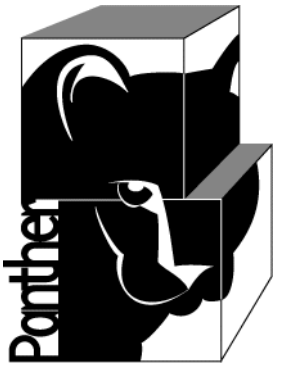
Contents:



Panther

COM/MTS Guide

Prolifics

Release 5.51

Document 0404
March 2017

Copyright

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:

3. Building COM Components

4. Building Client Screens

5. Deploying COM Components

A. COM/MTS Utilities

B. COM Samples

About This Document

The *COM/MTS Guide*, aimed primarily at developers building Panther COM components and client screens that call COM components, contains the following information:

- An overview of COM.
- Setup requirements for the Panther development environment.
- How to use the Panther editor to build COM components.
- How to call COM components from your client application.
- How to deploy Panther COM components under COM, DCOM, and MTS.
- Description of command-line utilities that can be used in a Panther COM application.
- Description of the COM samples in the distribution.
- Checklist for COM component deployment.

Related Information

- For more information in general about COM technologies, refer to the Microsoft site at http://www.microsoft.com/.

Documentation Website

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

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

How to Print the Document

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

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

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.

`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`

Convention	Item
`{ }`	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]...`
. . .	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

CHAPTER

1 Overview

Panther's COM support lets you:

- Use ActiveX controls.
- Build COM components in the Panther editor.
- Use third-party components in a Panther application.
- Use Panther-built components in third-party clients.
- Use Active Document Servers.
- Deploy your COM components using COM, DCOM (Distributed COM), or MTS (Microsoft Transaction Server).

The Panther editor makes it easy to create COM components by building the necessary files and interfaces from your method and property definitions.

What is COM and DCOM?

COM (Component Object Model) is Microsoft's framework for building modular software components. COM allows software components to communicate with each other and provides the following advantages:

- Local and remote transparency. COM components can be located on different machines in the same network.

- Language and tool independence. COM components can be built by different tools and written in different programming languages.
- Object-orientation. COM components are modular and implement a well-defined set of interfaces.
- Version control. Each COM component has a unique identifier. Later versions can replace earlier versions of the component.
- Scalability. The COM components within a single application, even a single transaction, can be partitioned across any number of component packages.

DCOM (Distributed COM) allows you to call COM components installed on another network machine. Since the location of the COM component is determined by the system registry, not in the COM component itself, you can develop COM components without needing to know where the components will be deployed.

What is MTS?

MTS (Microsoft Transaction Server) provides transaction support for COM components. In a database application, you often need to make a group of changes to the database at the same time. For example, in a banking application, if you transfer money from one account to another, the amount must be debited from one account and credited to the other account as a single operation. This is called a transaction. If both changes occur successfully, the transaction can be committed, or saved, to the database. If one or both of the changes fail, the transaction needs to be rolled back.

MTS allows you to set properties on each component to ensure that the component is running inside a transaction. In addition, MTS provides support for:

- Database connection pooling. One of the major benefits of three-tier applications is that the application server maintains the database connections, not application clients.
- Multi-threaded applications. Using threads instead of processes improves application performance.

- Security for component access. Defining roles on the MTS server determines which users can access the COM components in a given package.

COM Features

Unique Identifiers

When you create a COM component, Panther automatically assigns it a `CLSID` (class identifier) based on your network card ID, the current time, and other data. (Another term for `CLSID` is `GUID`—globally unique identifier.) When a COM component is registered on your machine, its `CLSID` is written into the system registry.

In addition to identifying the COM component, the `CLSID` has another use. Whenever you modify a component's interface, you should assign the component a new `CLSID`. This guarantees the behavior of COM components. If you find COM components with the same `CLSID`, you then know that both components have the same interfaces, properties, and methods.

Defined Interfaces

When you create a COM component, you define the interfaces that the component will support. An interface is a collection of:

- Methods (or functions) that you can call in order for the component to complete some action.
- Properties that you can query and set.

All COM components support the IUnknown interface; one of the methods in the IUnknown interface, QueryInterface, allows you to ask the COM component which other interfaces it supports. Since COM components are encapsulated, not allowing access to their internal implementation, interfaces are the only way to access a COM component.

Panther-built COM components support IDispatch, the “automation” interface, to allow easy access by scripting languages (JPL, VBScript, JavaScript) and expose a dual interface (for early or late binding). For more information about defining the component's interface in the Panther editor, refer to page 3-3, “Defining the Component's Interface.”

Client Access

Once a COM component is built, you need to access it from your client application. Panther provides support for accessing COM components via C, JPL, and Java.

For more information about accessing COM components from a client, refer to Chapter 4, “Building Client Screens.”

Deployment Options

You can install your COM components on each application client or use DCOM or MTS to access COM components on remote machines.

For more information about deploying COM components, refer to Chapter 5, “Deploying COM Components.”

COM/MTS Applications

A Panther application using COM or MTS typically consists of:

- Client screens in an application library on the client workstation.
- Panther service components in an application library on the component server machine. (For COM-based deployment, this will be the same machine as the client workstation.)
- Panther DLLs installed on the component server machine.

- DLLs for the COM components registered on the appropriate machine and, for MTS, installed through the MTS Management Console.

The following chapters describe:

- How to build COM components in the Panther editor.
- How to build client screens which access COM components.
- How to deploy Panther COM components using COM, DCOM, or MTS.

CHAPTER

2 Starting Development

Preparing the Development Workstation

In order to develop COM components, the following items need to be available on the Windows workstation:

- Panther editor for building client screens and service components.
- Panther templates for the DLL and `.inf` files (`PrlServer.dll` and `PrlServer.inf`), typically installed in:

  `PantherInstallDir``\config`
- An application directory, where initially you create two empty application libraries, `client.lib` for client screens and `server.lib` for the service components. During development, this directory will also hold the COM components' DLLs.

If you also want to test the components on the same machine, you will need:

- Panther' runtime DLLs in the workstation's `PATH`, typically installed in:

  `ProlificsInstallDir``\bin`

Preparing for Application Development

Before starting development of your COM application, you may wish to complete the following:

- Map the business logic of your application.

  COM components place the business logic of your application in objects to be accessed at runtime. You can map out the business needs and logic for your application manually or through a modeling tool.

- Build your database schema and a repository based on your database objects.

  Having a working database schema speeds the development process. The resulting repository based on that schema contains the necessary information about database table and column identification. Your client screens and service components built from that repository contain the necessary objects for database access.

  For more information on using a repository during application development, refer to Chapter 11, “Creating and Using a Repository,” in *Application Development Guide*.

- Plan how components will be used in your application.

  You can use components:

  - To implement all of your database access and business logic.
  - To implement repeated tasks in a portion of your application.

  One way to organize components in your application is to have one group of components that correspond to database tables and another group of components that implement the business logic. A component in the business logic group could access several database components as it completes a task.

Accessing Databases

In a COM/MTS application, the database can be accessed from the development client, the application client and/or the COM component.

Development clients

Development clients running the editor need to install a Panther database driver in order to make direct connections to the database. Direct connections are required for creating a repository and are also helpful for testing purposes.

Application clients

Client machines needing a direct database connection must install a Panther database driver for the desired database. In other words, a Panther database driver is not required if the clients use COM components to perform all the data fetches and updates.

COM Components

A machine operating as your COM component server must install the ODBC driver from your database vendor in order to access databases. Since the Panther ODBC database driver is built into the Panther DLLs, you will *not* need to install a Panther database driver on these machines.

In this version, Panther-built COM components can only access databases through ODBC.

CHAPTER

3 Building COM Components

A COM component built in Panther consists of a DLL file and a Panther service component having the same name. COM components are developed in the Panther editor, just as client screens and reports are, by:

- Creating the service component in the editor.
- Defining the component's interface: its identifier, properties, and methods.
- Writing the JPL, C, or Java processing that implements the component's methods.
- Specifying properties for the service component.
- Saving the service component, which saves the service component in a Panther application library and creates the DLL, type library, and registry files which allow you to deploy the service component in a COM, DCOM or MTS environment.

Creating Service Components

To create a service component, choose File→New→Service Component. A blank service component appears in the editor.

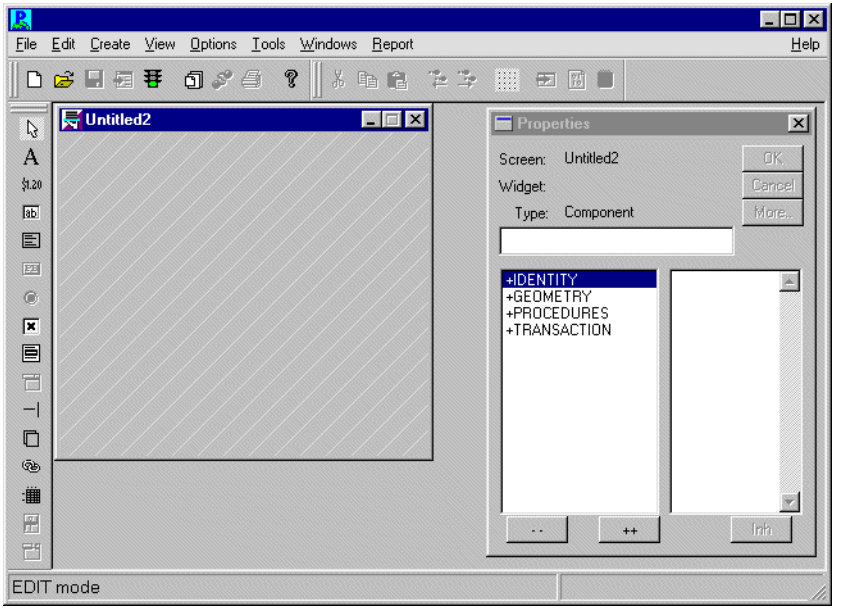


Figure 3-1 A service component is differentiated in appearance and in the number of editor properties in addition to their functionality.

For COM components, you must:

- Define the component's interface, its properties and methods.
- Implement the method's processing.
- Save the component in an application library, which in turn creates the files needed to register and use the component on a workstation.

Defining the Component's Interface

COM components must have an interface defined. This interface consists of a list of the properties and methods supported by the component, together with information about those properties and methods, such as data types, parameter lists, and so on. This interface definition is the public face of your component, and interactions with any application client will use the properties and methods defined in this interface.

To define the component's interface, choose View→Component Interface.

Defining Methods

The Methods section of the Component Interface window displays the methods currently defined for the component. Each row in the grid corresponds to a method. The first column shows the method's return type, the second the method's name, and the third the method's parameters, prefixed by their kind and type.

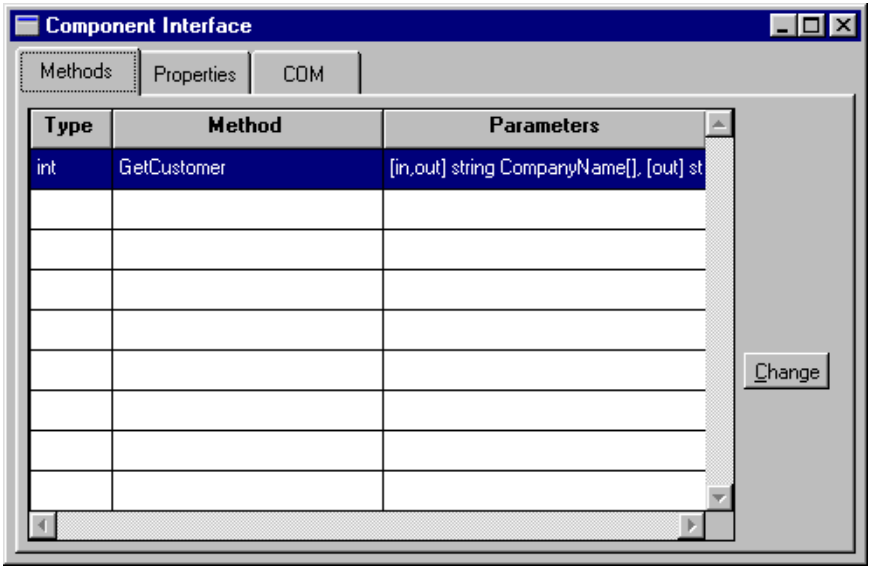
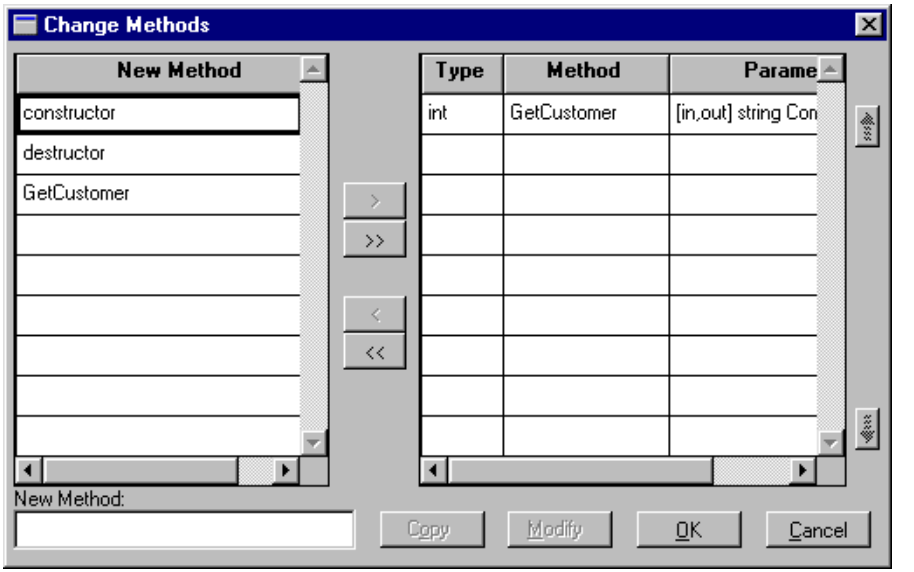


Figure 3-2 The Methods window allows you to view and define the component's methods.

Methods correspond to the actions to be performed by the component. The code to implement the service component's methods can be written in JPL, in C, or in Java. The procedure or function must be in scope at runtime, and the name must match the method name. For more information, refer to page 3-13, "Implementing Methods."

How to Add a New Method

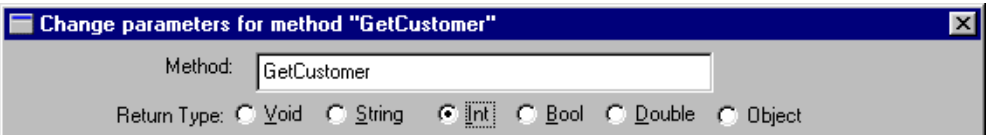
Choosing Change displays the Change Methods dialog where you can add a new method, copy an existing method, or edit a method. The New Methods column displays the JPL procedure names defined in the screen-level JPL Procedures property of the service component.



- In the New Methods column, select the method names and use the arrow buttons to add methods to the grid.
- Alternatively, enter the method name in the New Method field. (This name does not need to be listed in the New Methods column.)
- To add or edit the method's parameters, highlight the method in the grid (on the right). Choose Modify (or double-click) to open the Change Parameters dialog.

Specifying the Return Type

When you add a new method, or modify an existing method, you can edit the method name and specify the return type at the top of the Change Parameters dialog:

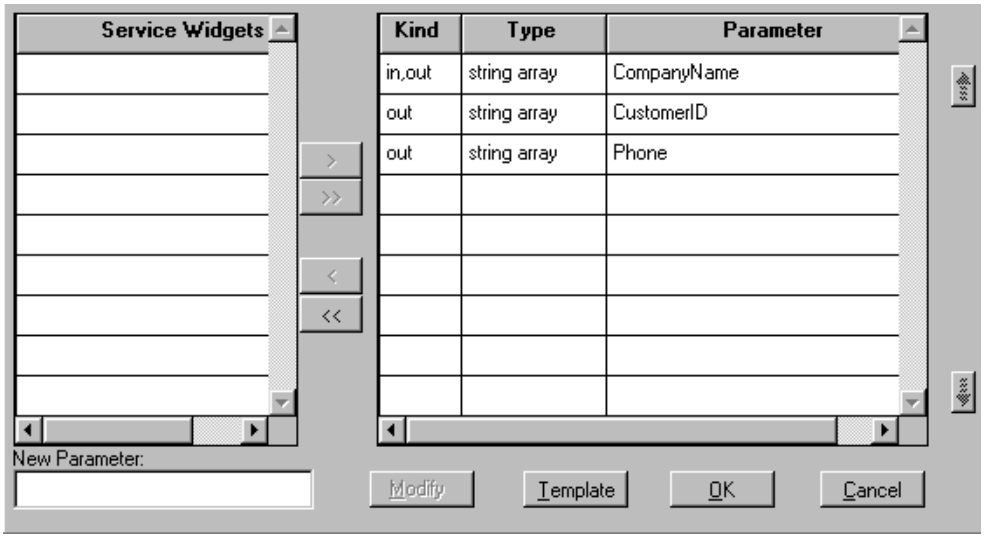


The return type for the method can be Void, String, Int, Bool, Double, or Object. The JPL return value for the method will be converted to the appropriate type and returned to the client.

The method's return cannot be an array. If this functionality is needed, the data should be returned in a parameter.

Specifying the Parameters

On the Change Parameters dialog, the Service Widgets column displays the names of the fields on the service component. Select the widget names and use the arrow buttons to add parameters to the grid. If the widgets are arrays, the array specification is selected for the parameter.

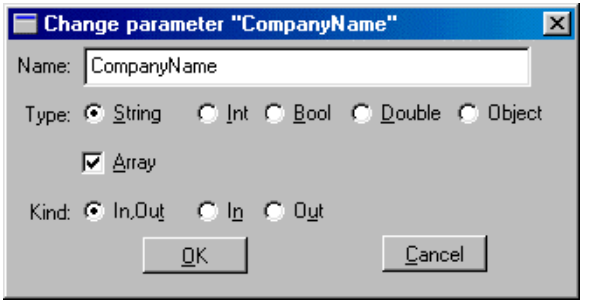


To the right, the grid displays the list of parameters for the method. Each row in the grid corresponds to a parameter. The order of the parameters is important; it is the order clients will use when calling the component.

Choosing Modify (or double clicking on a parameter name) opens the Change Parameter window, which contains four fields:

- Name—The parameter's name.
- Type—String, Int, Bool, Double, or Object.
- An Array check box.
- Kind—In, Out, or In/Out.

Choose the desired options for each parameter.



Implementing the Method

On the method's Change Parameters dialog, choosing Template generates a JPL procedure for the method and its parameters in the Windows clipboard. This generated JPL template can then be pasted into the screen-level JPL procedures or a file.

In the COM samples, the Template option generated the following JPL for the `GetCustomer` method:

```
proc GetCustomer
{
    receive_args (CompanyName)
    return_args (CompanyName, CustomerID, Phone)
}
```

Defining Properties

The Properties section in the Component Interface window displays a grid that shows the properties supported by the component. Often, properties are defined to contain information about the application state.

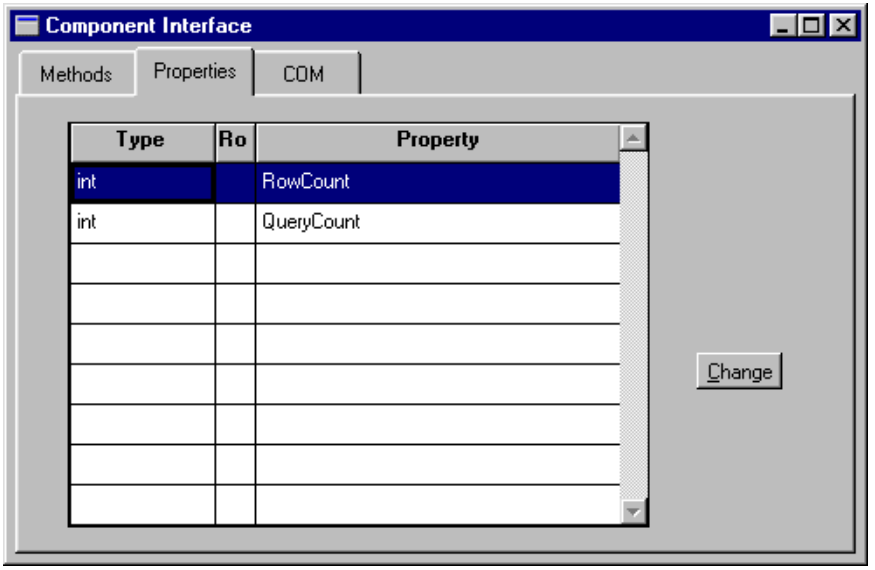
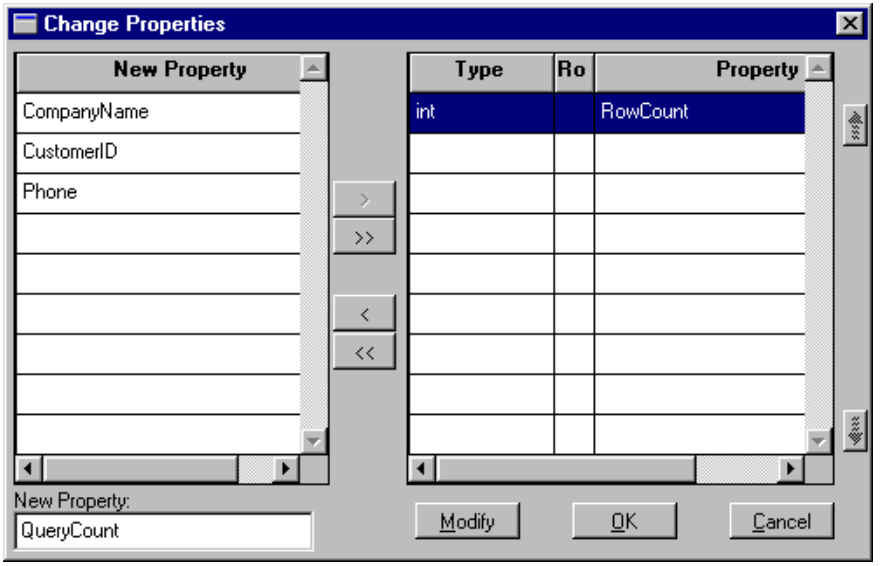


Figure 3-3 The Properties card allows you to add, modify, and delete a component's properties.

Each row in the grid corresponds to a property. The grid's columns correspond to the attributes of each property: Type, Name and Read-only. The order in which the properties appear in this grid is irrelevant to the functionality of your component.

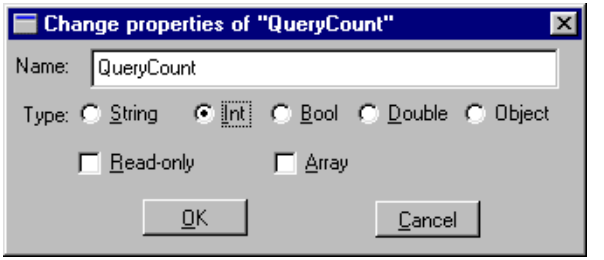
How to Add a New Property

Choosing Change opens the Change Properties dialog where you can add new properties or modify current properties. The New Property column displays the widget names found in the service component. You do not have to choose one of the names on this list. You can enter the name of a global JPL variable or the name of a field that you will create later.



- In the New Property column, select the widget names and use the arrow buttons to add properties to the grid.
- Alternatively, you can enter the name of the property in the New Property field.

Highlight the property name in the grid and choose Modify to display the Change Properties dialog.



- Name—The name clients will use when accessing the property. The name must correspond to a field or global JPL variable accessible when the component is open at runtime. This field or variable implements the property.
- Type—A property can be specified as String, Int, Bool, Double, or Object. The value in the field or JPL variable that implements the property will be converted

to this type when a client requests the value of the property, and handed to the client in the form specified.

- Read-only—If a property is marked Read-only, clients can get its value but not set it.
- Array—A property can have an array of values.

Defining the COM Settings

A COM component written with Panther consists of a DLL file and a Panther component screen that have the same name. In the COM section, you can specify:

- The directory where the component's DLL is stored.
- The version number of the component.
- That a new `CLSID` is needed.
- The template used to create the component's DLL.

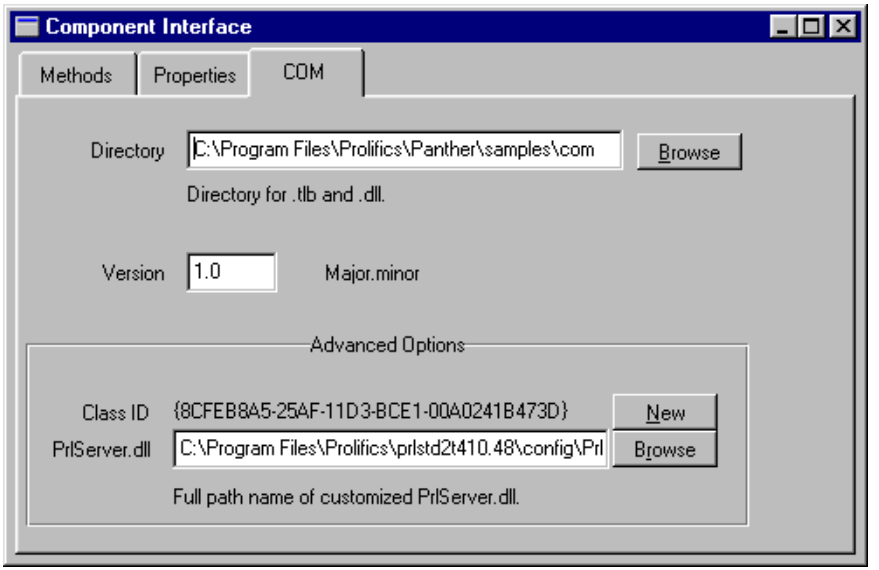


Figure 3-4 The COM card contains the CLSID and DLL information for the component.

Application Directory

The application directory will contain the COM component's DLL, its type library, and client registry initialization file. Since the application needs to locate the server's application library, place `server.lib` in this directory.

For more information on files in the application directory, refer to page 5-2, "Create an Application Directory."

Version Number

You can provide a version number for the component. At runtime, a client application can ask for a specific version of a component; asking for the component without a version number gets the latest version.

When you append new methods or properties to the component, you can update the version number (without generating a new CLSID). You can then register the new version of the component on the server. The old entry is still in the registry for clients calling for the old version, but since all versions have the same CLSID, all versions of the component will use the new DLL.

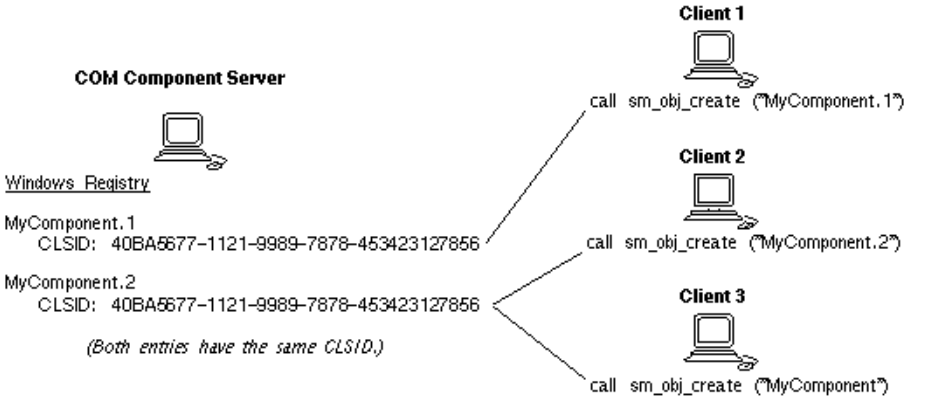


Figure 3-5 By having both entries in the registry, application clients calling for version 1 of the component can still access it, even though there are new methods or properties.

CLSID

The COM card also displays the `CLSID` (Class ID) to be generated for the component.

Press the New button to generate a new `CLSID` if:

- You modify the interface of a distributed COM component. An interface is a contract between the component and the other software components that access it. If you change the interface by modifying existing methods (for example, changing the method's return type or the type or number of parameters), those changes would break existing applications. For this reason, generate a new Class ID.
- You use one component as a template for another, and wish to save the component in the editor under a new name. Note that merely changing the name is not enough to define a new component. The Class ID is the unique identifier that COM uses to distinguish one component from another.

DLL Template

When a service component is saved in the editor, a template DLL, by default named `PrlServer.dll`, is used to create a new DLL whose name matches the name of the Panther service component. Its installed location is:

`PantherInstallDir``\config\PrlServer.dll`

If you need to customize `PrlServer.dll` in order to add your own C code, the distribution contains the source file for `PrlServer.dll`. In this file, you can add custom code by adding a `func_data` structure and calling `sm_install` (`sm_PrlSvcInstall` in the latest version) in order to install your own C functions.

The source code for `PrlServer.dll` is located at:

`PantherInstallDir``\comlink\PrlServer.c`

If you customize the DLL template, give it a new name to distinguish it from the distributed `PrlServer.dll`.

Other files in the `comlink` directory are:

- `makefile` —Makefile for running `nmake`
- `PrlServer.dsp`—Visual C++ 6 Project file
- `PrlServer.dsw`—Visual C++ 6 Workspace file

Implementing Methods

A method supported by a component is a piece of work that a client can request the component to perform. The component implements each of its methods by means of a function (written in C, JPL or Java) that has the same name as the method's name.

Implementing Methods in JPL

To implement a method in JPL, the JPL procedure must be in scope when the service component is open at runtime. The simplest way to do this is to use the Template option on the Change Methods dialog to write a JPL procedure to the clipboard and then paste that procedure in the screen-level JPL (under Procedures→JPL Procedures). However, the procedure could also be written in a separate JPL module and made available with `include` or `public` commands.

A JPL procedure that implements a component's method gets no parameters passed directly to it. To gain access to the method's `in` and `in/out` parameters, as sent by the client making the method request, use the `receive_args` command. For example, a sample JPL implementation of a component's method would be:

```
proc my_method()
{
    vars id, name
    receive_args (id, name, address, city, state, phone)
    . . .
}
```

The JPL command `receive_args` is followed by a list of targets. The values of the `in` and `in/out` parameters from the method call are placed in these target locations. The `out` parameters are skipped, therefore if a given method call has two `out` parameters, the first and third, only the values of the second, fourth and following parameters would be copied into the target list for the `receive_args` command.

The items in the target list must have a valid JPL syntax. In the example above, the first two items on the target list are variables declared locally in the procedure. For the target list to be valid, the latter four items on the target list must correspond to fields

on the screen, to JPL global variables, or to JPL variables declared in the screen's unnamed JPL. Since any valid JPL syntax can be used, you could copy an incoming parameter into a property by using the property API.

The client that made the request for a method is expecting values passed back to it in the `in`/`out` and `out` parameters. It also expects a return value for the method as a whole, and it might also be checking for error codes, to determine whether some error has occurred while attempting to perform the method. To send these values back to the clients code like the following would be used:

```
proc my_method()
{
    . . .
    return_args (id, name)
    raise_exception -2
    return 0
}
```

For information on calling methods from a client screen, refer to page 4-3, "Accessing the Component's Methods."

The `return_args` command works similarly to the `receive_args` command. The values of the JPL variables in the list will be written to the `in`/`out` and `out` parameters of the method, based on the order of the parameters in the method's definition.

The `raise_exception` command is used to send an error code back to the client. The client's error handler can then decide what to do based on the value sent. Microsoft has defined a set of exception codes for use in COM programming.

The JPL `return` command is used to provide the return value for the method.

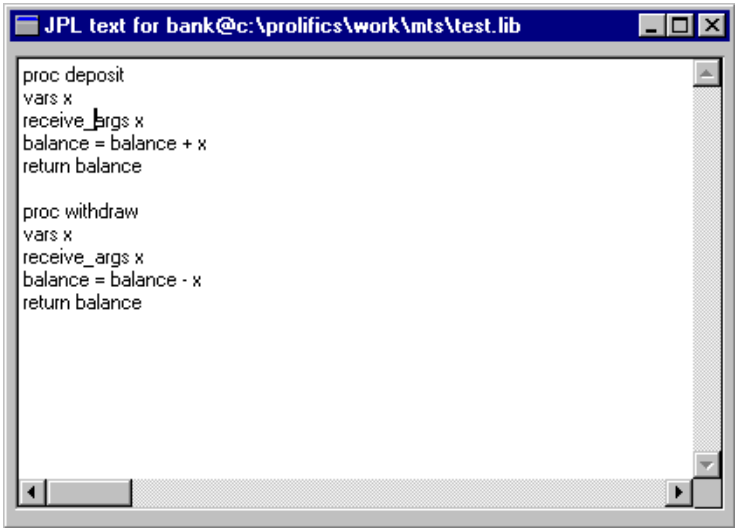


Figure 3-6 This sample screen-level JPL module implements the component's methods.

JPL Variables

JPL variables declared in the component's unnamed procedure are only available in the component-level JPL; they will not be in scope for widget-level JPL or for calls from the client. To increase the scope of the variables, use the JPL `global` command.

Implementing Methods in C

In C, rather than the JPL commands `receive_args`, `return_args`, and `raise_exception`, you would use the following functions:

- `sm_receive_args (char *)`
- `sm_return_args (char *)`
- `sm_raise_exception (int, char *)`

As in JPL, the functions `sm_receive_args` and `sm_return_args` take a list of Panther variables as a single string. The variables can be comma or space-delimited and can include the use of the property API syntax to refer to properties of fields on the component.

To make your C code accessible at runtime, your code must be prototyped and installed in Panther by customizing the DLL template, `PrlServer.dll`. For more information, refer to page 3-12, "DLL Template."

Implementing Methods in Java

The `ComFunctionsInterface` contains methods for `log`, `receive_args`, `return_args`, and `raise_exception`. This interface also contains methods for a series of C functions that implement some of the MTS methods for properties, security checking, and database transactions.

Saving as a COM Component

A COM component developed by Panther consists of the following pieces: a Panther service component, a DLL, a type library, and an `.inf` file.

The Panther service component and the DLL have the same name, such as `my_component.dll` and the service component `my_component`. The DLL lives in a particular directory; the library containing the corresponding service component must also be in the same directory. During development, the DLL is stored in the directory specified on the COM card of the Component Interface window.

To name a service component, choose File→Save As→Library Member, enter a name, and choose the library to save it in, generally `server.lib`.

When you save a service component, you are asked whether to generate a DLL for it. When you generate the DLL, you also generate the type library and a `.inf` file for the COM component. If you have changed the interface definition for the component since

the last time the component was saved, you should choose Yes. Otherwise, you can choose No. Choosing Yes if the component's interface hasn't been changed will have no ill effects; the new type library and `.inf` file will be the same as the old ones.

Once the DLL and the service component are complete, you are ready to deploy your COM component. For more information, refer to Chapter 5, “Deploying COM Components.”

You can view the contents of a type library using Microsoft tools, such as OleView.

Logging Server Messages

You can send server messages to a log file. Because of performance considerations, this is not suggested for application deployment, only for application testing.

How to Activate Message Logging

Place a file named `server.log` in the application directory, along with `server.lib` and the COM component's DLLs. Messages are automatically written to this file when a component is created and destroyed and when an error message is generated. You can write additional messages to this file using the JPL `log` command or its C equivalent `sm_log`.

The following sample log file illustrates some server messages:

```
Mon Jun 21 22:06:30 1999: Component cCustomers
   Created.
Mon Jun 21 22:06:44 1999: Component cCustomers, Method GetCustomer
   Searching on S
Mon Jun 21 22:06:50 1999: Component cCustomers
   Destroyed.
Mon Jun 21 22:07:03 1999: Component cCustomers
   Created.
Mon Jun 21 22:08:07 1999: Component cCustomers
   Destroyed.
```

Calling Other COM Components

A COM component can also instantiate other COM components and, in a Panther application, open Panther screens. Each component operates in a different context; each component also has its own form stack. This means that the property API cannot be used to pass information between COM components; instead, you must use the C functions that call methods and get/set properties.

Programming Component Events

For Panther-built COM components, the Procedures category on the Properties windows contains properties for two events:

- Entry Function—Occurs when the component is instantiated.
- Exit Function—Occurs when the component is destroyed.

During entry processing, it cannot yet be determined whether a component is deployed under MTS or under COM or DCOM. On entry, the `in_server` application property returns `PV_SERVER_COM` for all COM components. After the MTS object context has been defined, the `in_server` property returns `PV_SERVER_COM` for COM/DCOM and `PV_SERVER_MTS` for MTS.

If there is initialization processing that is dependent on the component running under MTS, you must create an initialization method and call it after the component has been created.

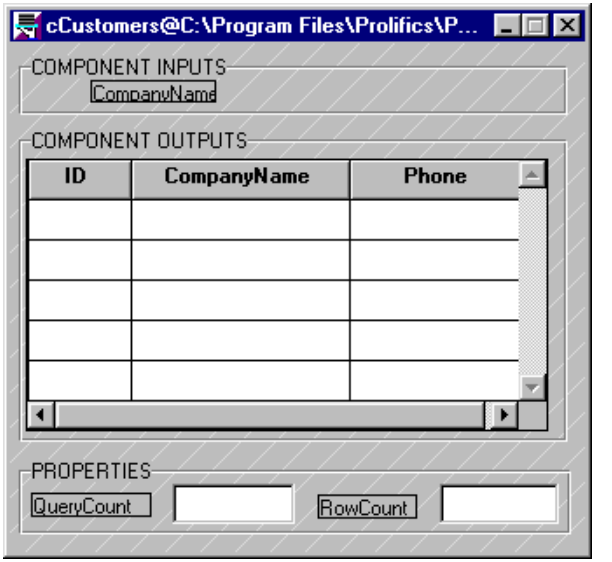
Programming for MTS Deployment

COM components deployed under MTS can call C functions to access MTS features, such as:

- Database transactions (refer to page 5-8, “Using Database Transactions”)
- Security checking (refer to page 5-7, “Assign Roles for Component Access”)

Sample COM Components

The Panther COM Gallery contains sample COM components. One of those components, `cCustomers`, fetches a list of restaurant names according to the user's search parameters. The service component looks like this in the Panther editor:



The Component Inputs box contains the names of the input parameters. In this component, the name is listed for reference only; it serves no functional purpose since it is an `in/out` parameter. The Component Outputs box contains the grid with the widgets designated as `in/out` and `out` parameters. The Properties box contains fields to hold the property values.

The screen-level JPL for this component consists of three procedures:

- `constructor`—This procedure is run on component instantiation through the Entry Function property. It connects to an existing ODBC data source on the workstation.

```
proc constructor
{
DBMS with engine odbc DECLARE c1 connection for \
    DATASOURCE 'Restaurants'
}
```

- `destructor`—This procedure is run when the component is destroyed and is set through the Exit Function property. It closes the database connection.

```
proc destructor
{
DBMS CLOSE_ALL_CONNECTIONS
}
```

- `GetCustomer`—This procedure implements the `GetCustomer` method by receiving the input parameters from the client screen, selecting data using the transaction manager, setting the variable for the `RowCount` property to the number of fetched rows, and passing the parameters back to the client.

```
proc GetCustomer
{
receive_args (CompanyName)
//log "Searching on :CompanyName"
call sm_tm_command ("VIEW")
RowCount = @dmrowcount
return_args (CompanyName, CustomerID, Phone)
}
```

The `log` command has been commented out, but was used during testing to send messages to `server.log`.

Since it is the position of the parameters that determines how the client will receive the data, the parameter list in the `return_args` command must match the client's invocation of the method.

For information about viewing the Panther COM Gallery, refer to Appendix B, “COM Samples.”

CHAPTER

4 Building Client Screens

In Panther Windows applications, any Panther client screen can access COM components. A series of C functions, also callable in JPL, are available for interacting with all Panther service components, in either a COM or Enterprise JavaBean Panther application. In addition, there are some C functions written specifically for COM or MTS deployment.

You can also embed ActiveX controls in a client screen. However, ActiveX controls are the only COM components visible in a client screen. All other COM components must first be instantiated before setting their properties or calling their methods. For information about using ActiveX controls in client screens, refer to Chapter 19, "ActiveX Controls," in *Using the Editors*.

Working with COM Components

Instantiating COM Components

You must first specify which type of service components are currently in use with the `current_component_system` property: `PV_SERVER_COM` for COM components.

To access a COM component, you must instantiate the component by calling `sm_obj_create`. It takes a string parameter, the name of the component, for example `cCustomers or cCustomers.1`.

You can also instantiate a COM component using its Class ID (`CLSID`) by calling `sm_c_com_obj_create`; this function takes a string parameter, the Class ID of the component. This is the unique identifier that is generated when COM components are first created. The curly braces in the Class ID specification are optional. A sample Class ID is something like:

```
{59CCB4A0-727D-11CF-AD36-00AA00A47DD2}
```

These functions return the object ID for the specified component or `PR_E_OBJECT` if they fail. Using the object ID, you can access the component's properties and methods.

Note that these functions are not needed when working with ActiveX controls. Because such controls are embedded in Panther screens, the properties of the ActiveX container specify the CLSID and name of the ActiveX control allowing Panther to instantiate the control.

In our sample screens, the unnamed JPL procedure declares a variable to contain the component's object ID and the screen entry function instantiates the component. For example:

```
vars id
proc entry
@app()->current_component_system=PV_SERVER_COM
id = sm_obj_create("cStrings")
return
```

If you use `sm_obj_create` to instantiate your components, you need to ensure that you do not have two components with the same name but with different CLSIDs on the same machine.

Destroying COM Components

After invoking and working with the methods or properties of a component, you should destroy it by calling `sm_obj_delete_id`. This function takes one parameter: the object ID for the component you wish to destroy. Otherwise, the component will continue to exist until the application terminates (or goes from test to edit mode).

If a component is running under MTS, its life cycle can be managed for you by MTS, depending on whether the component is marked as belonging to a transaction and whether the work in the transaction is complete.

In our sample screens, the components are programmatically destroyed during screen exit:

```
proc exit
call sm_obj_delete_id(id)
return
```

Accessing the Component's Methods

In order to access a COM component's methods, you need to know the component's parameters and call the function `sm_obj_call`. The syntax in JPL is as follows:

```
ret = sm_obj_call (objid, methodName, parm1, parm2, ....)
```

The function's first parameter is the object ID of the component whose method you wish to use. The second parameter is the name of the method you are calling. The rest of the parameters are a comma-separated list of the parameters to the method itself.

Specifying the method's parameters

COM components can take three kinds of parameters: in parameters, out parameters and in/out parameters. Parameters can be passed as literal strings or using the property API syntax. For out and `in/out` parameters, Panther assigns the returning values to the variables, fields or properties originally specified.

For example, you have a component called `cEmployee` that supports a method called `newEmployee`. `newEmployee` takes three parameters in the following order:

- An `out` parameter `EmpId`, which places its return value in a field on the client screen.
- An `in/out` parameter `EmpName`, which derives its input value from a field on the client screen.
- An `in` parameter `StartDate`, which derives its input value from a field on the client screen.

You can invoke `NewEmployee` method with the following JPL:

```
vars id, ret
@app()->current_component_system=PV_SERVER_COM
id = sm_obj_create ("cEmployee")
ret = sm_obj_call (id, "NewEmployee", \
    EmpId, EmpName, StartDate)
```

In addition to the out parameters, this method call returns a value. ret contains the return value for the method.

A method cannot return an array. In such cases, that information needs to be passed as a parameter.

Determining the parameter's data type

At runtime, Panther uses the information in the type library to determine the data type for each parameter. Even if the type library is missing, Panther can generally determine the correct data type.

However, if the parameter must be passed using a dispatch interface (and therefore the object ID must be specified) and the type library is either missing or does not indicate a dispatch interface is needed, you will get a “type mismatch” error. In this case, generate the object ID and specify the parameter using `@obj`. For example, the following JPL from the Treeview Control in the COM samples uses this syntax:

```
vars imagelist   // imagelist control
vars images      // list of images in the imagelist
vars pic         // one picture


imagelist = sm_obj_create ("COMCTL.ImageListCtrl")
images = sm_obj_get_property(imagelist, "ListImages")


pic = sm_com_load_picture ("logo.bmp")
call sm_obj_call (images, "Add", 1, '', @obj(pic))
sm_obj_delete_id (pic)
```

Calling Microsoft’s COM Components

Popular Microsoft applications, such as Microsoft Excel and Microsoft Word, are implemented as COM components or Active Document Servers and can be called from your Panther application. One of the Panther COM Samples illustrates calls to Microsoft Excel which get and set data in a spreadsheet.

The following procedures instantiate Microsoft Excel and write data to a spreadsheet:

```
vars wsid, cid
proc screen_enter
{
wsid = sm_obj_create("Excel.Sheet")
cid = sm_obj_get_property(wsid, "Application.ActiveSheet")
}

proc setcell
{
call sm_obj_set_property \
(cid, 'Range(":(col):(row)").Value', CellValue)
}
```

The following procedures create a Microsoft Word document. For more information, refer to Microsoft documentation on how to use OLE Automation with Microsoft Word.

```
vars word
// Run "Word" and get the Application object
word = sm_obj_create ("Word.Application")
// Add a new document (untitled)
call sm_obj_call (word, "Documents.Add")
// Add some sample text
call sm_obj_call \
    (word, "Selection.TypeText", "This is some text")

// Save the document with the specified name
// If you do not give a full path, the file will be
// put into Word's notion of current directory
// Note that colons must be doubled in JPL

call sm_obj_call \
    (word, "ActiveDocument.SaveAs", "C::\Test.doc")
// Quit Word

call sm_obj_call (word, "Quit")
// and destroy the COM object
call sm_obj_delete_id (word)
```

Accessing the Component's Properties

Properties in COM components generally contain the application state information. You can use the `sm_obj_set_property` and `sm_obj_get_property` functions to access properties. The following example sets a property on the component associated with the id variable:

```
ret = sm_obj_set_property(id, "PropName", "PropSetting")
```

ActiveX controls have additional functionality. Their properties are listed in the Properties window and can be accessed at runtime using the property API syntax. For more information on property access for ActiveX controls, refer to "Setting Properties at Runtime" on page 19-6 in *Using the Editors*.

Even though you cannot pass an array as a parameter to these functions, indexed properties are supported. The following command would get the 6th element of the property's array:

```
ret = sm_obj_get_property(id, "Name[6]")
```

Designating an Error Handler

You can define an error handler for COM method invocations, for example:

```
call sm_obj_onerror ("ErrorHandlerName")
```

The string passed to `sm_obj_onerror` is the name of a function that you want to designate as the error handler. If a COM operation (method call, property access, or object invocation) generates a negative exception code, the error handler function will be called. The specified function is passed three parameters: the error number in decimal format, the error number in hexadecimal format, and a description of the error.

COM methods can also return exception codes using the JPL verb `raise_exception` or its C equivalent `sm_raise_exception`. You can get the value returned by calling `sm_com_result`. The function `sm_com_result_msg` looks up the exception code in a Windows system table and returns the message to which it corresponds.

Designating an Event Handler

You can designate event handlers for COM components using `sm_com_set_handler`. Events are mainly programmed for ActiveX controls in response to user-initiated events, such as mouse clicks. For example:

```
call sm_com_set_handler (id, "Event", "EventHandlerName")
```

For more information, refer to "Specifying an ActiveX Event Handler" on page 19-11 in *Using the Editors*.

Sample Client Screens

Although simple in appearance, this screen contains the fields and push buttons needed to operate the client part of an application. Nothing in the client interface indicates that it is calling a COM component. However, the JPL Procedures property contains the JPL processing to create and destroy the COM component, call its methods, and get its property values.

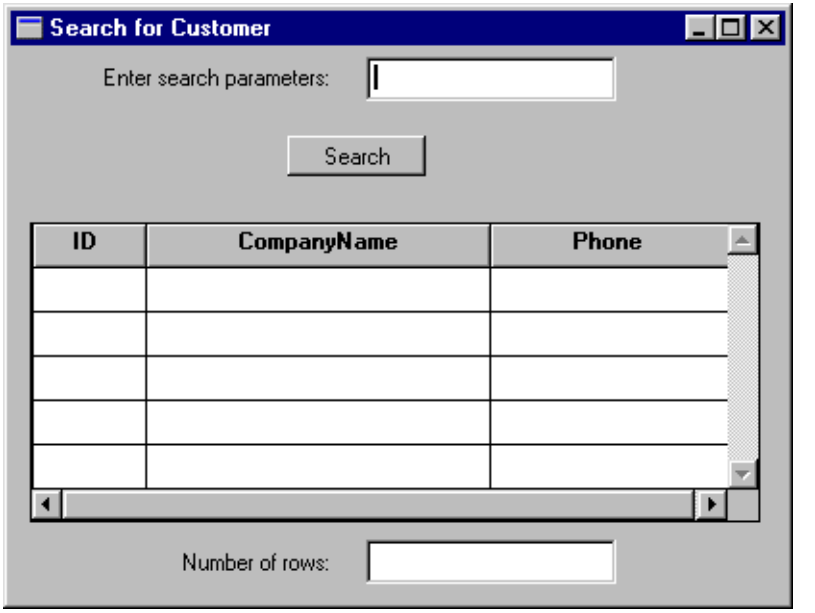


Figure 4-1 An application's client screen that searches for customer names.

The unnamed JPL procedure creates the variable which will hold the object ID of the COM component. During screen entry, the COM component is instantiated.

```
vars id
proc enter
{
@app()->current_component_system=PV_SERVER_COM
```

```
id = sm_obj_create("cCustomers")
}
```

On exiting the screen, the COM component is destroyed.

```
proc exit
{
call sm_obj_delete_id(id)
}
```

This JPL procedure calls the COM component's `GetCustomer` method and gets the value of the `RowCount` property:

```
proc do_search
{
vars error
CompanyName[1] = search
error = sm_obj_call (id, "GetCustomer", \
    CompanyName, CustomerID, Phone)
rowcount = sm_obj_get_property (id, "RowCount")
}
```

The Panther COM Gallery includes this sample in addition to other COM components and client screens. To view these samples, open *`PantherInstallDir`*`\samples\com\comsamples.lib` in the Panther editor.

Writing a Java Event Handler

To implement the processing for the sample screen in Java, a screen event handler instantiates and destroys the COM component. The client screen calling this event handler has a Java Tag of `ClientScreen`.

```
import com.prolifics.jni.*;

public class ClientScreen extends ScreenHandlerAdapter{

   public void screenEntry(ScreenInterface s, int context){
      FieldInterface id = s.getField("id");
      FieldInterface id1=s.getField("id1");
      CFunctionsInterface cFuncs = s.getCFunctions();
      ApplicationInterface appface=s.getApplication();
      ScreenInterface tscr=appface.getScreen();

   int a=appface.set_int
```

```
      (Constants.PR_CURRENT_COMPONENT_SYSTEM,
       Constants.PV_SERVER_COM);
       id1.itofield(a);

       id.itofield(cFuncs.sm_obj_create("cStrings"));
   }

   public void screenExit(ScreenInterface s, int context){
      CFunctionsInterface cFuncs = s.getCFunctions();
      FieldInterface id = s.getField("id");
      cFuncs.sm_obj_delete_id(id.intval());
```

A button event handler for the Search push button calls the method and gets the number of returned rows. The push button calling this event handler has a Java Tag of `SearchButtonHandler`.

```
import com.prolifics.jni.*;

public class SearchButtonHandler extends ButtonHandlerAdapter{

    public int buttonValidate
           (FieldInterface f, int item, int context){
       ScreenInterface s = f.getScreen();
       FieldInterface idField = s.getField("id");
       FieldInterface rowField = s.getField("RowCount");
       FieldInterface searchField = s.getField("search");
       FieldInterface
          companyNameField = s.getField("CompanyName");
       CFunctionsInterface cFuncs = f.getCFunctions();
       companyNameField.putfield(1,searchField.getfield());
       int id = idField.intval();
       String i = cFuncs.sm_obj_call("(" + id + ",
          'GetCustomer',CompanyName,CustomerID,Phone)");
       String st = cFuncs.sm_obj_get_property
          ( id, "RowCount");
       rowField.putfield(st);
       return id;
   }
}
```

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

CHAPTER

5 Deploying COM Components

A COM component built in the Panther editor consists of a DLL file, a Panther service component, and possibly a type library, all having the same name. Once these files are available, the COM component can be deployed on:

- Windows workstation running COM or DCOM.
- Windows 2003 servers or later running COM, DCOM, or MTS.

Steps for Deployment

In order to make a COM component available in a Panther application:

- The COM component's DLL must reside on the server machine. If the DLL does not contain the type library definition, the type library must also be present.
- The Panther application library containing the service component must be available.
- The Panther DLLs must be in the system `PATH`.
- The COM component must be registered.

Create an Application Directory

Since the DLL for the COM component and the Panther application library containing the service component are typically located in the same directory, create an application directory and copy the following files to this directory:

- The application library containing the Panther service components. The default library is named `server.lib`.
- The COM components' DLLs.

Changing the Library Name

When the COM component is instantiated, Panther looks for a service component matching that name in an application library named `server.lib`. That library must be located in the same directory as the component's DLL.

To specify a different library name or location, create a `smvars.bin` file containing a setting for `SMFLIBS` with the library names. That `smvars.bin` file must reside in the same directory as the COM component's DLL.

Note: The `smvars.bin` file for COM components can contain two settings: `SMFLIBS` and `SMPATH`. Other settings are ignored.

Accessing Libraries

In order for an application to be able to access all its components (COM components, screens, menus), you must use `SMFLIBS` to open libraries. Threading specifications dictate that if an object opens a library, only that object will be able to use it.

Install the Panther DLLs

Three DLLs distributed with Panther must be on the system's `PATH` in order for Panther-built components to work:

- `PrlSmCom.dll`
- `PrlDmCom.dll`
- `PrlTmCom.dll`

The default location for these DLLs is *PantherInstallDir*\bin. The installation program places this directory in the system's PATH.

These DLLs support calls from individual components according to the COM apartment-threading model.

Register the COM Component

For COM Deployment

In COM applications, the client application and the COM component reside on the same machine; the COM component must be registered on that machine.

To register the component, use regsvr32.exe, located in your Panther util directory. For example, typing the following command in a DOS prompt window registers forecast.dll:

```
regsvr32 forecast
```

This process will include the path to the DLL in the Windows system registry. When the component is instantiated at runtime, COM will look for the component's DLL in this location.

For DCOM Deployment

In DCOM applications, the Panther application and the COM component reside on different machines. Each machine must register the component, but the process for the server machine (where the COM component is located) differs from the client machine (which runs the application and accesses the component).

For the server machine, register the component using regsvr32.exe (located in your Panther util directory).

```
regsvr32 forecast
```

For each client machine, register the component using the *componentName*.inf file. Before making the component's .inf file available to your application clients, check its settings (particularly the machine name), and edit those settings as needed. This is an ASCII file and can be edited with any text editor.

Once the .inf file is ready:

- Use `regcli32.bat` to install the component on the client machine.

  ```
  regcli32 forecast
  ```

- If the drive containing the component and the `.inf` file is mounted, run the file's install command (available by right-clicking on the file).

For more information about the settings in the `.inf` file, view the default file, `PrlServer.inf`, which is in ASCII format and located in the Panther `config` directory.

Changing the settings in `PrlServer.inf` affects all COM components generated from Panther. This file must be named `PrlServer`.inf.

Configuring DCOM

In order to use DCOM, it must be installed on the client and server machine and configured using `dcomcfg`. In particular, the check box for Enable Distributed Communication on This Machine must be selected. For more information, check Microsoft's documentation on configuring machines for DCOM.

For MTS Deployment

For MTS, the Microsoft Management Console contains options for registering components and creating export files for application clients. The following section details this process.

Registering Components for MTS

To deploy Panther-built COM components under MTS, you must complete the steps in the previous section, populating the application directory and installing the Panther DLLs, before registering the component.

Create a Component Package

Components must be installed into a component package. A package is a collection of components that run in the same process. You can create an new package or add components to an existing package.

To view the existing packages, start the Microsoft Management Console and choose the desired server machine. One of the settings for that machine is Packages Installed. One method to use to create a new package is to highlight Packages Installed and choose Action→New→Package.

Refer to Microsoft's *MTS Documentation* for more information about creating packages.

Install the Component on the Server

To install a component under MTS on a server, use the Microsoft Management Console. This saves the path to the DLL in the Windows system registry.

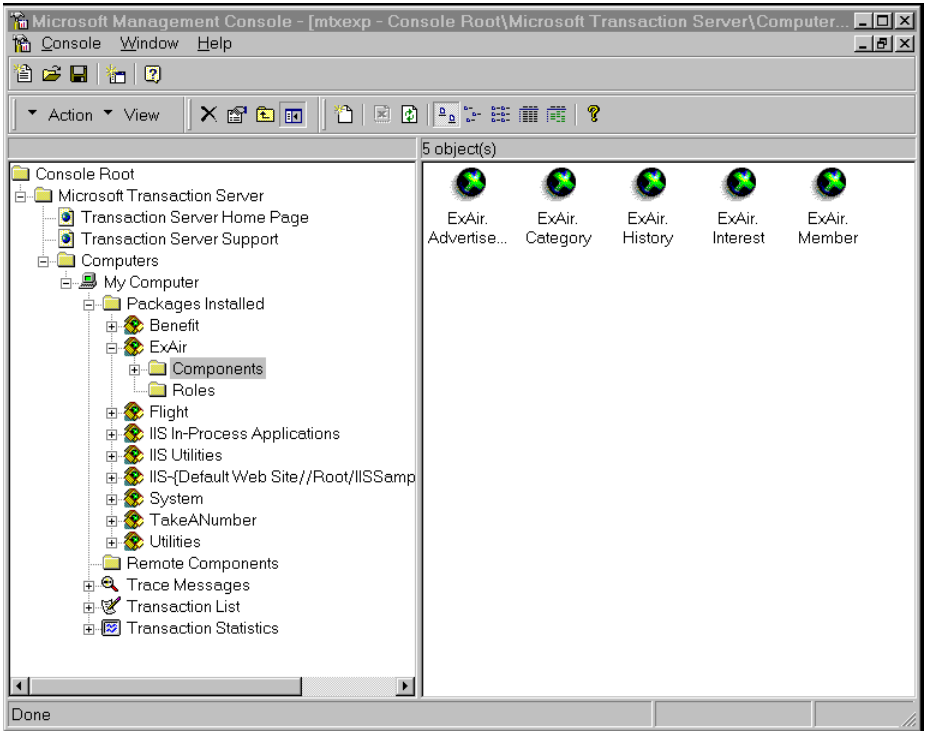


Figure 5-1 The Microsoft Management Console window for this sample MTS application shows five COM components in the right panel.

How to Install the Component under MTS

1. Highlight the Components folder under the desired package.
2. Choose Action→New→Component.

   The MTS Management Console will lead you through the steps to install a new component or a pre-existing component.
3. Choose Install New Component(s) if the components are not in the system registry. You will then select the component's DLL.

4. Choose Import Component(s) that Are Already Registered if the component is in the system registry. You will then be able to choose it by name from the list of COM components.

Note: If you import a component that is already registered, you will be unable to view its interfaces through the MTS Management Console. For more information, refer to Microsoft's *MTS Documentation*.

How to Unregister a Component

- To unregister the component so that it can be installed as a new component, run:

  ```
  regsvr32 /u componentName
  ```

Assign Roles for Component Access

If you want to implement security features on your component packages, MTS offers two types of security checks: programmatic security, where you call interfaces from within the application, and declarative security, where you assign users, or groups of users, to roles.

A role is the name assigned to a group of users that will access a component package. For example, a human resources application could define roles for Manager and Employee. In the Microsoft Management Console, you assign users to the defined roles.

Panther MTS applications can implement programmatic security using the following functions which call methods of the IObjectContext interface:

- `sm_mts_IsCallerInRole`—Determines if a caller is assigned to a role, where caller is the identity of the process calling into the server.
- `sm_mts_IsSecurityEnabled`—Determines if security checking is enabled.

The following section of a JPL procedure, taken from a service component's processing for the `UpdateData` method, checks to see if the caller is in the `writers` role before proceeding:

```
proc UpdateData
vars security, role
```

```
role = sm_mts_IsCallerInRole("writers")
...
```

For more information about implementing security processing, refer to Microsoft's *MTS Documentation*.

Export the Component to Clients

If the COM component is to be accessed from other machines, it must be registered on those machines as well. The MTS Management Console provides facilities for exporting a component package which creates an executable for installing the package on a remote machine.

Highlight the component package and choose Action→Export.

Refer to Microsoft's *MTS Documentation* for information about exporting a package.

Using Database Transactions

MTS includes support for database transactions at the component level. For each component installed under MTS, the component's Transaction properties specify whether the component:

- Requires an existing transaction.
- Requires a new transaction.
- Supports transactions.
- Does not support transactions.

Panther MTS applications can control database transactions by calling the following functions:

- `sm_mts_SetComplete`—Tells MTS that the work is complete.
- `sm_mts_SetAbort`—Tells MTS to abort the transaction.

- `sm_mts_DisableCommit`—Tells MTS not to commit the transaction at this time.
- `sm_mts_EnableCommit`—Tells MTS that the work is complete and ready to be committed.
- `sm_mts_IsInTransaction`—Determines if a transaction is currently active.

Refer to Microsoft's *MTS Documentation* for additional information about MTS database transactions.

APPENDIX

A COM/MTS Utilities

This chapter describes command-line utilities that can help you develop and manage a Panther COM/MTS application. Utility descriptions are organized into the following sections, as applicable:

- Utility name and brief description.
- Syntax line and argument descriptions.
- Description of the utility.

To get a command-line description of a utility's available arguments and command options, type the utility's name with the `-h` switch. For example:

```
makedlls -h
```

makedlls

Generate the service component's DLL and associated files

```
makedlls [-pv] library [library...]
```

`-p`
: Put the output in the application directory specified by the service component. If not specified, the output is placed in the current directory. The directory setting can be viewed in the editor on the COM section of the Component Interface window.

`-v`
: Display information in verbose mode. Generates a list of service components being processed.

`library`
: Name of library.

Description

The `makedlls` utility generates a DLL (`.dll`), a type library (`.tlb`), and a client registry file (`.inf`) for each service component in a Panther application library.

The location of `PrlServer.dll`, the Panther template DLL, is also specified on the COM section of the Component Interface window. If `makedlls` cannot find `PrlServer.dll` at that location, the utility generates an error:

```
Cannot create service DLL from DLL_path
```

If the path for `PrlServer.dll` is unspecified, `makedlls` looks for the template at `$SMBASE\config\PrlServer.dll`.

APPENDIX

B COM Samples

The Panther distribution includes some sample COM components and client screens to illustrate the procedures described in this manual. To run the samples, open the samples library in the editor. By using the editor, you can easily view the component's interface and the JPL coding used to implement the processing.

How to Install the COM Samples

- Open a DOS prompt.
- Change to the directory:

  PantherInstallDir\samples\com
- Run:

```
regsvr32 cStrings
regsvr32 cCustomers
```

How to Configure Database Access for the COM Samples

- On the Control Panel, select ODBC.
- Select System DSN.
- Add a System DSN for a Microsoft Access database.
  - Choose Add.
  - Select the Microsoft Access ODBC Driver.
  - Choose Finish.
  - Enter the Data Source Name as `Restaurants`.
  - Choose Select.

- Select `Restaurants.mdb` (located at *`PantherInstallDir`*`\samples\com\Restaurants.mdb`).

How to View the COM Samples

- On the Start Menu, choose Panther→Samples→COM Samples.

Description of the COM Samples

Using an ActiveX Control
Use Microsoft's treeview control to view the samples.

Calling Methods on a Client
See how a client screen calls a method on a COM component.

Implementing Methods on a Component
For the method called by the client, see how that method was implemented on the COM component.

Accessing a Database - Client
Enter a letter of the alphabet to see the restaurant names beginning with that letter. Pressing Search calls a method that fetches the data.

Accessing a Database - Component
The COM component connects to a Microsoft Access database and uses the transaction manager to fetch data.

Calling a COM Application
Enter data in the text field and choose Send Data to transfer the data to an Excel spreadsheet. Enter a cell location and choose Get Data to retrieve data from the spreadsheet. To enter data in Microsoft Excel, you must start Excel before opening the screen.

Note: If you enter data in the Excel spreadsheet, you must move the cursor to the next cell to complete the entry. Otherwise, you get an error: "Call was rejected by callee."

Calling a Panther Application

Call a Panther-built COM component from within Microsoft Excel. When you open `com.xls` with the macros enabled, you can go to Sheet 1 and call the `cCustomers` component.

APPENDIX

C New Windows Executables

As part of the Panther COM/MTS installation package, a set of standard Panther executables is provided for the client and the web application broker. Depending on your configuration and on the platform and database being used, you might need to create new executables. Specifically, a new executable is required if you are using a database for which DLLs are not provided or if you are adding C functions to your Panther application.

Alternatively, Panther distributes workspace and project files for use with Microsoft Developer Studio. Instructions on how to use the workspace are provided in the Release Notes.

Client and Web Application Broker Executables

To create new executables, Panther distributes a single makefile that can be invoked via the command line utility, `nmake`.

How to Create a New Panther Executable

Complete the installation process of Panther components.

1. Ensure that the appropriate application variables (`SMBASE`, etc.) have been applied to your environment.
2. Go to your application directory (or create one) and copy all files from the Panther `link` subdirectory to it.
3. Edit the `makefile` in your application directory, commenting or uncommenting lines as needed to build the appropriate executables. For further information, refer to page C-2, "Specifying the Executables."
4. Uncomment the appropriate database in the `makefile` and edit the database-specific `makevars.dbs` (where `dbs` is the extension of the database) file to choose the correct version of your database software. For further information, refer to page C-3, "Linking in the Database."
5. Type `nmake` at the command line to build the executable. By default, the `makefile` in the current directory is used for the `make` (and `make`) command.
6. The `nmake` process creates a new Panther executable, `prodev32.exe`. Give the executable file a unique name to distinguish it from the distributed executable or others that you have built differently.
7. If you built a new web application broker executable (`jserver`), modify the `SERVER` variable in your application's initialization file to reference the new `jserver` executable in its own directory. Do not overwrite the `jserver` executable in the `util` directory.

Specifying the Executables

To indicate the executables to build, edit the makefile in your application directory:

1. Comment or uncomment the appropriate client executables as needed (these are uncommented by default):

`PRORUN = prorun32.exe`	Runtime executable
`PRODEV = prodev32.exe`	Development executable
`RWRUN = rwrun32.exe`	Report utility

If Web application broker software has been installed, comment or uncomment the server executable as needed:

`JSERVER = jserver.exe`	Web application broker executable (uncommented by default)

2. To override the value of certain application variables such as `SMBASE`, uncomment the appropriate lines in the `PARAMETERS` section.
3. If you have the Panther web application broker on the same machine as other Panther software, it is recommended that they be installed in the same directory. However, if they are not in the same directory, set `WEBBASE` (in `WEB PARAMETERS` section) to the web application broker installation directory.
4. The Panther debugger allows you to trace JPL and Panther screen events and is installed by default for clients. If you do not wish to use the debugger, comment the lines in the `DEBUGGER PARAMETERS` section.
5. You have the option of building a standalone executable. Comment or uncomment the appropriate lines in the `MIDDLEWARE PARAMETERS` section.
6. If you want to add Microsoft Codeview debugging information, uncomment the corresponding block.

Linking in the Database

If you are using a database for which DLLs are not provided, you must edit the makefile in your application directory to link in the appropriate database.To include the appropriate database in your executables:

1. Uncomment the appropriate `include` statement in the `SELECT DATABASE SOFTWARE` section of the `makefile`.
2. Edit the corresponding `makevars.`*dbs* (where *dbs* is the extension of the database) file to choose the correct version of your database software.
3. In the `makevars.`*dbs* file, verify or update the following:
   - Set the flag *dbs*`_INIT` to one of the following: `d`, `l`, `u`, `p`. This flag controls the handling for case sensitivity. The default is `d`. To find out what the

default is for your database engine, refer to the online database-specific driver notes.

- In the *databaseName* PARAMETERS section of makevars, verify your database engine's version. Uncomment the appropriate block of parameters based upon this version. Also, verify and correct the pathnames if necessary.
- Set the flag *dbs*_ENGNAME to specify the default engine name.

Changing the Panther COM Template DLL

PrlServer.dll is the template for Panther COM components and can be edited to enable you to link your own C functions into your COM component. Files included in the comlink directory are:

PrlServer.c	Source code
PrlServer.rc	Resource file for copyright information
PrlServer.dsw	Microsoft Developer Studio Workspace File
PrlServer.dsp	Microsoft Developer Studio Project File
makefile	For command line makes

Edit the source code in PrlServer.c to include your C Functions, make a new DLL, and in the Panther editor, update the name and/or location of the template DLL on the COM tab of the Component Service interface. It is recommended that you give the new DLL a different name (other than PrlServer.dll).

APPENDIX

D Deployment Checklist

The following checklist summarizes the deployment steps for COM components.

COM, DCOM, MTS	
❑	Application directory: The COM component's DLL.
❑	Application directory: The Panther application library containing the service components, such as server.lib.
❑	Application directory: If the application library name is not `server.lib`, `smvars.bin` containing a setting for `SMFLIBS`.
❑	The Panther DLLs must be in the system's PATH, typically `$SMBASE\bin`.
❑	The COM component must be registered as needed for COM, DCOM, or MTS.
COM	
❑	Register the COM component using `regsvr32.exe`.
DCOM Server	
❑	Register the COM component using `regsvr32.exe`.
DCOM Clients	
❑	Register the COM component using `regcli32.bat` or the Install option of the component's `.inf file`.
MTS Server	

❑	Register the COM component using the Microsoft Management Console.
MTS Clients	
❑	For each component package, run the MTS-generated installation file on the application clients.

APPENDIX

E Troubleshooting Guide

Errors listed in this chapter can be encountered when developing or deploying a COM/MTS application.

Errors on the server machine can be logged during development testing. For more information on log files, refer to page 3-17, “Logging Server Messages.”

Error Message Listing

ERROR: at line xxx in *procedureName* **(proc** *procedureName***) 'ret=sm_obj_call‘ -->'id...**

Cause	JPL programming error. In this instance, JPL procedures were included to instantiate the component, but the Screen Entry and Screen Exit properties to call those procedures had not been set.
Action	Set the Screen Entry and Screen Exit properties to the required names.

ERROR: Call was rejected by callee.	
Cause	Your entry in the other COM component is incomplete
Action	If you get this error when you request data from Microsoft Excel, your cursor is still in the cell from which you requested data. You must move the cursor to a new cell on the spreadsheet to complete an entry in that application.

ERROR: COM status returnMember not found.	
Cause	`server.lib` cannot be found.
Action	1. Place server.lib in the same application directory as the DLLs for the Panther COM components. 2. If the library has a different name or a different location, create an smvars.bin file with a setting for `SMFLIBS` to open the desired library.

ERROR: mtx.exe PrlSmCom.dll cannot be found in ...	
Cause	The `PATH` variable has been updated, but MTS references an old setting.
Action	Reboot the machine.

ERROR: Server execution failed.	
Cause	This error appears when there is an error on component entry.
Action	1. Check that the procedure/function specified for entry is available. 2. Check that the Panther DLLs are present and specified as part of the `PATH`.

ERROR: Type mismatch error.	
Cause	The type library does not specify enough information about the data types used in the COM control.
Action	Attempt the call using the `@obj` syntax. For an example of its use, refer to `sm_com_load_picture`.

Index

E

J

M

P

R

S

T

U

W