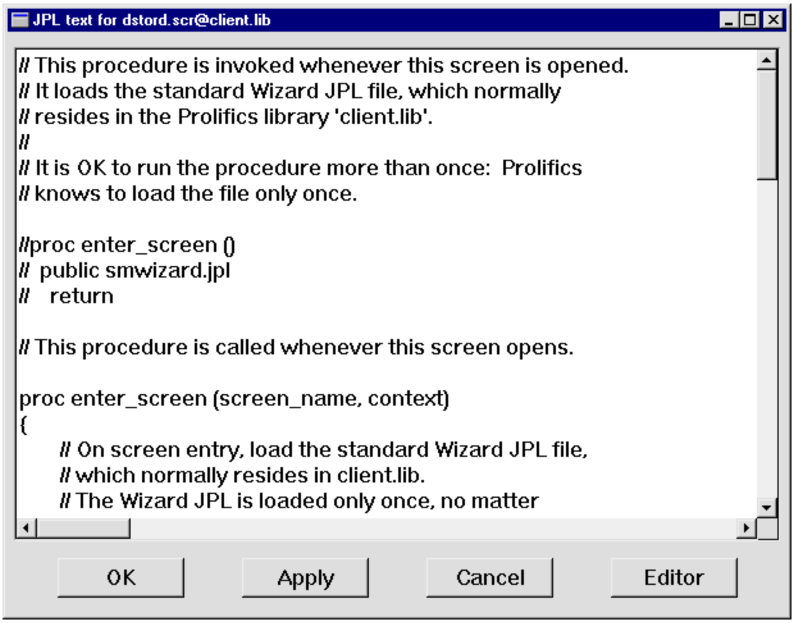
```
// This procedure is invoked whenever this screen is opened.
// It loads the standard Wizard JPL file, which normally
// resides in the Prolifics library 'client.lib'.
//
// It is OK to run the procedure more than once:  Prolifics
// knows to load the file only once.

//proc enter_screen ()
//  public smwizard.jpl
//    return
```

11 Position the cursor a line or two below the `return` line of the commented procedure.

12 Choose Edit⟶Insert From Library.

13 From the Open JPL Module dialog, select `se_dst.jpl` from `tutorstd.lib`. Choose OK.



```
// 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
```

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
    }
}
```

More About the `enter_screen` Procedure

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 `dstslect.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 `dstslect.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`.

14 Save the procedure by choosing Apply.

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.

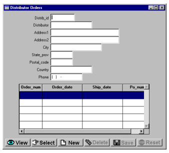
Panther provides several ways for screens to access each other's data:

- JPL can directly reference variables and widgets on other open screens with the syntax convention `screen_name!widget_name`.
- 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, refer to Chapter 25, “Moving Data Between Screens,” in the *Application Development Guide*.

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 `dstslect.scr` (Select Distributor) screen, and opens `dstord.scr` (Distributor Orders) in insert mode. You can now write a JPL hook function for `dstord.scr` that generates a unique distributor ID number. This hook function executes during transaction manager processing and supplements the default transaction model.

15 Position the cursor at the very end of the JPL file for the `dstord.scr` to start a new procedure.

16 Choose Edit→Insert From Library.

17 From the Open JPL Module dialog, select `evnt_svr.jpl` from `tutorstd.lib` and choose OK.

The `tm_events` procedure is read into the JPL edit window:

```
proc tm_events (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
    }
    else if (event_id == TM_POST_SAVE)
    {
        distrib_id->hidden=PV_NO
        LDistrib_id->hidden=PV_NO
    }
    return TM_PROCEED
}
```

More About the `tm_events` Procedure

The `tm_events` 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` statement specifies a `SELECT` to find the highest distributor ID in the `distributors` database table and increments that number by one.

The hook function then 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.

18 Choose OK to save the procedure and exit the JPL window.

19 Choose File→Save to save the screen.

More About SQL DBMS 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 Chapter 28, “Writing SQL Statements,” in *Application Development Guide* for more information

Invoke the hook function

When you used the screen wizard to create `dstord.scr` in Lesson 4, 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 entry's source table; it is created automatically when you import the table into the repository (Lesson 3).

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.

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.

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.

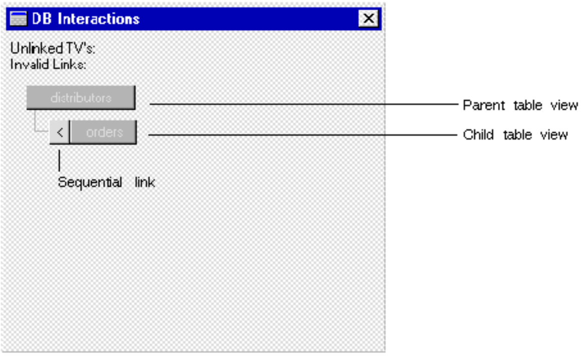
You attach a hook function to a table view through its Function property. In this case, you want to attach a hook function to the `distributors` table view. You can select a table widget and view its properties through either the Widget List or the DB Interactions window. In this instance, use the DB 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.

20 With focus on `dstord.scr`, choose View→DB Interactions.

The DB Interactions window opens, displaying a graphical representation of table views and links on the `dstord.scr` screen.



More About Sequential and Server Links

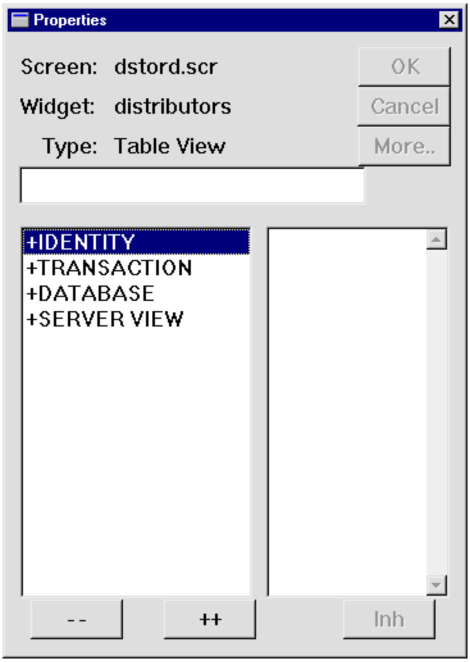
Link widgets are not visible at runtime, but are visible in the editor so that you can access their properties.

There are two type of links-sequential and server:

- Sequential links (denoted by < 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 = 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.

21 In the DB Interactions window, select the button that represents the `distributors` table view.

The Properties window now displays table view properties.



Screen: dstord.scr
Widget: distributors
Type: Table View

+IDENTITY
+TRANSACTION
+DATABASE
+SERVER VIEW

22 Under Transaction, set the Function property to `tm_events` and choose OK.

23 Save all the open screens (press F6).

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*.

Refine the screen for GUI display

In Lesson 4, you created the screen and chose web-friendly output, which affects the appearance of the screen when it is called from another screen. To adjust the appearance of the screen for GUI:

24 Bring focus to the `dstord.scr` screen. Click on an empty area in the screen to deselect all widgets. If the Properties window is not open, double-click on the screen or choose View→Properties Window.

25 In the Properties window, select the Display heading.

26 Select the Border property.

27 Click on the option menu to display its drop-down list, and select Yes. Press Enter or choose OK.

28 Select the System Menu property and repeat Step 27.

29 Select the Title Bar property and repeat Step 27.

30 Save the `dstord.scr` screen (press F5).

Add a new database record

You now have a working application. You can now test whether the application flow functions as designed.

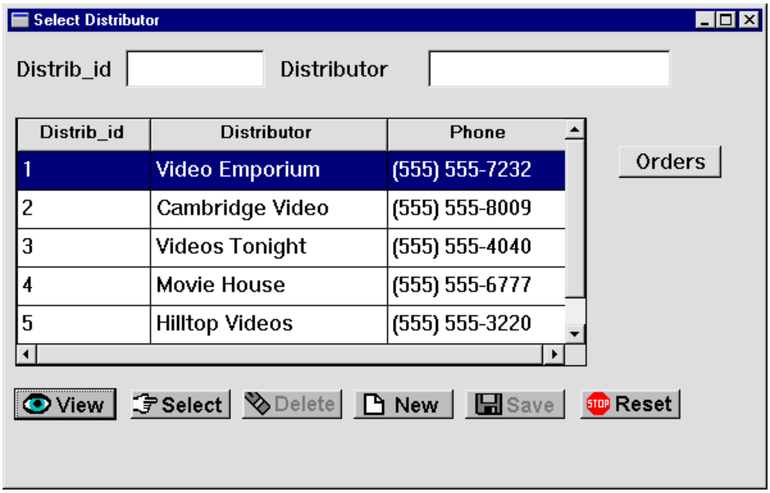
31 Bring focus to the `dstslect.scr` client screen.

32 Choose File→Test Mode or 🚦.

The Select Distributor screen opens.

33 Choose View .

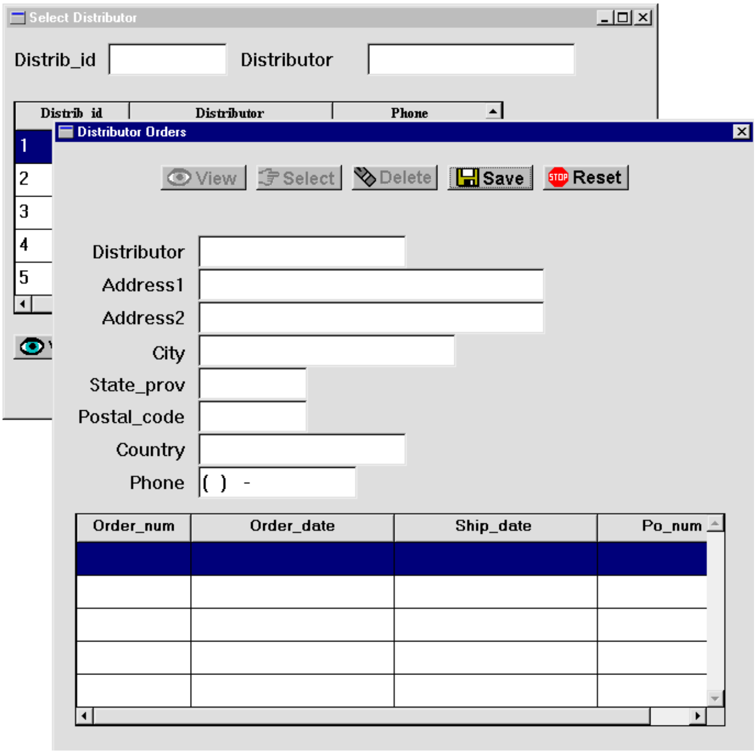
All distributors in the `vidsales` database are listed.



Distrib_id	Distributor	Phone
1	Video Emporium	(555) 555-7232
2	Cambridge Video	(555) 555-8009
3	Videos Tonight	(555) 555-4040
4	Movie House	(555) 555-6777
5	Hilltop Videos	(555) 555-3220

34 Choose New .

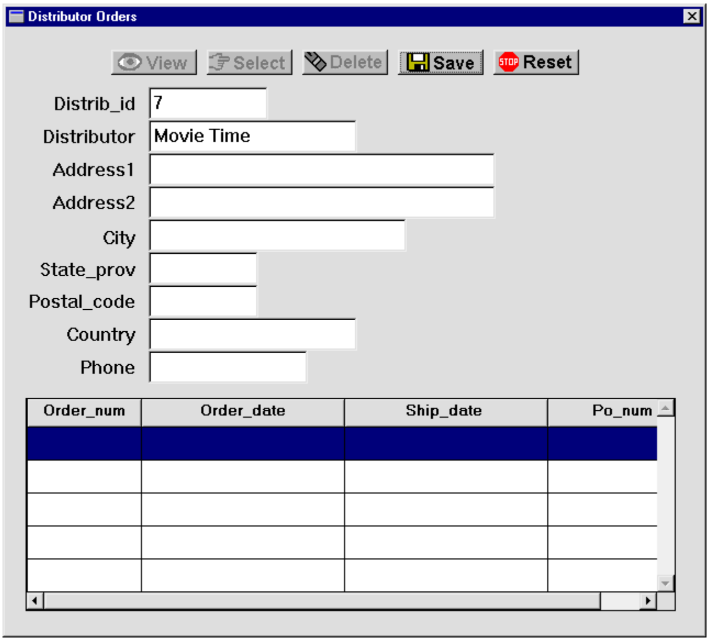
The Distributor Orders screen opens with all data fields clear. The distributor ID fields are not visible.



35 Enter `Movie Time` in the Distributor field.

36 Choose Save.

The distributor is added to the database and the ID is displayed in the `Distrib_id` field.



Distrib_id 7
Distributor Movie Time
Address1
Address2
City
State_prov
Postal_code
Country
Phone

Order_num	Order_date	Ship_date	Po_num

37 Close the Distributor Orders screen (choose Close from the system menu or press Escape).

The Select Distributor screen regains focus. If you choose View and then scroll down the list of distributors, notice that the new distributor is now 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.

38 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.

39 When done, return to the editor and save all screens (File→Save All).

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 including 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 `dstslect.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 that automatically generates a unique ID when a new distributor record is added to the database.

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.

LESSSON

10 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:

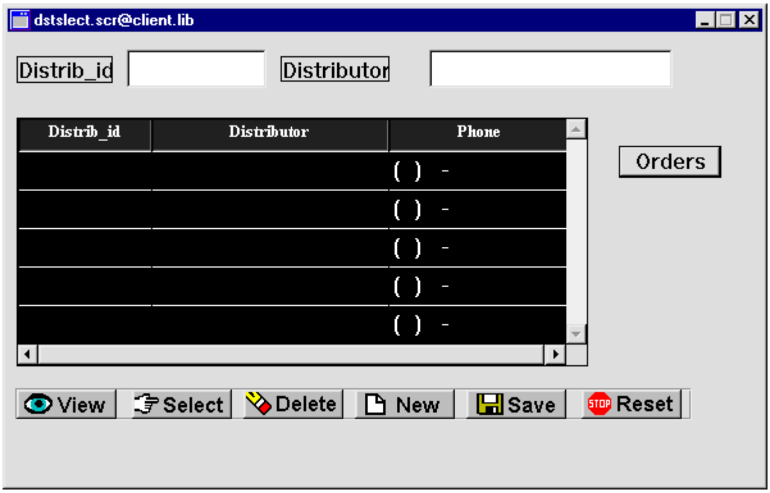
- Invoke the editor and connect to the database.
- Reopen `dstslect.scr` and `dstord.scr`.

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 `dstslect.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 Lesson 8 and Lesson 9).

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 the Double Click property to `^send_dst_data("SELECT")` for these widgets.

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()` for these columns.

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.

6 Save all open screens (press F6).

Add 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 `dstslect.scr` (Select Distributor) screen. You can enhance the screen entry procedure on the `dstslect.scr` client screen so that when the screen redisplays (after the Distributor Orders screen closes), a View command automatically executes. This causes the updated database records to redisplay.

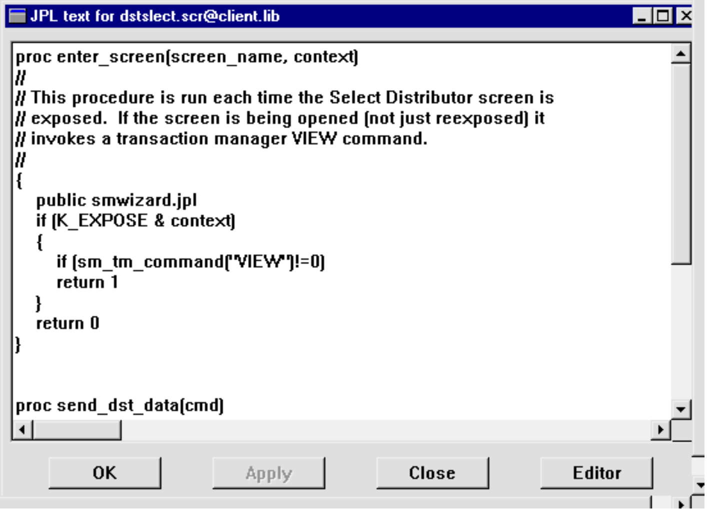
7 If necessary, reopen the JPL edit window for the `dstslect.scr` client screen:

- Select the `dstslect.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 `tutorstd.lib` by choosing Open and selecting it from the Open Library dialog.
- From the Open JPL Module dialog, select `tutorstd.lib` and select `se_slect.jpl` from its list of members. Choose OK.



```
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 `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 Apply.

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.

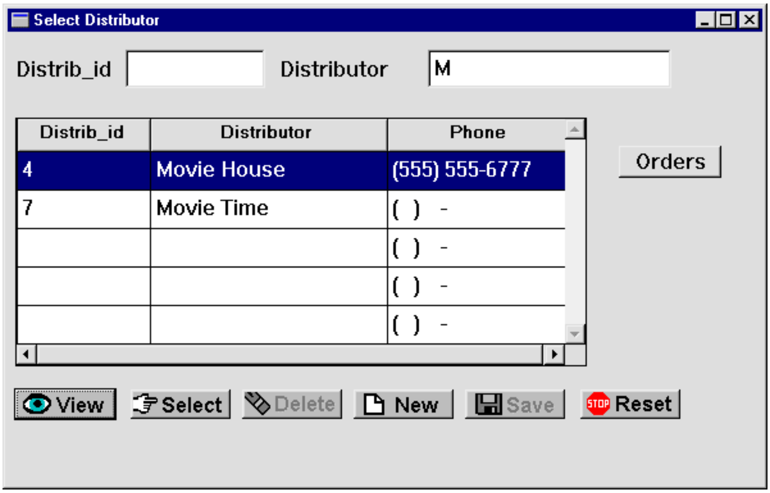
11 Bring focus to the `dstslect.scr` client screen.

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 9.

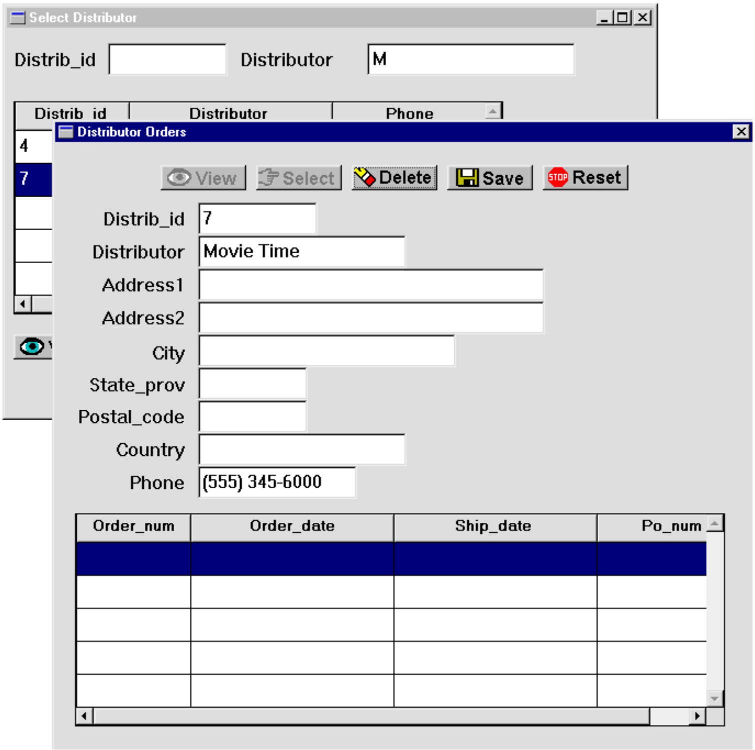


Distrib_id:
Distributor: M

Distrib_id	Distributor	Phone
4	Movie House	(555) 555-6777
7	Movie Time	( ) -
		( ) -
		( ) -
		( ) -

14 Double-click anywhere on the `Movie Time` row.

The Distributor Orders screen opens, overlaying the Select Distributor screen.



Select Distributor

Distrib_id:

Distributor: M

Distrib_id	Distributor	Phone
4		
7		

Distributor Orders

Distrib_id: 7

Distributor: Movie Time

Address1:

Address2:

City:

State_prov:

Postal_code:

Country:

Phone: (555) 345-6000

Order_num	Order_date	Ship_date	Po_num

15 Enter `(555)345-6000` in the Phone field and choose Save.

16 Close the Distributor Orders screen (choose Close from the system menu or press Escape).

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.
- Added a screen entry procedure for the `dstslect.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.

OVERVIEW

Module 4: Extending the Application

LESSSON

11 Implementing Selection Screens

Selection screens are available for the GUI interface, but not for web applications.

In Lesson 4, 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 from the database.
- Lets you generate a selection screen where users can pick from valid choices.
- Assigns the master table view of 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.
- Test the behavior of selection screens when adding new records to the database.

1 If necessary:

- Invoke the editor and connect to the database.
- Reopen the repository, `data.dic` (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 and deselect the Web Friendly Output option. Choose Next.

The First Master dialog opens.

5 Select `orders` from the list of Tables To Pick From.

6 Choose >>.

This specifies to include all columns in the `orders` table on the completed screen.

7 Choose Next.

The First Detail dialog opens.

8 Select `order_items` from the list of tables.

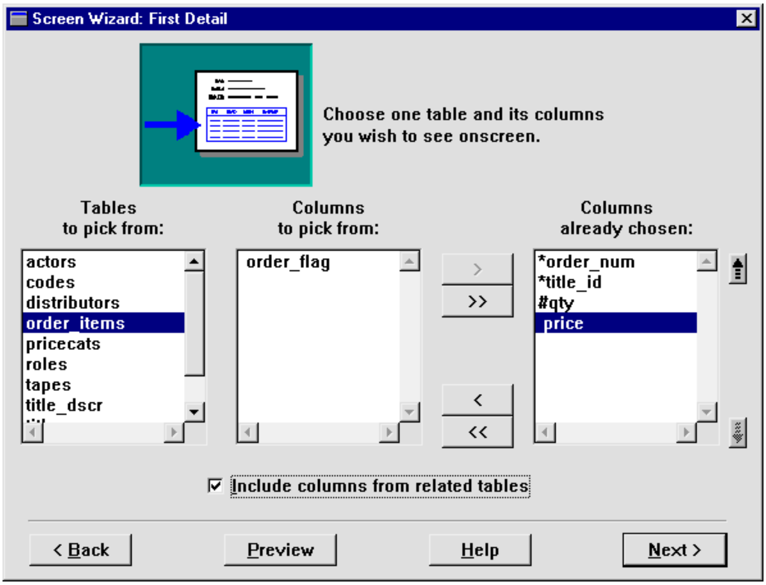
9 Double-click on `price` in the list of columns in `order_items`.

The selected columns are 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`.

10 Select the Include Columns From Related Tables check box.


Screen Wizard: First Detail

Choose one table and its columns you wish to see onscreen.

Tables to pick from:
actors
codes
distributors
order_items
pricecats
roles
tapes
title_dscr

Columns to pick from:
order_flag

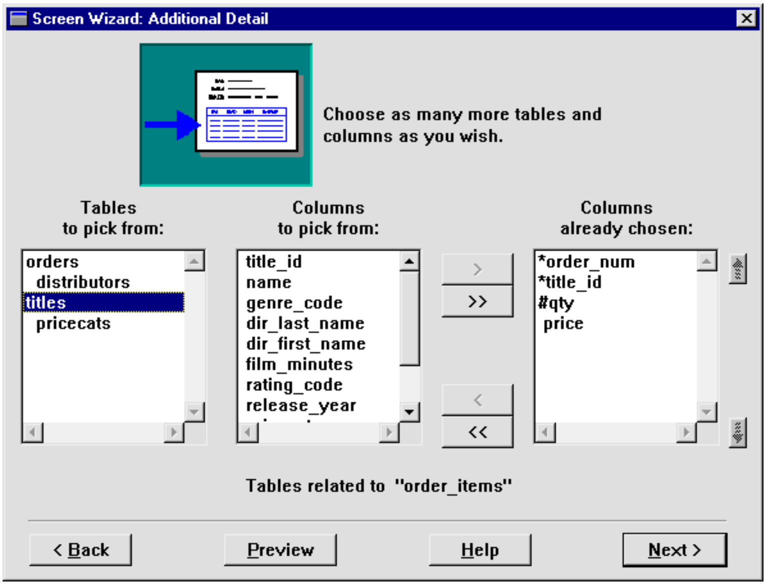
Columns already chosen:
*order_num
*title_id
#qty
price

☑ Include columns from related tables

11 Choose Next.

The Additional Detail dialog opens and shows which tables can be joined to the `order_items` table.

12 Select `titles` from the list of tables.


Screen Wizard: Additional Detail
Choose as many more tables and columns as you wish.
Tables to pick from:
Columns to pick from:
Columns already chosen:
>
>>
<
<<
Tables related to "order_items"
< Back
Preview
Help
Next >

Tables to pick from:

orders
 distributors
titles
 pricecats

Columns to pick from:

title_id
name
genre_code
dir_last_name
dir_first_name
film_minutes
rating_code
release_year

Columns already chosen:

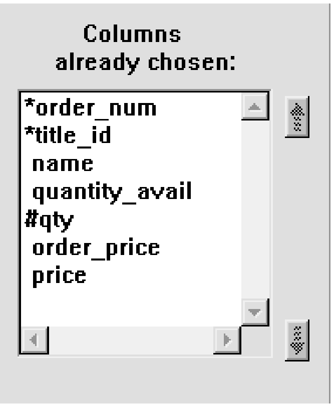
*order_num
*title_id
#qty
 price

The list of columns belonging to the `titles` table displays.

13 Select (Ctrl+click) `name`, `order_price`, and `quantity_avail`.

14 Choose >

15 Reorder the columns (select items and use the up/down position arrows) as shown:



*order_num
*title_id
 name
 quantity_avail
#qty
 order_price
 price

16 Choose Next.

The Layout Selection dialog opens.

17 Choose Next to accept the default layout specification: single row for the master and grid display for the detail.

The Style and Finish dialog opens.

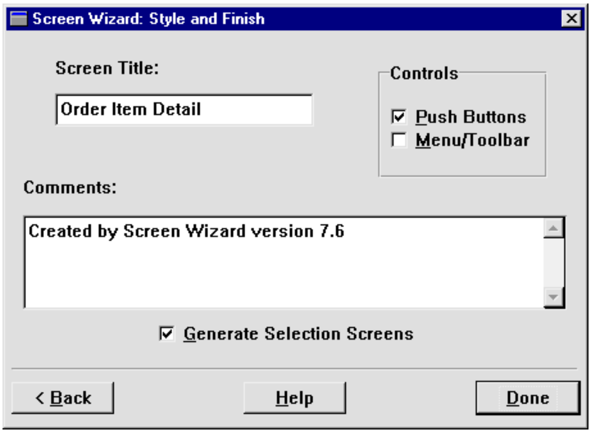
Generate selection screens

When you include columns from additional tables, the screen wizard lets you decide whether to generate selection screens. 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.

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.



Screen Title: Order Item Detail

☑ Push Buttons

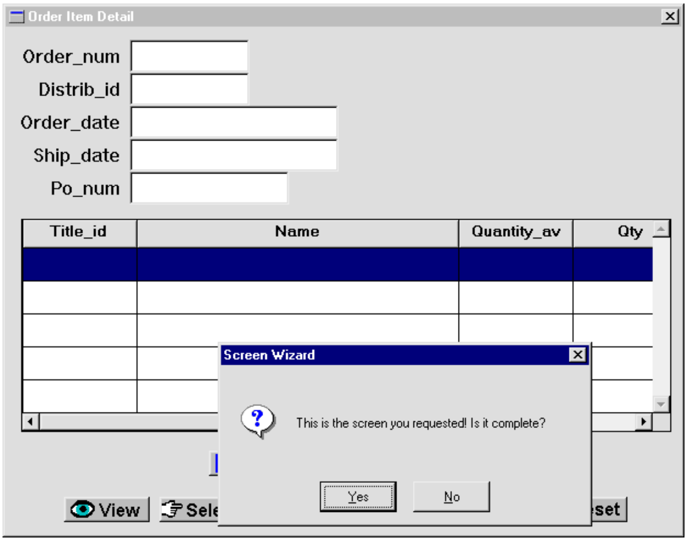
□ Menu/Toolbar

Comments: Created by Screen Wizard version 7.6

☑ Generate Selection Screens

18 The Generate Selection Screens check box is selected by default. Change the screen title to `Order Item Detail` and choose Done.

A preview of the client screen displays:



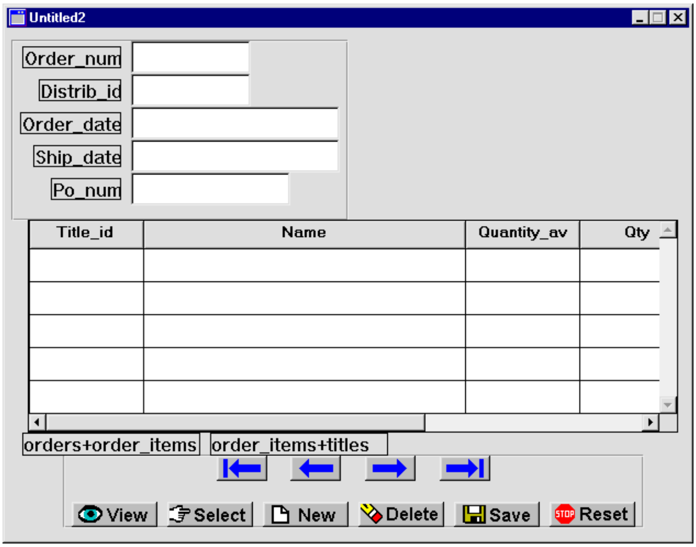
This is the screen you requested! Is it complete?

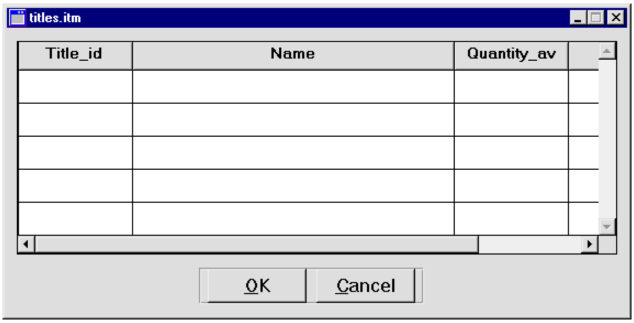
19 Choose Yes to confirm the contents of the screen.

When the wizard finishes building the screens (notice the status bar), the results include two screens:

- Client screen
- Selection screen for the `titles` table `titles.itm`

You need to move or minimize the top screen to see the one beneath it.





Save the wizard output

Save the screens to `client.lib`.

20 Bring focus to the client screen and choose Save.

The Save Screen dialog opens.

21 Save the screen as `orditm.scr` in library `client.lib`.

22 Save `titles.itm` to `client.lib`. You can close the screen after saving.

Test the selection screen

Now test the screens.

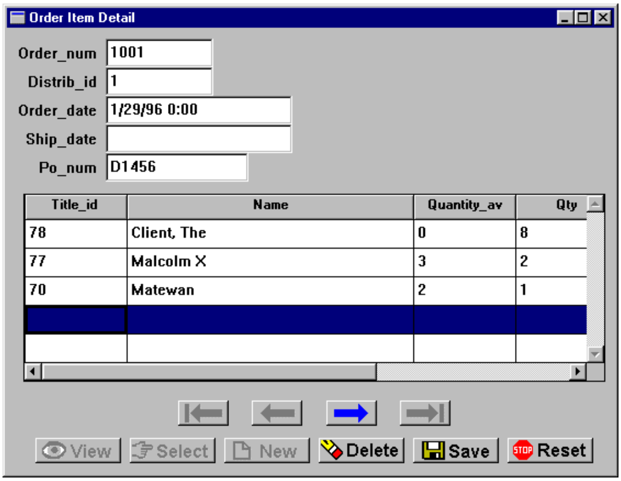
23 Give focus to client screen `orditm.scr` .

24 Choose File→Test Mode or .

The Order Item Detail screen opens.

25 Choose Select .

The first record in the `orders` table displays. Several `order_item` records are associated with this order. They are displayed in the grid.



Order Item Detail

Order_num 1001

Distrib_id 1

Order_date 1/29/96 0:00

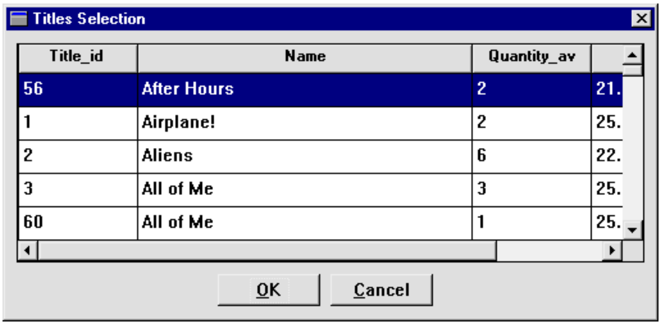
Ship_date

Po_num D1456

Title_id	Name	Quantity_av	Qty
78	Client, The	0	8
77	Malcolm X	3	2
70	Matewan	2	1

26 Click in the `Title_id` field of the first empty row in the grid.

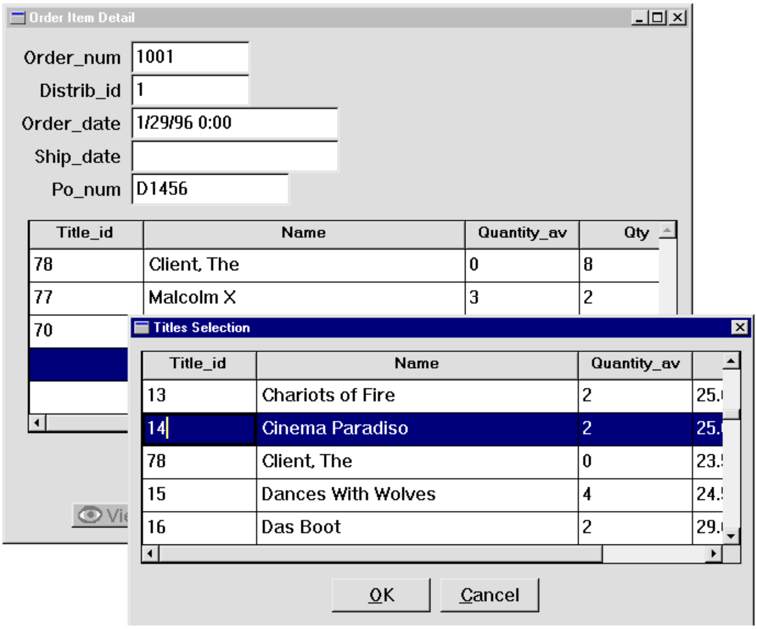
27 Press F1 (or HELP).



Title_id	Name	Quantity_av	
56	After Hours	2	21.
1	Airplane!	2	25.
2	Aliens	6	22.
3	All of Me	3	25.
60	All of Me	1	25.

The Titles Selection screen opens and displays all video records in the `titles` database table.

28 Scroll down to the title `Cinema Paradiso` and double-click on the ID to select that video.



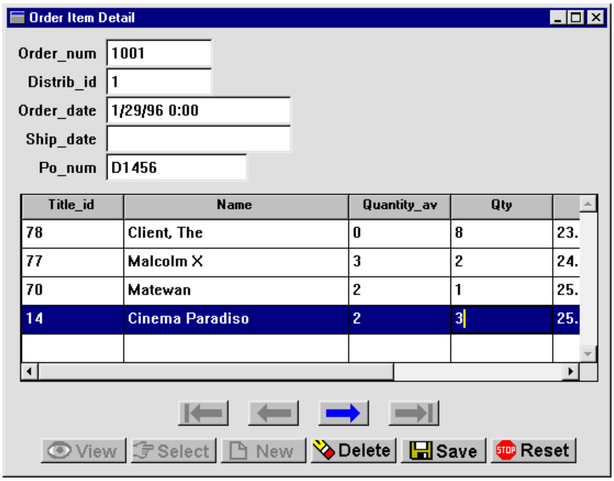
Order_num 1001
Distrib_id 1
Order_date 1/29/96 0:00
Ship_date
Po_num D1456

Title_id	Name	Quantity_av	Qty
78	Client, The	0	8
77	Malcolm X	3	2
70			

Title_id	Name	Quantity_av	
13	Chariots of Fire	2	25.
14	Cinema Paradiso	2	25.
78	Client, The	0	23.
15	Dances With Wolves	4	24.
16	Das Boot	2	29.

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`).

29 Enter `3` in the Qty column.



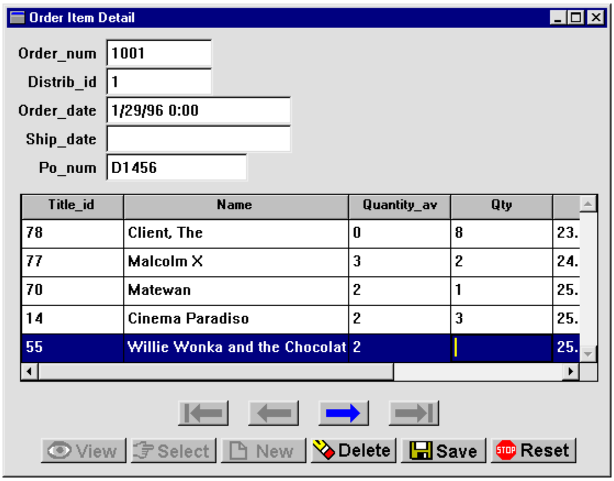
Order_num 1001
Distrib_id 1
Order_date 1/29/96 0:00
Ship_date
Po_num D1456

Title_id	Name	Quantity_av	Qty	
78	Client, The	0	8	23.
77	Malcolm X	3	2	24.
70	Matewan	2	1	25.
14	Cinema Paradiso	2	3	25.

Validate the data

Add another item to this order, but this time, enter the title identification number. Panther uses a validation routine to ensure that the entry is valid.

30 Press TAB to advance to the next empty row and type `55` in the `Title_id` column. Press Enter or TAB.



Order_num 1001
Distrib_id 1
Order_date 1/29/96 0:00
Ship_date
Po_num D1456

Title_id	Name	Quantity_av	Qty	
78	Client, The	0	8	23.
77	Malcolm X	3	2	24.
70	Matewan	2	1	25.
14	Cinema Paradiso	2	3	25.
55	Willie Wonka and the Chocolat	2		25.

`Willie Wonka` and the `Chocolate Factory` is the name of the video associated with the specified ID number.

31 Choose Save.

This action saves the new items to the selected distributor's order.

32 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 routine 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 routines needed to validate data and populate selection screens.

LESSSON

12 Calculating Data from Database Values

For this lesson, you are provided with an enhanced client screen based on the Order Item Detail screen that you created in Lesson 11. 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.

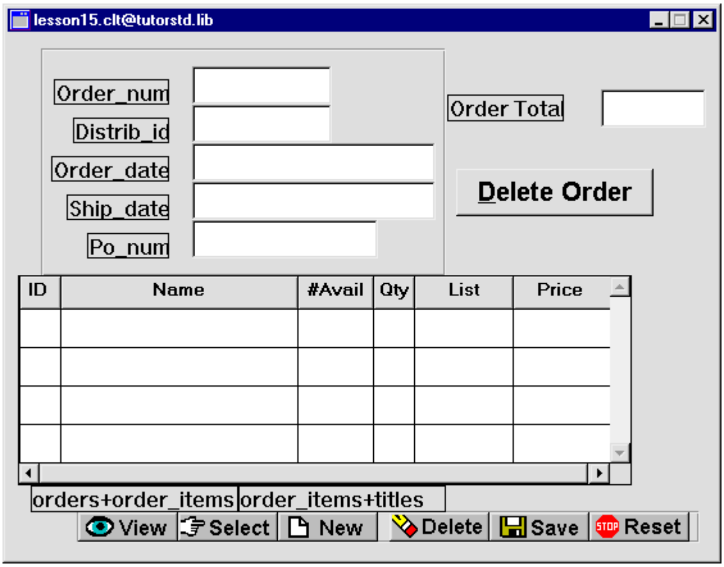
In this lesson you learn how to:

- 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;

   - Invoke the editor and connect to the database.
   - Reopen `tutorstd.lib` (File→Open→Library).

2. Open `lesson15.clt` from `tutorstd.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 library `client.lib`.

   This prompts you to overwrite the `orditm.scr` screen that you created in Lesson 11.

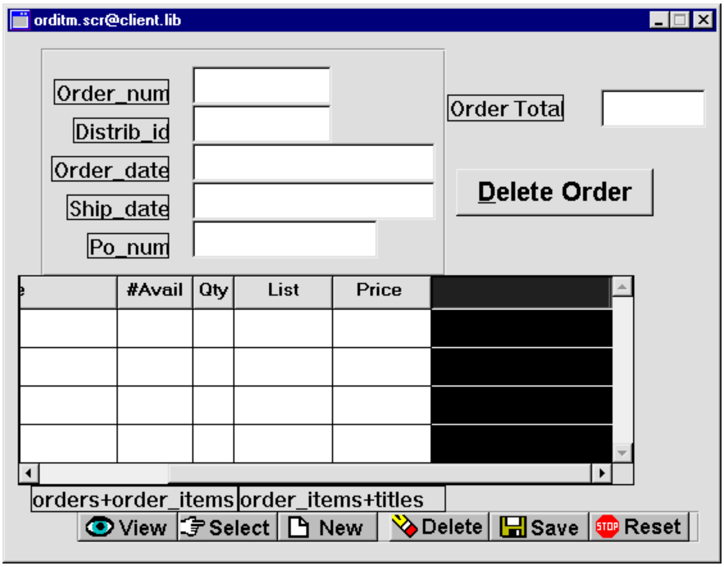
   Note: If you prefer not to overwrite the original `orditm.scr` client screen, open it and save it under a different name before saving the new version of `orditm.scr`.

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 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, refer to Chapter 45, “Customizing the User Interface,” in *Application Development Guide*.

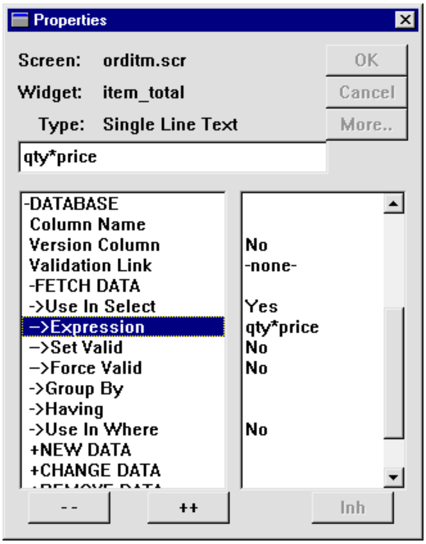
Define a math expression

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.

8 Under Database, under FETCH DATA, set the Use In Select property to Yes.

Related subproperties are displayed.

9 In the Expression subproperty, enter: qty*price



Screen: orditm.scr
Widget: item_total
Type: Single Line Text

qty*price

-DATABASE	
Column Name	
Version Column	No
Validation Link	-none-
-FETCH DATA	
->Use In Select	Yes
->Expression	qty*price
->Set Valid	No
->Force Valid	No
->Group By	
->Having	
->Use In Where	No
+NEW DATA	
+CHANGE DATA	
+REMOVE DATA	

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. It shows the widgets on the current screen: the widget's name, field number or contents is in the middle column and its widget type in the right column.

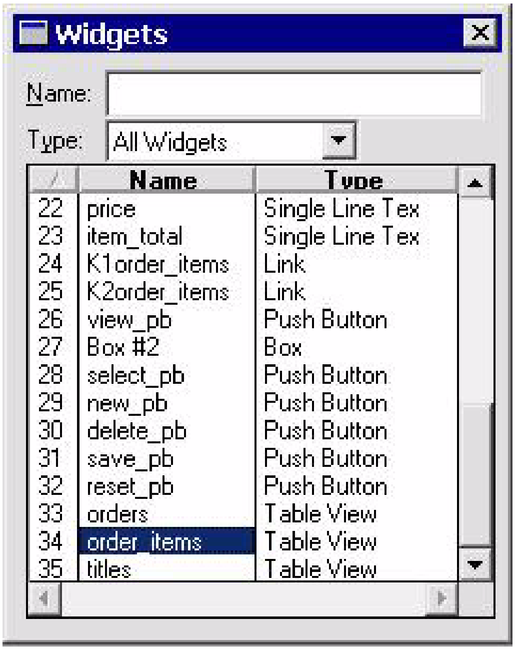
More About the Widget List

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.



Widgets

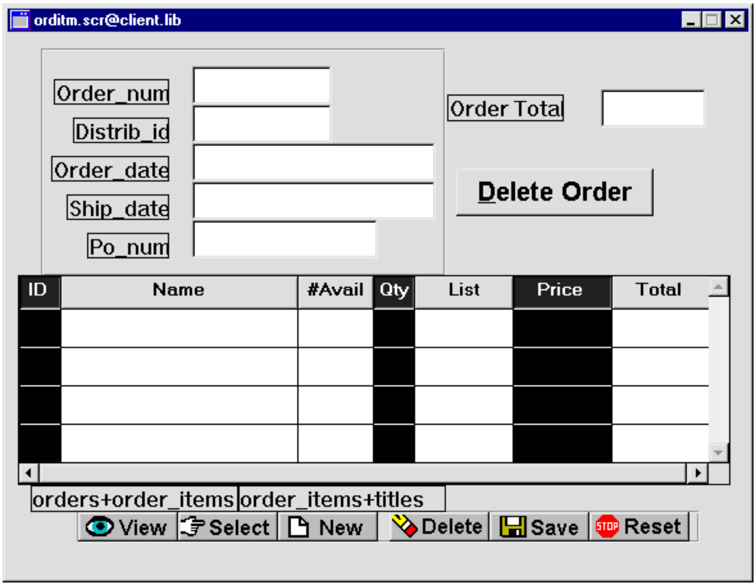
Name:

Type: All Widgets

	Name	Type
22	price	Single Line Tex
23	item_total	Single Line Tex
24	K1order_items	Link
25	K2order_items	Link
26	view_pb	Push Button
27	Box #2	Box
28	select_pb	Push Button
29	new_pb	Push Button
30	delete_pb	Push Button
31	save_pb	Push Button
32	reset_pb	Push Button
33	orders	Table View
34	order_items	Table View
35	titles	Table View

If the Properties window displays the table view properties, it confirms the table view is selected.

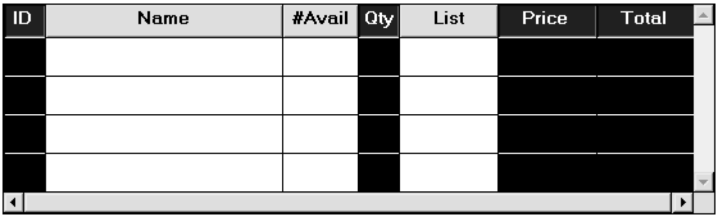
12 Choose Edit→Group→Select Members.



All members of the `order_items` table view are selected: ID (`title_id`), Qty (`qty`), and Price (`price`).

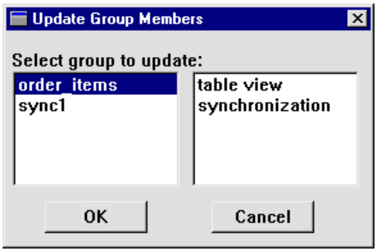
13 Add the `item_total` (Total) grid member to the selection set:

- *Windows* Shift+click on the column's title.
- *Motif* Shift+click within a cell in the column.



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 Step 11 and Step 12. The Total grid member should be selected along with `title_id`, `qty`, and `price`. If it is not, repeat Step 13 - Step 15.

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 widgets 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.

Calculate results

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.

16 Select orditm.scr (deselect all widgets).

17 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.

18 Scroll to the bottom of the JPL edit window and type in the upd_order_totals procedure defined below.

```
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.

19 Scroll to the bottom of the JPL edit window and insert `evnt_ord_clt.jpl` from `tutorstd.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.

20 Save the JPL. Choose Apply.

21 Return to the `orditm.scr` client screen and select its root table view `orders`, via the DB Interactions window or Widget List.

22 Under Transaction, enter `tm_events_clt` in the Function property.

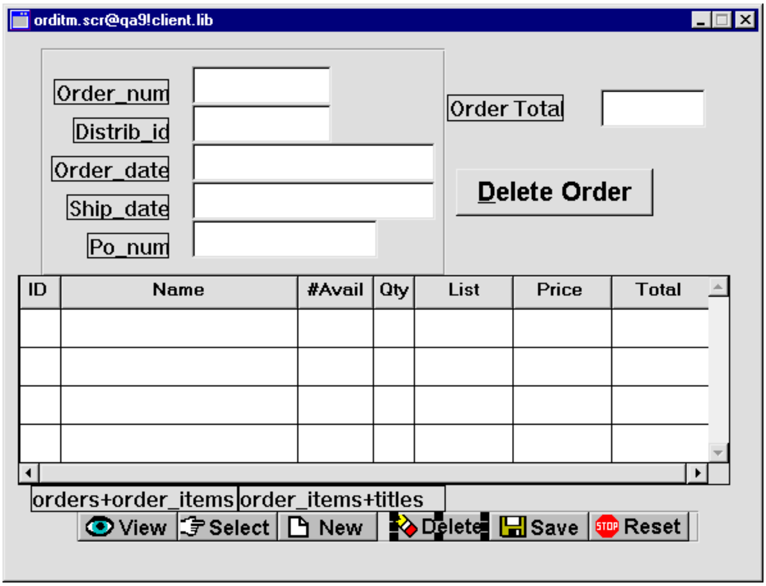
The root table view now has `tm_events_clt` set as its hook function. The transaction manager executes this function when it starts traversing the screen's table views.

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.

23 Select the Delete button at the bottom of the `orditm.scr` client screen.



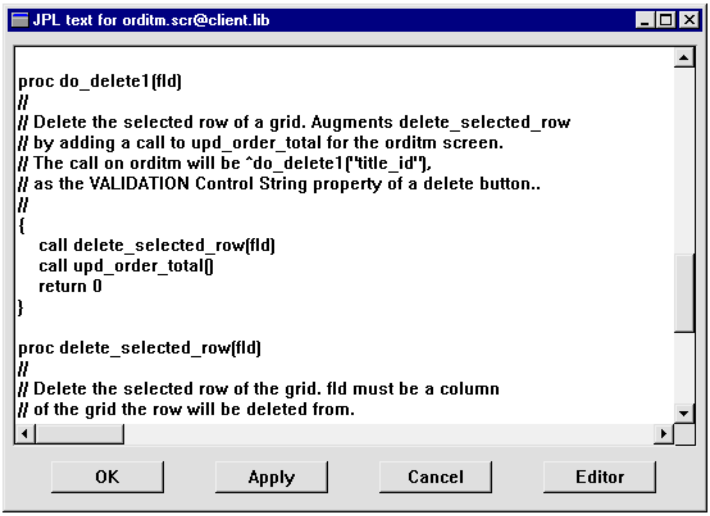
24 Change the widget's name to `delete1_pb`.

25 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.

26 Return to the JPL edit window.

27 Scroll to the bottom and insert `delete1.jpl` from the `tutorstd.lib` library.



```
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_gofield (fld, occ)
    call sm_i_doccur(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_doccur` 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.

28 Save the JPL file. Choose Apply.

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.

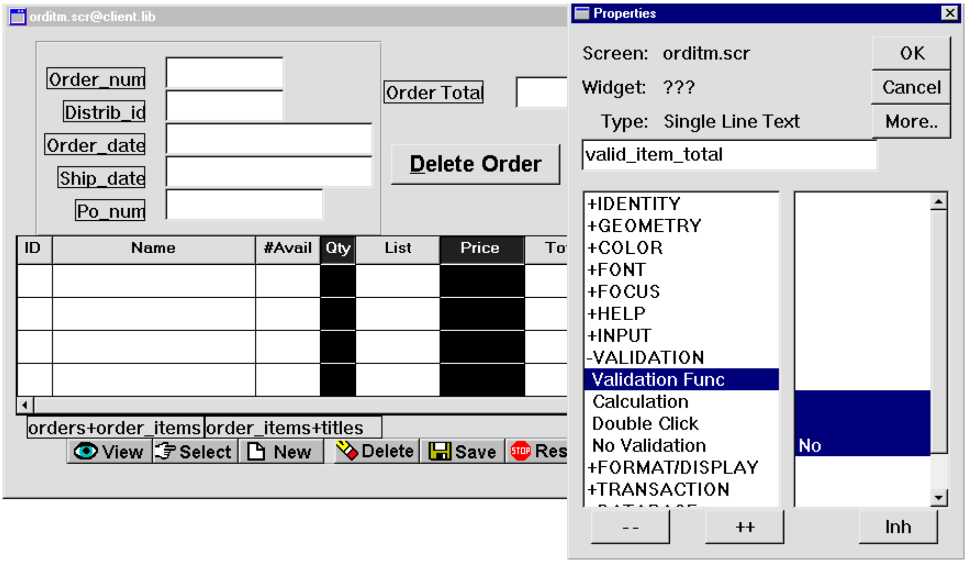
Validation functions are also called under the following conditions:

- 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.

Refer to Chapter 17, “Understanding Application Events,” in *Application Development Guide* for more information about application events.

29 Select the Qty (`qty`) and Price (`price`) grid members.

30 Under Validation, enter `valid_item_total` in the Validation Func property.



Screen: orditm.scr
Widget: ???
Type: Single Line Text
valid_item_total
Validation Func
No

31 Return to the JPL edit window.

32 Scroll to the bottom of the window and insert `order_valid.jpl` from the `tutorstd.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
}
```

More About the valid_item_total Procedure

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.

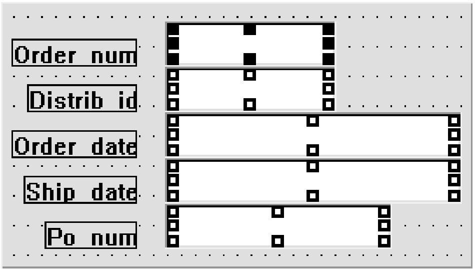
33 Choose Apply to save all changes in the JPL edit window.

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`.

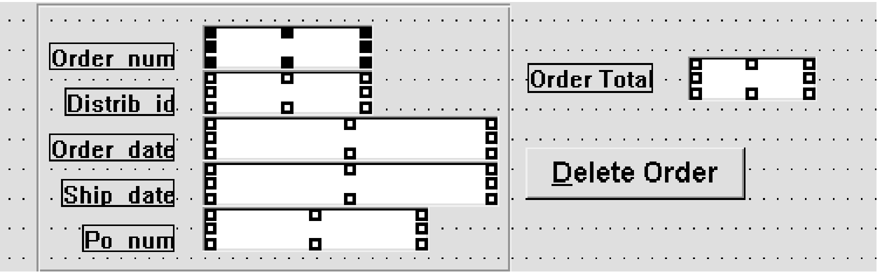
34 Select the `orders` table view with the Widget List or the DB Interactions window.

35 Choose Edit→Group→Select Members.



All members on the screen that belong to the `orders` table view are selected.

36 Add the `order_total` single line text widget to the selection group (Shift+click).



37 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.

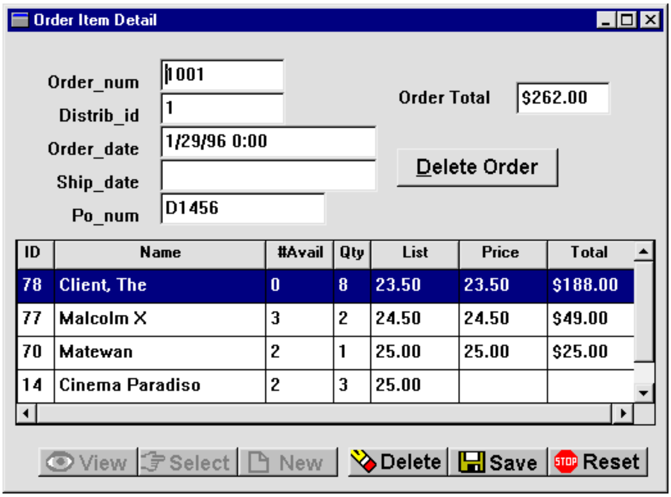
38 Save `orditm.scr`.

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.

39 Choose File→Test Mode (press F2) or .

The first order record is displayed.



Order_num	1001
Distrib_id	1
Order_date	1/29/96 0:00
Ship_date	
Po_num	D1456
Order Total	$262.00

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		

Item totals and the order total are calculated on screen. Rows that lack quantity or price data also omit total data.

40 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.

41 Click in the row with the ID 70 and choose Delete.

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.

42 Choose Save.

The database is updated with the changed data in order 1001.

43 Choose Reset.

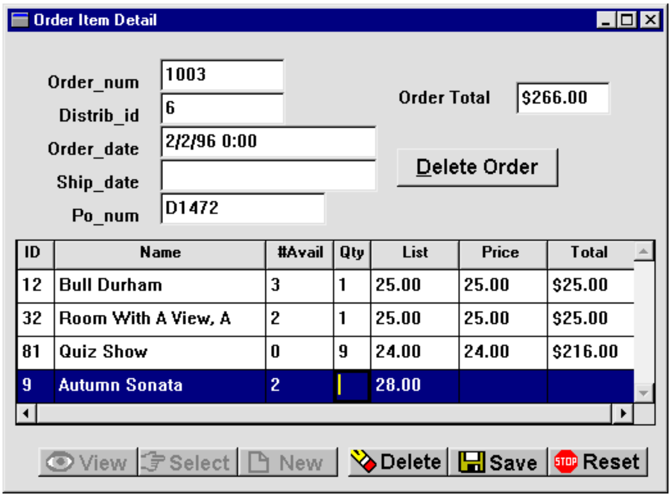
All fields including `order_total` are cleared of data.

44 Type `1003` in the `order_num` field and choose Select.

The order associated with distributor 6 displays.

45 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.



Order Item Detail

Order_num 1003

Distrib_id 6

Order_date 2/2/96 0:00

Ship_date

Po_num D1472

Order Total $266.00

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		

46 Enter `2` for the quantity, press `TAB`, and enter `25.00` in the `price` field. Press `TAB` again.

The totals are immediately recalculated.

47 Choose Save .

48 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.

49 Open the `dstord.scr` client screen from `client.lib`.

50 With the screen selected, under Focus, select the JPL Procedures property.

51 Scroll to the bottom of the JPL edit window and insert `send_order.jpl` from `tutorstd.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
}
```

More About the send_order_data Procedure

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 10). The data required to execute the appropriate SQL is sent and the orditm.scr screen opens.

52 Choose Apply to save the JPL procedure.

53 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. 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.

LESSSON

13 The Finale

Congratulations! You have created a fully functional Panther application.

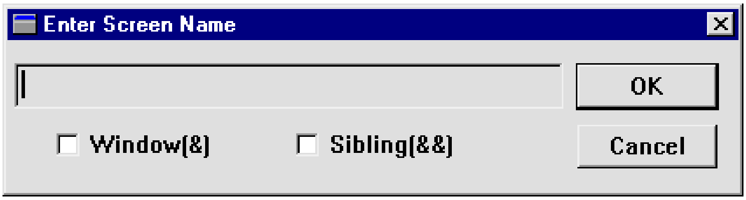
So, now take it from the top!

1 Invoke the editor and connect to the database.

2 Choose File⟶Test Mode (press F2) or .

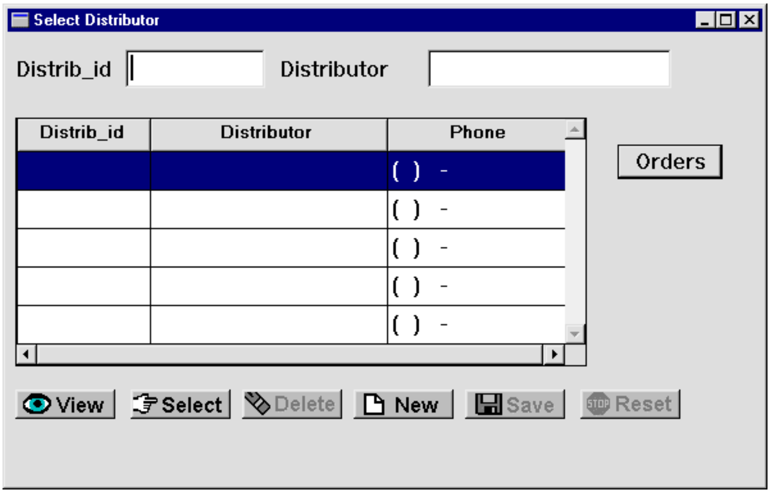
3 Choose Options⟶Open Screen.

The Enter Screen Name dialog box opens.



4 Enter `dstslect.scr` and choose OK.

The Select Distributor screen opens.

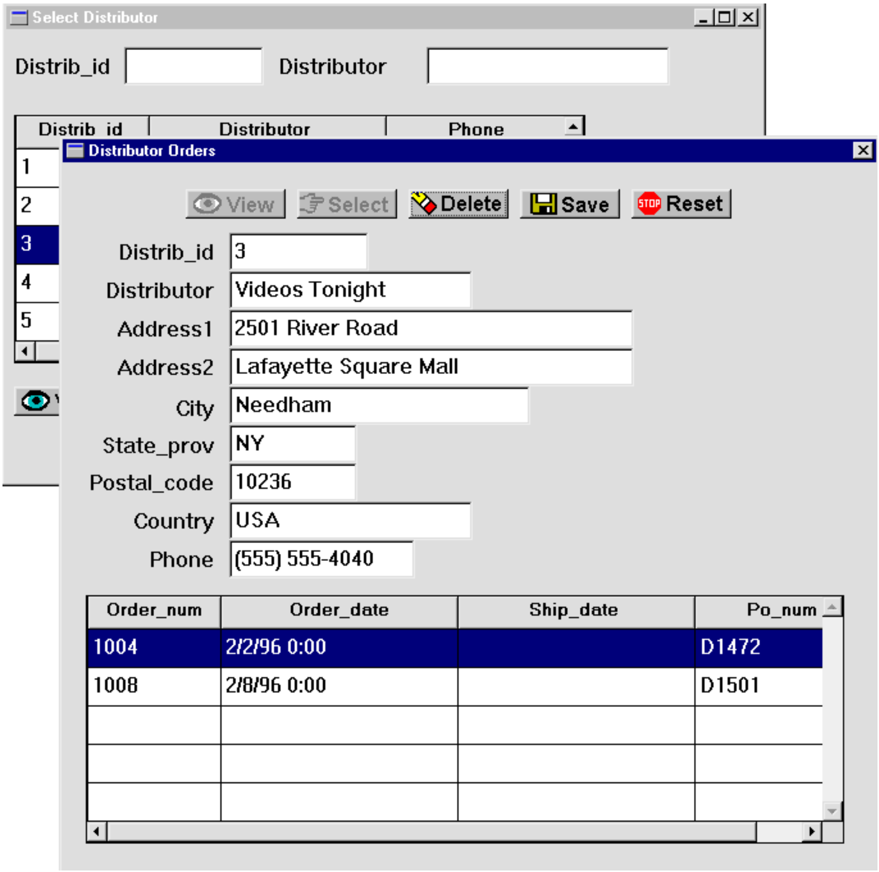


Distrib_id	Distributor	Phone
		( ) -
		( ) -
		( ) -
		( ) -
		( ) -

5 Choose View .

All distributors in the `vidsales` database are displayed.

6 Scroll down the list and double-click on the row for distributor ID `3`. The Distributor Orders screen opens and displays the information associated with the distributor.



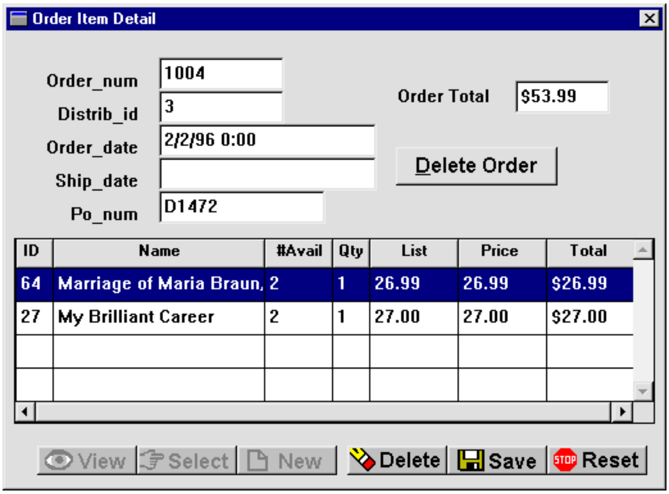
Distrib_id Distributor

Distrib_id	Distributor	Phone
1		
2		
3		
4		
5		

Distrib_id	3
Distributor	Videos Tonight
Address1	2501 River Road
Address2	Lafayette Square Mall
City	Needham
State_prov	NY
Postal_code	10236
Country	USA
Phone	(555) 555-4040

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

7 Double-click on the `Order_num` `1004`.



Order Item Detail

Order_num: 1004

Distrib_id: 3

Order_date: 2/2/96 0:00

Ship_date:

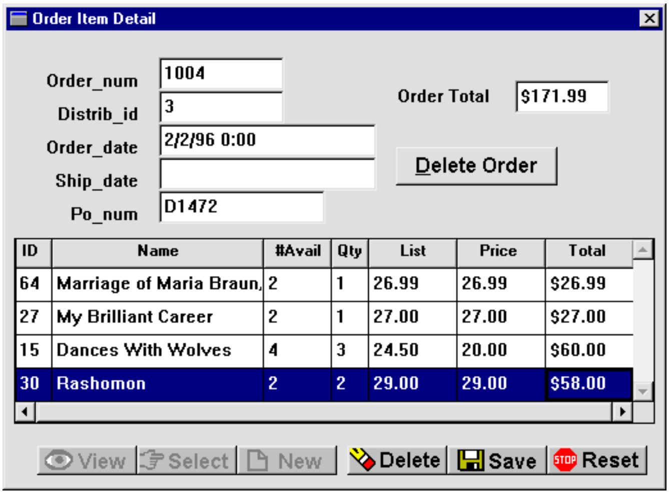
Po_num: D1472

Order Total: $53.99

ID	Name	#Avail	Qty	List	Price	Total
64	Marriage of Maria Braun,	2	1	26.99	26.99	$26.99
27	My Brilliant Career	2	1	27.00	27.00	$27.00

8 Add the following new order items:

ID	Qty	Price
15	3	20.00
30	2	29.00



Order Item Detail

Order_num: 1004

Distrib_id: 3

Order_date: 2/2/96 0:00

Ship_date:

Po_num: D1472

Order Total: $171.99

ID	Name	#Avail	Qty	List	Price	Total
64	Marriage of Maria Braun,	2	1	26.99	26.99	$26.99
27	My Brilliant Career	2	1	27.00	27.00	$27.00
15	Dances With Wolves	4	3	24.50	20.00	$60.00
30	Rashomon	2	2	29.00	29.00	$58.00

9 Press `TAB` between each entry. The totals are updated appropriately.

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_order_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!

APPENDIX

A Setting Up the Tutorial

The tutorial steps assume that the Panther development client 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`.

Figure A-1 illustrates a typical configuration for two-tier UNIX development clients and web servers.

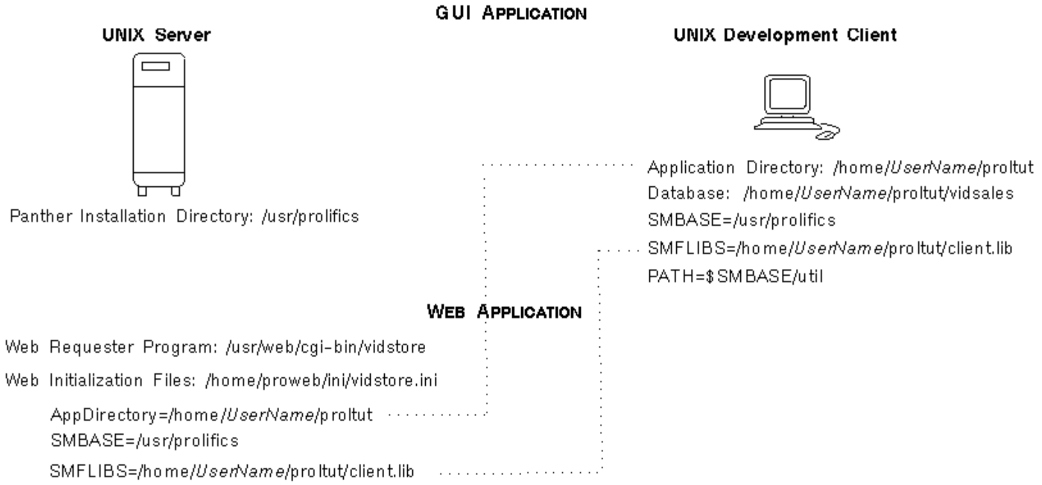
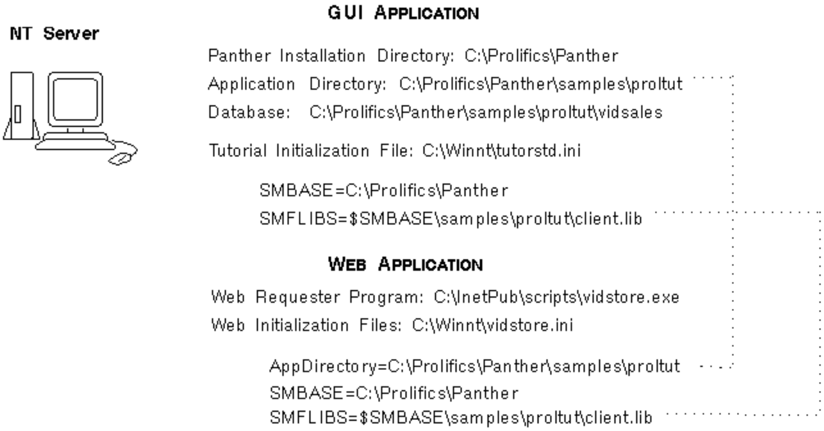

GUI APPLICATION
UNIX Server
UNIX Development Client
Panther Installation Directory: /usr/prolifics
Application Directory: /home/*UserName*/proltut
Database: /home/*UserName*/proltut/vidsales
SMBASE=/usr/prolifics
SMFLIBS=/home/*UserName*/proltut/client.lib
PATH=$SMBASE/util
WEB APPLICATION
Web Requester Program: /usr/web/cgi-bin/vidstore
Web Initialization Files: /home/proweb/ini/vidstore.ini
AppDirectory=/home/*UserName*/proltut
SMBASE=/usr/prolifics
SMFLIBS=/home/*UserName*/proltut/client.lib

Figure A-1 Typical UNIX configuration

Figure A-2 illustrates a typical configuration for two-tier Windows development clients and web servers.



GUI Application

Panther Installation Directory: C:\Prolifics\Panther

Application Directory: C:\Prolifics\Panther\samples\proltut

Database: C:\Prolifics\Panther\samples\proltut\vidsales

Tutorial Initialization File: C:\Winnt\tutorstd.ini

```
SMBASE=C:\Prolifics\Panther
SMFLIBS=$SMBASE\samples\proltut\client.lib
```

Web Application

Web Requester Program: C:\InetPub\scripts\vidstore.exe

Web Initialization Files: C:\Winnt\vidstore.ini

```
AppDirectory=C:\Prolifics\Panther\samples\proltut
SMBASE=C:\Prolifics\Panther
SMFLIBS=$SMBASE\samples\proltut\client.lib
```

Figure A-2 Typical Windows configuration

A Windows development client can be used with a UNIX Panther web application server if:

- The client library and database are available on, or are copied to, a directory on the UNIX machine.
- A Panther web application (Lesson 2) specifies that directory as its application directory.

Figure A-3 illustrates this configuration.

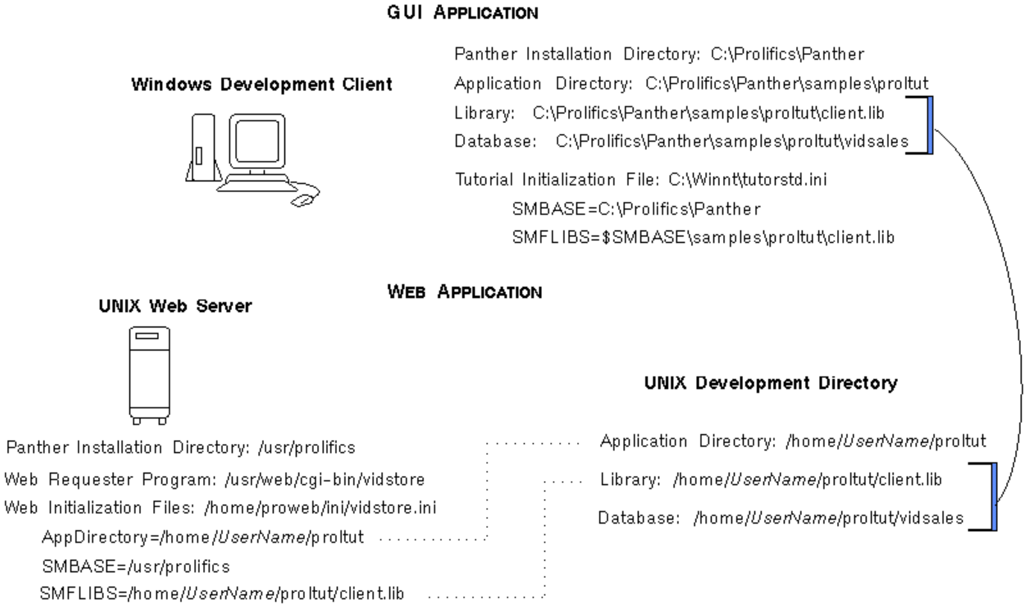

GUI APPLICATION
Windows Development Client
Panther Installation Directory: C:\Prolifics\Panther
Application Directory: C:\Prolifics\Panther\samples\proltut
Library: C:\Prolifics\Panther\samples\proltut\client.lib
Database: C:\Prolifics\Panther\samples\proltut\vidsales
Tutorial Initialization File: C:\Winnt\tutorstd.ini
SMBASE=C:\Prolifics\Panther
SMFLIBS=$SMBASE\samples\proltut\client.lib
WEB APPLICATION
UNIX Web Server
Panther Installation Directory: /usr/prolifics
Web Requester Program: /usr/web/cgi-bin/vidstore
Web Initialization Files: /home/proweb/ini/vidstore.ini
AppDirectory=/home/*UserName*/proltut
SMBASE=/usr/prolifics
SMFLIBS=/home/*UserName*/proltut/client.lib
UNIX Development Directory
Application Directory: /home/*UserName*/proltut
Library: /home/*UserName*/proltut/client.lib
Database: /home/*UserName*/proltut/vidsales

Figure A-3 Windows development clients with a UNIX web application server

APPENDIX

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.

Program Startup

By default, Java is initialized on program startup. (You can change this initialization by setting one of the behavior variables, `JAVA_USE`.)

Windows clients having an old version of the Java DLLs display an error message when starting Panther, `Java Not Supported`. To update the Java DLLs, run the Microsoft Java executable in `$SMBASE\jvm`.

Tutorial Errors

If you try to select data or enter new data without a database connection, two error messages will be displayed, one from the database interface and one from the transaction manager.

```
Database Interface Error:  Error executing statement
   Error Number :  53271
   Error Message:  No connection active.
```

```
Error: error in common model dm_common_model event TM_TEST_ERROR:
   Error in User Hook Function or Transaction Model
```

Declare a database connection using Database→Connect in test mode or File→Open→Database in the editor.

Web Applications

If a web application's cache files have been deleted, Panther displays the following error message:

```
Your cached data is no longer available.
```

Press the web browser's Refresh/Reload button to redisplay the specified screen.

Index