TABLE OF CONTENTS

Contents:

Panther

Upgrade Guide

Prolifics®

Release 5.51

Document 0404
March 2017

Contents:

PREFACE

About This Document

The *Upgrade Guide* contains information for current users of JAM 7 on what has changed in the product, what is new, and how to migrate existing applications to the new environment.

This guide is composed of several chapters. The first chapter contains information for users who are upgrading from JAM 7. Other chapters include information on upgrading from JAM 5 to JAM 7 and JAM documentation.

If you are a first-time customer, the information in this guide may be useful, but is not necessary. Starting with *Getting Started* and the overview chapter in *Application Development Guide* is a more appropriate reading path.

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.

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

CHAPTER

1 Upgrading to Panther from JAM 7

This chapter lists features that are new and changed since JAM 7 for customers upgrading to Panther.

Additional upgrade information is available in the following chapters:

- For information about a utility to help in the Panther upgrade process, refer to Chapter 2, “Using the JAM Upgrade Utility.”
- For information about upgrading JAM 5 applications, refer to Chapter 4, “Upgrading to Panther from JAM 5.”
- For upgrades from the JetNet middleware adapter to Oracle Tuxedo, refer to Chapter 3, “Upgrading to Oracle Tuxedo from JetNet.”

A major development change is that application files are now stored in libraries. Refer to page 1-10, “Libraries, not Files” for more information about this feature.

Installation

Start-up License

A start-up license is included in all Panther products. This temporary license allows you to start using Panther immediately. The license expires in 45 days, during which time you should have contacted the Prolifics License Desk to receive your permanent license, as arranged by your salesperson. The start-up license does not preclude or eliminate the need to obtain a permanent license.

When a temporary license is in use, a warning message is issued when you start the Panther editor. This warning is issued once a week until the last week, at which point it is issued daily.

Program Startup

By default, Java is initialized on program startup. You can change this initialization using the new behavior variable `JAVA_USE`.

Windows clients having an old version of the Java DLLs display an error message, "Java Not Supported". To update the Java DLLs, run the executable in `%SMBASE%\jvm`.

Editor

Menu Changes

File Menu

- Open and Save recognize local and remote libraries and repositories, not standalone files or screens. A library must be opened, and all objects must be saved in a library. Once opened, objects in the library can be opened individually.

The New, Open and Save options have the following new options:

- Service Component—Creates or opens a service component for use in distributed application processing.
- JPL—Opens a JPL file in a library for editing (in an external editor if set) so that it can be saved back to the library or to disk.
- Java—Opens an external editor for writing or editing a Java file.

The Import Database Objects option is now on the Tools menu.

Edit Menu

- (Web applications) Find→Overlapping Widgets selects every overlapping pair of widgets. All other widgets are deselected. HTML does not support overlapping widgets, and as a result, Web browsers can render widgets in positions that differ from the appearance in the screen editor workspace.
- Includes the following new options, as well as corresponding toolbar buttons, which are available when a JPL library module is open or when the screen- or report-level JPL Procedures, JavaScript or VBScript properties are being written/edited:
  - Insert From Library—Includes a JPL, JavaScript or VBScript file from another open library.

- Read File—Includes a JPL, JavaScript, or VBScript file from disk.
- External Editor—Invokes the editor you specify (via the `SMEDITOR` variable).

Create Menu

- With the Extended Widgets category, the Create menu indicates dual-deployment widgets versus environment-specific widgets. Widgets listed in the Extended Widgets category cannot be deployed on every platform. The widgets currently found in this category are:
  - Graph widgets—available for Motif, Windows and Web (on certain platforms)
  - ActiveX control containers—available for Web and Windows
  - Tab decks and tab cards—available for Motif and Windows
  - Toggle button, combo box and scale widgets—available for Motif and Windows

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

- The Create menu has new widget types:
  - ActiveX control containers (Web and Windows applications)—In your Panther screen, create a container and then specify through the container's properties the ActiveX control to appear in that container. (Refer to Chapter 19, “ActiveX Controls,” in *Using the Editors*.)
  - Tab decks and tab cards (Motif and Windows applications)—The tab control allows widgets to be grouped onto individual display “cards.” The tab control or “deck” contains a series of tab cards; these cards are accessed by means of index tabs, which are analogous to the dividers in a notebook or the labels on a group of file folders. (Refer to Chapter 17, “Tab Controls,” in *Using the Editors*.)

View Menu

- Library TOC—A new option on the View menu. The library table of contents provides access to all libraries and their members. You can use it to open, add files to, and extract files from libraries. For more information, refer to “Viewing the Library Table of Contents” on page 2-5 in *Using the Editors*.

- Component Interface—For service components, allows you to define the public interface–the properties and methods—of COM components and Enterprise JavaBeans.
- Report Structure—Displays the report structure view of a report. (Panther, by definition, incorporates report and web options.)

Options Menu

The Options menu is now a subset of the Tools menu and includes the following new options:

- Check Overlap on Screen Save—(Web applications) When active, any save operation (Save, Save As, Save All) forces the editor to check for overlapping widgets before actually saving the screen or screens affected by the save request. HTML does not support overlapping widgets, and as a result, Web browsers can render widgets in positions that differ from the appearance in the editor workspace.
- Direct to External Editor—When active, invokes your preferred text editor (specified via the `SMEDITOR` variable) when you open the JPL, JavaScript, or VBScript window; or edit properties that allow input of multiple lines of text, such as the Control Strings property and the Initial Text property (when it is associated with a widget having an array size greater than one).
- Configure Toolbars—For Windows executables, allows you to specify which toolbars to display in the editor.
- Editor Tabs—Works in conjunction with the Direct to External Editor menu options to allow you to specify the number of spaces that defines a TAB character.
- Service Alias—For developing JetNet and Oracle Tuxedo applications, specify a user identifier to use when testing services. For more information, refer to “Using Service Aliases to Test Services” on page 5-8 in *JetNet/Oracle Tuxedo Guide*.
- Reload Java Classes—When active, reloads the Java classes when entering test mode or exiting the editor.

Tools Menu

This new menu bar item contains the Import Database Objects that was located on the File menu in previous releases, and gives editor access to the styles editor, menu bar editor, and JIF editor (in JetNet and Oracle Tuxedo). There are also the following new options in Panther:

- Generate TM SQL—For the current screen, writes the SQL statements that the transaction manager generates for the screen to a file. These SQL statements could then be used to construct stored procedures or invoke `DBMS QUERY` or `DBMS RUN` directly.
- Generate Component—For COM components, generate a type lib file after changing the component interface without having to save the Panther service component. For Enterprise JavaBeans, generate and/or compile the bean's Java files.
- Compile Java—Compiles the specified Java class.
- IBM VisualAge for Java—(WebSphere only) Starts IBM's Visual Age for Java program.
- IBM WebSphere Administrative Console—(WebSphere only) Starts IBM's WebSphere Administrative Console where you install and deploy Enterprise JavaBeans.

Other Editor Changes

Other editor changes are:

Properties

For descriptions of new properties and information about changes in the Properties window, refer to page 1-20, “API Changes.”

Library Member Access

Access to library members is provided by the Select Library Member dialog box. This dialog is available from the Properties window when properties requiring filenames are selected, and wherever else access to libraries is required. Both local and remote libraries can be accessed.

JPL Modules

JPL modules can be created in the editor. Access to the JPL window is provided from the File menu. JPL can be read into the window from a library or from a file on disk. The JPL can be saved to a library or to an external text file on disk. The JPL is automatically compiled when it is saved to a library, eliminating the need to do any file/library manipulation outside of the editor.

Non-modal Text Windows

The JPL Program Text window, JavaScript window, and VBScript window are no longer modal when writing screen- or report-level JPL procedures or JavaScript/ VBScript functions. This allows you to edit multiple files as well as move freely between script files and the editor workspace. The buttons have also changed for these windows.

Date/time Formats for Year 2000 Compliance

Date/time formats have two new Format Type property specifications in the Properties window for assigning date/time formats that display a four-digit year: `MON/DATE/YR4 HR:MIN2` and `MON/DATE/YR4`. These are associated with the `DEFAULT3` and `DEFAULT4` mnemonics, respectively. In addition, `DEFAULT3` is set as the default type in the Properties window.

Alternatively, you can use the behavior variable, `DA_CENTBREAK`, to set the behavior for applications using two-digit years. The default value of `DA_CENTBREAK` is 50. Therefore, if the year setting is equal to or greater than 50, the year is processed as 19xx; if the year is less than 50, the year is processed as 20xx. You can change the setting of `DA_CENTBREAK` by setting its value in the `smvars` file or at runtime by using `sm_option`.

Name Extensions

Screen names no longer have a default value is set for the application variable `SMFEXTENSION`. If you do not explicitly set this variable in a setup file or the environment, Panther no longer adds an extension to names during file searches; and all filenames must be fully qualified; for example, supplied a screen name of `myscreen`, Panther searches only for `myscreen` and not `myscreen.`*`ext`* also.

The recommended file extension for binary screen files is `.scr`. For binary report files, the recommended extension is `.rpt`. The file extension for temporary file names has been changed from `.jam` to `.pro`.

Editor Toolbars

A new menu option, Options→Configure Toolbars, determines which toolbars are available in the editor.

Screen Wizard

When constructing screens in the screen wizard, columns defined as being NOT NULL in the database are automatically selected to be part of the screen and are designated with the number symbol (#).

For JetNet and Oracle Tuxedo, refer to page 1-54, “Screen Wizard” for information about changes to the screen wizard in those executables.

Grids

There are new properties and functions pertaining to grids which control the amount of space between grid rows and which sort the data appearing in grids. For more information, refer to page 1-24, “New Properties for Grids.”

Menu Bar Editor

The menu bar editor is now located on the Tools menu in the editor workspace.

For JetNet and Oracle Tuxedo, refer to page 1-55, “Menu Bar Editor” for information about changes to the menu bar editor in those executables.

Docking Toolbars

For Windows applications, toolbars can dock to the MDI frame or float within the MDI frame. A new menu pixmap property, Hot Pixmap, controls the appearance of the item when a mouse moves over an active toolbar item. In addition, pixmaps can be specified for the Inactive Pixmap property.

In previous releases, an Inactive Pixmap was a grayed version of the Active Pixmap in Windows applications. That capability is still there, but you can also specify a separate inactive pixmap. For each toolbar state that you want to indicate in your application–active, inactive and hot–you must supply a pixmap for each toolbar item. The size of the pixmaps for the entire toolbar is taken from the size of the first pixmap.

At runtime, application properties control the appearance and position of the toolbar; refer to “Dockable Toolbar Properties” on page 15-10 in *Application Development Guide*.

Styles Editor

The styles editor is now located on the Tools menu in the editor workspace.

JIF Editor

For JetNet and Oracle Tuxedo applications, refer to page 1-55, “JIF Editor” for information about the JIF editor, a graphical utility used to create and edit a JIF file that contains information about your application's services.

Development and Deployment

New Executable Names

The development executable name has been changed from `jamdev` to `prodev`. The runtime executable name has been changed from `jam` to `prorun`.

Universal Makefile

A single, universal makefile is provided. In prior releases, different pieces of JAM required their own individual makefiles, and it was sometimes necessary to merge the individual makefiles for your specific needs. With the new makefile, you can build either two-tier or three-tier executables:

- With or without Motif
- With JDB and/or one or more database engines

For more information, refer to the *Installation Guide*.

Libraries, not Files

To facilitate deployment, all application components (screens, JPL modules, bitmaps, and so on) are now required to reside in libraries. In the screen editor, the components are opened as library members by default, or as repository entries by request. The notion of an independent disk file no longer exists. Refer to Chapter 2, “Using the JAM Upgrade Utility,” for information about a utility which groups your disk files into libraries.

Panther cannot write to libraries created by versions of JAM. To upgrade and make writable a preexisting library, use `formlib` with the `-w` flag. In the screen editor, if an attempt is made to access a library that has not been upgraded, a warning instructs you to upgrade the library. Note that Panther can still read preexisting libraries.

During development, libraries can exist anyplace on your network. If your development environment includes shared libraries on a server (remote libraries), your clients can have access to those libraries via a (provided) development server (`devserv`) configured to access remote libraries. With Panther, access is provided to both local and remote libraries from the screen editor, the menu bar editor, the styles editor, and the JIF editor.

For more information on opening libraries, refer to “Opening and Creating a Library” on page 2-7 in *Using the Editors*.

Library Locking

Libraries now automatically reuse space as part of normal processing. Because of this, libraries now require read locks as well as write locks. Two methods of library file locking are used, internal and external, depending on the platform. By default, the internal (native OS) file locking system is used on UNIX and Windows. If an external file locking method is used, lock files must be created in the directory where the library exists; make sure the directory where the libraries reside is writable.

During deployment, it is advantageous to make all your libraries read-only, so that no file locking is required. However, if you decide to make an application library writable, make sure that if external locking is used, lock files can be created in the directory where the library resides.

For more information, refer to Chapter 10, “Accessing Libraries,” in *Application Development Guide*.

Source Control

Source control is now available (SCCS, PVCS) for JPL, menus, and styles–in addition to screens—that are in libraries under source control.

Libraries Names

A Panther application can consist of a set of libraries, depending on the product and architecture.

All Panther applications are installed with a library named client.lib in the `config` directory. When you start Panther, this library will be opened unless different libraries are specified in the environment or initialization file using `SMFLIBS`. It contains:

- `client.lib`—`smwizard.bin`, `smwzmenu`, `styles.sty`, and numerous graphics files.

JetNet and Oracle Tuxedo Applications

A Panther JetNet/Oracle Tuxedo application consists of a set of libraries:

- A client library that contains client screens, JPL files, styles, bitmaps, menus, and any other objects that define the user interface.
- A server library that contains service components, JPL files, styles, etc., for defining the server in three-tier architecture.
- A common library that contains the JIF, configuration files, JPL files, and any application objects used.

`client.lib`, `server.lib`, and `common.lib` are three libraries distributed with Panther. You can use these libraries as a starting point to build your application. The contents of these libraries are:

- `client.lib`—`smwizard.bin`, `smwzmenu`, `styles.sty`, and numerous graphics files
- `server.lib`—`smwizsrv.bin` and styles.sty
- `common.lib`—`jif.bin`

The Library Table of Contents window gives you access to library members, and permits you to add external files, such as bitmaps, into open libraries.

References to Files Outside of Libraries

Although it is recommended and documented that all application files (for example, screens, JPL modules, graphics, and menus) be stored in libraries, this release of Panther will continue to support applications that reference files outside of Panther libraries. To ensure compatibility with future releases, it is recommended that newly developed applications store files in libraries.

JPL Programming

Declaring Variables

Use commas to delimit initial values in `global` and `vars` declarations.

Sending and Receiving Data

The `send` and `receive` commands have changed for word-wrapped fields. Word-wrapped fields are now sent as a single item; the receive command should specify a word-wrapped field which permits it to use `sm_ww_write` to place the text.

Another change is that the number of available bundles can be set with the `max_bundles` application property. It defaults to ten bundles (including the unnamed bundle) if unspecified.

Variable Assignments

In previous versions of the product, an expression which mixed numeric with string variable assignments yielded inconsistent results. It is illegal to mix these assignments within one expression. For example, the following assignment previously yielded either 0 or an empty string, depending on which version is being used:

```
%.0 a='' // Assigned '' to a
```

Now, this assignment generates a syntax error.

Application Properties

The `@jam` property shortcut for the application name has been replaced with `@app()`. `@jam` will continue to work for backward compatibility.

New Commands

COM and WebSphere applications can call the following JPL commands:

- `log`—Writes a message to the server log file.
- `receive_args`—Receives a method's parameters from a client.

- `return_args`—Returns a method's parameters back to the client.
- `raise_exception`—Sends an error code back to the client.

JetNet and Oracle Tuxedo applications also have additional commands (see page 1-59, “JPL Commands.”

Java Interface

In addition to C and JPL, you can program your application behavior in Java. In the editor, Panther objects (screens, service components, widgets) can be assigned a Java tag, which defines a Java class to act as an event handler for that object.

At runtime, when a given object has an event handler associated with it, Panther will invoke the methods supported by the event handler in response to application events.

The event handler classes must provide methods that correspond to the various kinds of events supported by the object with which it is associated. To this end, predefined interfaces, that the event handler classes must implement, have been provided.

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

Internal File Locking Available on Windows

Internal (native) file locking is now the default for Windows, and you will need to run `formlib -e` on a library in order to continue using external lock files. Once a library is set to external file locking with `formlib -e`, you must run `formlib -i` on that library in order to use internal file locking.

Opening Library Files in Windows

In Windows executables, double clicking on the name of any file in a library will open the file in the program associated with it according to the Windows File Type setting.

MSVC Project Files

MSVC project files are now available for rebuilding your Panther executables.

Team Development

In JetNet and Oracle Tuxedo executables, developers can have personal copies of screens in library files and services in the JIF in order to make and test changes during development.

Utilities

File Extension Option

A change has been made to the way the `-e` file extension option is implemented on UNIX in the utilities `cmap2bin`, `jpl2bin`, `key2bin`, `msg2bin`, `msg2hdr`, `var2bin`, and `vid2bin`. Formerly, in UNIX, file extensions were appended to existing extensions. In DOS, using -e replaced an existing file extension. Now, use of the `-e` option works the same on both platforms—any existing file extensions are replaced.

Changed Utilities

Changes were made to the following utilities:

`binherit`

In addition to updating screens with inherited values from the repository, binherit also updates reports.

The following changes were implemented for the -u option:

- The -u option will be ignored for members of libraries that can only be opened read-only.

- If the -u option is not selected, libraries will be opened read-only.

`dd5upg`

(JAM 5 updates only) The `dd5to6` utility has been renamed as `dd5upg`.

`f2asc`

In addition to converting screens between binary and ASCII format, `f2asc` also converts reports and service components. ASCII files for screens and service components start with a S: output area containing screen/component properties; ASCII files for reports start with a R: output area. The output area for static labels has changed from S: to L:. The output area for the service component's interface starts with I:.

`f5upg`

(JAM 5 updates only) In addition to being renamed `f5upg` (previously `f5to6`), a new `-p` option includes the GUI interface values for the `hmargin`, `vmargin`, `hbuffer`, `vbuffer` properties in the converted screens.

`formlib`

Panther cannot write to libraries created by any version of JAM. To upgrade and make writable a preexisting library, use `formlib` with the `-w` flag. In the editor, if an attempt is made to access a library that has not been upgraded, a warning instructs you to upgrade the library. Note that Panther can still read preexisting libraries.

The `-m` flag now compacts the library by removing unused space. Using this option before making the library read-only will allow the read-only operation to be reversible. (This option is only available for Panther libraries.)

The -o option makes a library read-only. Be aware that once this is done, the library cannot be made writable again unless the library is first compacted with the `-m` flag.

The `-s` flag now synchronizes a library with the source code management directory. This used to be accomplished with the `-m` flag, along with library compacting.

`monitor`

In Panther, you must start the web application by using monitor or by installing the application as an Windows service which uses Services properties in the Control Panel to start the application.

monitor has new options: -restart which combines the clean, stop, and start options and -install for installing the application as an service.

`r2asc`

The `r2asc` utility has been superseded by `f2asc`; therefore, to convert a report between binary and ASCII output, use `f2asc`.

`rinherit`

The `rinherit` utility has been superseded by `binherit`; therefore, to batch update all reports with inherited values from your application's repository, use `binherit`.

`rw6toprl`

`rw6to7` has been renamed as `rw6toprl`.

New Utilities

Jam to Panther

A utility to help you upgrade your JAM application to Panther by packaging the application files into libraries. (Refer to Chapter 2, “Using the JAM Upgrade Utility.”)

AxView

In the Windows development environment, COM components, including ActiveX controls, are displayed by AxView, the ActiveX and COM Control Viewer. With this utility, you can view a component's methods, properties, CLSID number, and installation location.

Web Setup Manager

A Web-based utility is available for creating and updating the files needed on your Web application server: the requester executable and the Web initialization file.

For step-by-step instructions, refer to Appendix B, “Web Setup Manager,” in *Web Development Guide*.

COM/MTS Utilities

`MakeDLLs`

In COM/MTS applications, a utility to generate the service components's DLLs for the specified libraries.

JetNet/Oracle Tuxedo Utilities

Refer to page 1-57, “Administration Utilities” for information on utilities to administer your JetNet or Oracle Tuxedo application.

WebSphere Utilities

`makeejb`
: In Panther/WebSphere applications, a utility to generate the Java files for the service components in the specified libraries.

Configuration

Name extensions
: Screen names no longer have a default value set for the application variable `SMFEXTENSION` which, in previous releases, specified the default file extension for screens.

Video files
: Video files are no longer needed on GUI platforms. The `SMVIDEO` environment variable is only required for character-mode Panther users.

Opening multiple libraries
: A single declaration of `SMFLIBS` can now point to multiple libraries. Separate directories with a vertical bar (|) or use the convention used by your operating system for listing multiple directories in path. (For UNIX, this is a colon, and for Windows, it is a semi-colon.) For the JetNet/Oracle Tuxedo middleware adapter, the default setting automatically opens client.lib, server.lib, and common.lib.

Java
: The behavior variable `JAVA_USE` specifies whether Java is initialized. This variable can also be used to control the opening Java support message.If you are using Java event handlers, four optional environment variables are available. `SMJAVAEDITOR` specifies a different text editor than `SMEDITOR`.

`SMJAVALIBRARY` specifies the location of Java libraries, if the default location needs to be changed. `SMJAVAFACTORY` specifies the location of the class factory if the default class factory is not used. `SMJAVACOMPILE` specifies the command used to compile Java using Tools⟶Compile Java.

Motif resources

A new Motif resource has been added to the Prolifics resource file, `Prolifics*positionIsFrame`. When placing a window at a specific position on the display, the requested position can be for the placement of the frame or for the placement of the client window inside the frame. If the position is for the frame, set this resource to true (the default setting).The window manager can have a resource of the same name. The value of the Prolifics resource should match the value of the window manager. The distributed resource files are in the `config` directory.

JetNet/Oracle Tuxedo Variables

The following variables were developed specifically for the JetNet/Oracle Tuxedo middleware adapters:

Connecting to the middleware

For the JetNet/Oracle Tuxedo executables, variables are used to connect to the middleware: `SMRBCONFIG`, `SMRBHOST`, and `SMRBPORT`. For more information on middleware connections, refer to Chapter 9, “Connecting to the Middleware,” in *Application Development Guide*.

WebSphere Variables

The following variables were developed specifically for the Panther/WebSphere environment:

IBM WebSphere Administrative Console

For Panther/WebSphere applications, you can specify the command to launch IBM's WebSphere Administrative Console program with `SMWSADMIN`.

Oracle Tuxedo support in Panther/WebSphere

In Panther/WebSphere applications, the initialization file (Panther.ini) can set `SMTPCLIENT` to specify whether Oracle Tuxedo connectivity is enabled and what type of client is needed (native or workstation). Set `SMTPINIT` to specify the default arguments to the `client_init` command.

WebSphere Application Server
: In Panther/WebSphere applications, specify URL of the machine running WebSphere Application Server in `SMPROVIDERURL`.

API Changes

The Panther API has been significantly expanded to accommodate the middleware API and to incorporate other product changes.

Specifying Application Properties

The `@jam` property shortcut for the application name has been replaced with `@app()`. `@jam` will continue to work for backward compatibility.

Additional Flags for Widget Functions

Two additional flags are now documented for widget functions:

`K_EXTEND`
: The widget is an extended selection list box.

`K_EXTEND_LAST`
: For extended selection list boxes, the widget is the last item in the list box.

Properties Window

Database property category
: The properties listed under Database have been reorganized under new subheadings for text widgets:

- Fetch Data includes Select-related properties (for example Use In Select and Use In Where).

- New Data includes Insert-related properties (such as Use In Insert).
- Change Data includes Update-related properties (such as Use In Update).
- Remove Data includes the In Delete Where property.

Database, Transaction and Server View property categories

The properties listed under Database and Transaction for table views have been reorganized. In addition, a new category, Server View, contains all the properties having to do with select processing.

Service property category in JetNet and Oracle Tuxedo executables

Table view and link widgets have a Service property category. With the JetNet and Oracle Tuxedo middleware adapters, service names can be specified to implement database access operations. Table views can have the Insert, Update, Delete, and Select service properties set. Link widgets can have the Validation Service property set. If you create your client screens with the screen wizard, these properties are set automatically.

For more information, refer to “Creating Services with the Screen Wizard” on page 5-5 in *JetNet/Oracle Tuxedo Guide*.

Component API Changes

New Library Functions for Components

As of Panther 4.2, you can use the same programming interface for both COM components and Enterprise JavaBeans in a Panther application:

`sm_log`

Write a message to a server log.

`sm_obj_call`

Call a service component's method.

`sm_obj_create`

Instantiate a service component. Before calling this function, the application must specify the type of components currently in use with `current_component_system`.

`sm_obj_delete_id`

Remove a service component.

`sm_obj_get_property`
: Get the value of a service component's property.

`sm_obj_onerror`
: Install an error handler for a service component.

`sm_obj_set_property`
: Set the value of a service component's property.

`sm_raise_exception`
: Send an error code back to the client.

`sm_receive_args`
: Receive the method's parameters from the client.

`sm_return_args`
: Return a list of parameters back to the client.

New Properties for Components

There are new properties associated with service components:

Current Component System (`current_component_system`)
: A runtime-only application property that specifies the type of component system currently in use: `PV_SERVER_COM` for COM components or `PV_SERVER_EJB` for Enterprise JavaBeans deployed under WebSphere Application Server.

In Server (`in_server`)
: An application property which specifies which server is in use for a service component: `PV_SERVER_COM`, `PV_SERVER_MTS` or `PV_SERVER_EJB`.

Provider URL (`provider_url`)
: For WebSphere applications, a runtime-only application property specifying the location of the WebSphere application server machine. If `SMPROVIDERURL` is set in the environment, the property is initially set to this value.

ActiveX Controls and COM Components

Web and Windows applications can create and deploy ActiveX controls in client screens. Windows 32-bit applications can create COM components and deploy them under COM, DCOM, and MTS.

New Library Functions for COM Components

There are new library functions associated only with use of ActiveX controls and other COM objects. Additional functions are documented on page 1-21 in “Component API Changes.”

`sm_com_load_picture`
: Get the object ID for the specified picture.

`sm_com_QueryInterface`
: Access the QueryInterface method for the specified COM component.

`sm_com_result`
: Get the error code returned by the last call to a COM component.

`sm_com_result_msg`
: Get the error message returned by the last call to a COM component.

`sm_com_set_handler`
: Set an event handler for the specified event on a COM component.

New MTS Functions

For COM components running under MTS, there is a set of wrapper functions to the associated MTS methods.

```
sm_mts_CreateInstance
sm_mts_CreateProperty
sm_mts_CreatePropertyGroup
sm_mts_DisableCommit
sm_mts_EnableCommit

sm_mts_GetPropertyValue
sm_mts_IsCallerInRole
sm_mts_IsInTransaction
sm_mts_IsSecurityEnabled
sm_mts_PutPropertyValue
sm_mts_SetAbort
sm_mts_SetComplete
```

New Properties for COM Components

In addition to the properties for ActiveX controls listed here, refer to page 1-22, “New Properties for Components.”

ActiveX Controls

For ActiveX Controls, there is a new property category with three Panther properties: Control Name (`control_name`), `CLSID` (`clsid`), and Runtime License (`runtime_license`). ActiveX controls are available in Web and Windows applications.

- To select an ActiveX control registered on your workstation:

  Under Active X, select the Control Name from the option menu. The `CLSID` property is automatically filled.

- To use an ActiveX control unavailable on your workstation:

  Under Active X, enter the `CLSID` for the ActiveX control.

An ActiveX control can have its own set of properties which you can access at runtime using the syntax `ax_property_name`, which prevents naming conflicts between Panther properties and ActiveX control properties.

For more information about ActiveX controls, refer to Chapter 19, "ActiveX Controls," in *Using the Editors*.

Grid API Changes

New Library Functions for Grids

There are new library functions associated with sorting data in arrays and grids:

`sm_obj_sort`

Sort the object's occurrences according to the rules specified in the object's Sort Order property.

`sm_obj_sort_auto`

Sort the object's occurrences according to the conventions for grids in Windows.

New Properties for Grids

There are new properties associated with the use of grids:

Column Click Action (`column_click_action`)

For widgets in grids, under Format/Display, specify the action–sort or custom function–that occurs when a user clicks on the grid column header.

Column Click Function (`column_click_func`)
: For widgets in grids, under Format/Display, specify the custom function to invoke when a user clicks on the grid column header. For this property to be available, Column Click Action must be set to Custom.

Default Row Margin (`default_row_margin`)
: Use this application property to control the grid row height if the Row Margin property is not set for the grid widget.

Row Margin (`row_margin`)
: For grid widgets, under Geometry, adjust the space between the text and row dividers to control the row height.

Sort Order (`sort_order`)
: Under Format/Display, specify the sort order to be used when the widget is in an array or in a grid. If the widget is in a grid, the Column Click Action property must also be set to Sort.

Sort Order Function (`sort_order_func`)
: Under Format/Display, specify the custom function to be invoked when Sort Order is set to Custom. The function can be either a JPL procedure or prototyped C function.

Tab Control API Changes

New Properties for Tab Controls

There are new properties associated with use of the tab controls in Windows applications:

Card (`card`)
: For widgets on tab cards, a runtime, read-only property returning the object id of the tab card of which the widget is a member.

Card Entry Function (`card_entry_func`)
: For tab cards, under Focus, the name of the function to be called when the tab card is entered.

Card Exit Function (`card_exit_func`)
: For tab cards, under Focus, the name of the function to be called when the tab card is exited.

Card Expose Function (`expose_function`)
: For tab cards, under Focus, the name of the function to be called when the tab card is made the topmost card in the deck.

Card Hide Function (`hide_function`)
: For tab cards, under Focus, the name of the function to be called when the tab card ceases to be the topmost card in the deck.

Card Number (`card_number`)
: For tab cards, under Identity, specify the number location of the card in the deck.

Conceal Tabs (`conceal_tabs`)
: For tab decks, under Identity, determines whether the index tabs for the cards in the deck are visible.

Deck (`deck`)
: For tab cards, a runtime-only, read-only property returning the object id of the tab deck of which the tab card is a member.

Number of Cards (`number_of_cards`)
: For a tab deck, a runtime-only, read-only property specifying the number of cards in a tab deck, including hidden cards.

Tab Entry Function (`tab_entry_func`)
: For the index tab field on tab cards, under Focus, the name of the function to be called when the tab card is topmost and its index tab gains focus.

Tab Exit Function (`tab_exit_func`)
: For the index tab field on tab cards, under Focus, the name of the function to be called when the tab card is topmost and its index tab loses focus.

New Logical Keys for Tab Controls

New logical keys, `NCARD` and `PCARD`, move to the next card and previous card respectively.

Database Interface API Changes

New Functions for the Database Interfaces

The new library functions associated with the database interfaces are:

`dm_convert_empty`
: Determine if empty numeric fields should be replaced with a 0. This setting is database-specific since some databases do not allow NULL values in numeric columns.

`dm_cursor_connection`
: Return the database connection for the specified cursor.

`dm_cursor_consistent`
: Determine if the specified cursor is on the default connection.

`dm_cursor_engine`
: Return the database engine for the specified cursor.

`dm_get_db_conn_handle`
: Return a handle to the database connection's structure.

`dm_get_db_cursor_handle`
: Return a handle to the database cursor's structure.

`dm_get_driver_option`
: Return the value of a database driver option.

`dm_odb_preserves_cursor`
: Check to see whether the ODBC datasource preserves the cursor on a commit or a rollback.

`dm_set_driver_option`
: Set the value of a database driver option.

`dm_set_max_fetches`
: Set the maximum number of rows in a select set.

`dm_set_max_rows_per_fetch`
: Set the maximum number of rows per fetch.

New Properties for the Database Interfaces

There is a new property associated with database connections:

Connection Pooling (`conn_pool_size`)
: In Panther/WebSphere applications specify the number of concurrent database connections.

New Commands for Database Interfaces

The new database interface commands are:

DBMS QUERY
DBMS RUN

Database Interface Command Changes

The following commands used in conjunction with the database interface have changed:

DBMS CATQUERY
DBMS DECLARE CONNECTION

Transaction Manager API Changes

New Library Functions for the Transaction Manager

The new library functions associated with the transaction manager are:

`dm_disable_styles`
: Suppress the enforcement of styles in the transaction manager.

`dm_enable_styles`
: Enable enforcement of styles in the transaction manager.

`dm_set_tm_clear_fast`
: Clear all fields in a server view.

`sm_get_tv_bi_data`
: Get before-image data.

`sm_tm_handling`
: Process the specified transaction manger functions for special insert, update, select and delete handling.

`sm_tm_old_bi_context`
: Specify the method of before-image processing.

The following library functions used in conjunction with the transaction manager have changed:

`dm_gen_change_select_list`
: For this function, do not use a local JPL variable as the target of a transaction manager fetch.

`sm_tm_inquire`
: A new argument, `TM_SV_SEL_COUNT` determines if an initial query will be performed in order to determine the number of rows in the select set. Five new arguments are available: `TM_CANCEL_ON_DISCARD`, `TM_CURRENT_COMMAND`, `TM_SAVE_COUNT`, `TM_SV_SEL_COUNT`, `TM_XA_TRANSACTION_BEGUN`.

`sm_tm_iset`
: Three new arguments are available: `TM_CANCEL_ON_DISCARD`, `TM_SV_SEL_COUNT`, `TM_XA_TRANSACTION_BEGUN`.

`sm_tm_pinquire`
: A new argument, `TM_COMMAND_ROOT`, identifies the root table view of the current command.

New Properties for the Transaction Manager

There are several new properties associated with the transaction manager. Some are settable via the Properties window and others are readable and/or writable only at runtime. They are:

Before Image Rows (`bi_string[`*`iter`*`]`)
: A widget, runtime-only, property. Provides access to the before image values of rows in the transaction manager as strings. The *`iter`* specification lets you walk through the list of rows.

Continue Function Name (`continue_func_name`)
: For table views in two-tier applications, specify the function for handling `CONTINUE` operations in the transaction manager for the specified server/table view. The Select Handling property must be set to Function Name.

Count Result (`count_result`)
: A table view, runtime-only, property. This readable/writable property holds the value returned from a count query (from a `TM_SELECT_COUNT` event), that is, the total number of rows in the result set. The value is examined to determine whether to query the user about proceeding with the normal `SELECT` statement.

Count Select (`count_select`)
: A table view property, located under Transaction, takes a value of Yes or No. Instructs the transaction manager whether or not to count the number of rows in a result set and compare it (stored in the server view's `count_result` property) to a specified threshold (Count Threshold property) value before actually fetching data. This property is readable and writable.

Default Transaction (`default_tran`)
: A runtime-only, read-only screen property that provides the name of the default transaction manager transaction. This property always contains the name, even if the transaction is not currently open, and can be used to stop and then re-start the default transaction when making runtime property changes.

Delete Function Name (`del_func_name`)
: For table views, specify the function for handling delete statements in the transaction manager for the specified table view. The Delete Handling property must be set to Function Name.

Delete Handling (`delete_handling`)
: For table views, select the method for handling delete statements in the transaction manager for the specified table view: SQL Statement Generation (`PV_HANDLING_SQL`), Function Call (`PV_HANDLING_FUNC`), or Nothing (`PV_HANDLING_NOTHING`).

Deleted Rows (`di_string[`*`iter`*`]`)
: A widget, runtime-only, property. Provides access to the values of deleted rows in the transaction manager as strings. The *`iter`* specification lets you walk through the list of deleted rows. Use in conjunction with the `num_del_images` property.

Insert Function Name (`ins_func_name`)
: For table views, specify the function for handling insert statements in the transaction manager for the specified table view. The Insert Handling property must be set to Function Name.

Insert Handling (`insert_handling`)
: For table views, select the method for handling insert statements in the transaction manager for the specified table view: SQL Statement Generation (`PV_HANDLING_SQL`), Function Call (`PV_HANDLING_FUNC`), or Nothing (`PV_HANDLING_NOTHING`).

Join Type (`join_type`)
: For link widgets, under Transaction, a subproperty of Type. When the link is identified as a server link, that is if the link's Type (type) property is set to Server (`PV_LNK_SERVER`), the Join Type property is available. It can be set to: Inner (`PV_INNER`) (default), Left Outer (`PV_LEFT_OUTER`), Right Outer (`PV_RIGHT_OUTER`), or Full Outer (`PV_FULL_OUTER`). This property lets you take advantage of SQL join facilities, whereby you can control the join operation of a `SELECT` statement that combines information from two database tables.

Number of Columns (`num_columns`)
: A read-only and runtime-only property associated with table view widgets. This property returns the number of columns belonging to a specific table view, or more specifically, the number of occurrences defined in the Columns (columns) property.

Number of Deleted Rows (`num_del_images`)
: A widget, read-only and runtime-only, property that returns the number of deleted rows in the transaction manager.

Primary Key Update (`primary_key_update`)
: A runtime-only application property determining how primary key changes are processed in the transaction manager: whether the row is updated or whether it is deleted and then inserted.

Regenerate SQL (`regenerate_ins_sql`, `regenerate_upd_sql`)
: If the transaction manager generates SQL statements, as determined by the Method property, specify if the SQL statement should be regenerated for each row in the table.

Save Function Name (`save_func_name`)
: For table views, specify the function for handling `SAVE` operations in the transaction manager for the specified server/table view. The Delete Handling, Insert Handling, or Update Handling properties must be set to Function Name.

Select Function Name (`sel_func_name`)
: For table views, specify the function for handling select statements in the transaction manager for the specified server/table view. The Select Handling property must be set to Function Name.

Select Handling (`select_handling`)
: For table views, select the method for handling select statements in the transaction manager for the specified server/table view: SQL Statement Generation (`PV_HANDLING_SQL`), Function Call (`PV_HANDLING_FUNC`), or Nothing (`PV_HANDLING_NOTHING`).

Service Transaction (`tm_transaction`)
: A runtime-only application property in JetNet and Oracle Tuxedo executables that determines whether a service is transaction-manager enabled and, if so, which transaction manager operation is to be performed.

Threshold (`count_threshold`)
: For table view widgets, this property is a subproperty of the Count Warning property when Count Select and Count Warning are set to Yes. Use to specify the maximum number of rows to fetch in a result set. If a result set (stored in the server view's `count_result` property) exceeds this value, the user is prompted before the data is actually fetched.

Update Function Name (`upd_func_name`)
: For table views, specify the function for handling update statements in the transaction manager for the specified table view. The Update Handling property must be set to Function Name.

Update Handling (`update_handling`)
: For table views, select the method for handling update statements in the transaction manager for the specified table view: SQL Statement Generation (`PV_HANDLING_SQL`), Function Call (`PV_HANDLING_FUNC`), or Nothing (`PV_HANDLING_NOTHING`).

Warning (`count_warning`)
: For table view widgets, this property is a subproperty of the `Count Select` property when `Count Select` is set to Yes. Use to specify whether the user is prompted, before the data is actually fetched, when the size of a result set (stored in the server view's `count_result` property) exceeds the value in the Count Threshold property.

Property Changes for the Transaction Manager

The property changes associated with the transaction manager are:

Fetch Directions/Directions (`fetch_directions`)
: The table view Fetch Directions property has been renamed to Directions and is located in the new Server View category.

  Both the table view Directions property and the screen Fetch Directions property have an additional value, none, which when set eliminates the possibility of doing `CONTINUE` command processing on a server view. `CONTINUE` functionality can consume system resources, therefore, using this property value can allow you to better control how a `SELECT` is issued against the table view.

Memo Text (`memo1...memo9`) properties for table views and links
: Under Identity, both table view and link widgets can now have Memo Text properties assignments.

Relations (relations)
: This property, which describes the relationship between two table views, has been refined into three sub-properties: `rel_child` (database column in child table view), `rel_parent` (database column in parent table view), and `rel_op` (type of relationship–join or lookup).

Other property changes for the transaction manager are:

Readable transaction properties
: Prior to Panther, almost all transaction properties were readable at runtime; only five were not. All transaction properties are now readable via the property API. They include widget properties (under Column Edits): Length (`column_length`), Precision (`column_precision`), Scale (`column_scale`), and Type (`column_type`).

Writable transaction manager properties
: If a transaction manager transaction is not in effect, all transaction manager properties are writable.

New Commands for the Transaction Manager

The new transaction manager commands are:

RELEASE

The transaction manager has a new command, RELEASE, which releases the database cursors when the transaction manager is active.

WALK commands

The WALK commands direct the transaction manager to traverse the transaction tree of an application screen. These commands have no processing attached to them in the transaction models so the traversal can be used to fire any transaction event functions.

WALK_DELETE	Traverses the tree in delete order.
WALK_INSERT	Traverses the tree in insert order.
WALK_SELECT	Traverses the tree in select order.
WALK_UPDATE	Traverses the tree in update order.

New Events in Transaction Manager Processing

If you write your own transaction manager event functions, three new slices were added to the TM_SELECT and TM_VIEW request events in order to check the size of the select set:

TM_SET_SEL_COUNT_FLAG
TM_SEL_COUNT_CHECK
TM_CLEAR_SEL_COUNT_FLAG

For database transactions, the following slices were added to the SAVE command:

TM_SAVE_BEGIN
TM_SAVE_COMMIT
TM_SAVE_ROLLBACK
TM_SAVE_SET_MODE

Web Application API Changes

Browser Events

In conjunction with VBScript support, the JavaScript Events category in the Properties window was renamed Browser Events. A new event is also available:

`OnMouseOut` event
: For JavaScript and VBScript, the `OnMouseOut` event for widgets is now available. This property lets you specify a JavaScript or VBScript function to execute when the mouse pointer leaves an area (in client-side image maps) or a link.

New Library Functions for Web Applications

There is a new library function associated with Web applications:

`sm_web_log_error`
: Write Web application errors to a log file.

New Properties for Web Applications

There are several new properties associated with Web applications:

Default Link (`default_link`)
: For Web applications, specify the URL location for this hyperlink. (This replaces the link property in previous releases.) If the property is specified for an array, it is the hyperlink location for every occurrence in the array. (See Item Link.)

HTML Max Loop (`html_max_loop`)
: For HTML templates using condition processing, specify the number of loop iterations to perform before terminating the process. The default setting is 1000.

HTML Max Nest (`html_max_nest`)
: For HTML templates using condition processing, specify the number of nesting levels. Each if, while, or include constitutes one level. The default setting is 20.

HTML Name (`html_name`)
: Read-only access to the converted HTML name, which is based on the Panther variable name, using the syntax `{{variable->html_name}}`.

Insert/Delete Buttons property (`ins_del_buttons`)
: For grid widgets in Web applications, if set to Yes (default), Insert (Insert Above and Insert Below) and Delete buttons are generated in the HTML representation of the grid widget. If set to No, the buttons are not generated under any circumstances.

OnMouseOut (`on_mouse_out`)
: For Web applications, under Browser Events, this property lets you specify a JavaScript or VBScript function to execute when the mouse pointer leaves an area (in client-side image maps) or a link.

Previous Form (`previous_form`)
: For Web applications, get the screen name as stored in the current cache file. Typically, this would be the name of the last screen that was accessed.

Stylesheet Data (`stylesheet_data`)
: Under Web Options, for inline style sheets, enter the style sheet specification.

Stylesheet Link (`stylesheet_link`)
: Under Web Options, specify the URL location of the style sheet. On the HTTP server, the style sheet should be located in the public documents directory.

Stylesheet Source (`stylesheet_source`)
: Under Web Options, specify whether the style sheet for the web application screen is included in the screen itself (Inline) or in a separate document (Link).

Stylesheet Type (`stylesheet_type`)
: Under Web Options, specify the type of style sheet to be used for the web application screen: CSS (cascading style sheets) or JavaScript.

Submit (`submit`)
: Under Web Options, setting this new push button property to No will keep the screen from being submitted back to the web application server when the button is pressed.

VBScript (`vbscript`)

Under Web Options, setting this new push button property to No will keep the screen from being submitted back to the web application server when the button is pressed.

Web ID (`webid`)

For Web applications, this application property obtains the name of the next cache file to be generated.

Property Changes for Web Applications

There are several property changes associated with Web applications:

Label (`label`) property

The Label property (`label`) is now available for grid widgets. The setting provides a caption for the HTML table in Web applications.

Link (`default_link`) property

The Link property (`link`) in previous releases has been changed to Default Link (`default_link`).

In addition, business graphs can be assigned a URL. If no value is set, the graph does not act as an HTML link.

Style property (`style`)

The screen subproperty of the Pixmap property now defaults to Tile instead of Center. This only affects newly created screens.

Dockable Toolbars

New Properties for Dockable Toolbars

There are new properties associated with the use of dockable toolbars in Windows applications:

Toolbar Allowed Sites (`toolbar_allowed_sites`)

For toolbars in Windows applications, a runtime application property sets the frame placement for the toolbar using one or more of the following bit flags: `PV_TOOLBAR_FLOAT`, `PV_TOOLBAR_TOP`, `PV_TOOLBAR_BOTTOM`, `PV_TOOLBAR_LEFT` or `PV_TOOLBAR_RIGHT`.

Toolbar Coordinates (`toolbar_x_position`, `toolbar_y_position`)
: For toolbars in Windows applications, runtime application properties set the screen coordinates of the upper-left corner of the floating toolbar.

Toolbar Current Site (`toolbar_current_site`)
: For toolbars in Windows applications, a runtime application property sets the current placement of the toolbar using one of the defined bit flags: `PV_TOOLBAR_FLOAT`, `PV_TOOLBAR_TOP` (default), `PV_TOOLBAR_BOTTOM`, `PV_TOOLBAR_LEFT`, or `PV_TOOLBAR_RIGHT`.

Toolbar Hidden (`toolbar_hidden`)
: For toolbars in Windows applications, a runtime application property sets whether the toolbar is currently displayed using `PV_YES` and `PV_NO`. Users can hide the toolbar by clicking on the X in the upper-right corner of the menu.

Other API Changes

New Properties

These are the remaining new properties not covered in previous sections:

Endsession (`endsession`)
: For Windows applications, an application property which specifies the function to call which closes down the application when Windows sends the `WM_ENDSESSION` message.

Java Tag (`java_tag`)
: Under Identity, specify the Java class implementing the event handler for this object (screen, service component, widget).

Max Bundles (`max_bundles`)
: A runtime-only application property specifying the number of JPL bundles available for send and receive commands. It defaults to ten bundles (including the unnamed bundle) if unspecified.

Queryendsession (`queryendsession`)
: For Windows applications, an application property which specifies the function to call which prepares to close the application when Windows sends the `WM_QUERYENDSESSION` message.

Screen Type (`screen_type`)
: For screens and service components in distributed applications, a property under Identity which displays whether the screen object is a client screen or service component.

Property Changes

The following properties have changed in Panther:

Font properties
: Screen and widget font properties that identify, what was JAM-specific fonts, have been updated to be Panther-specific fonts both in the Properties window and in the configuration map file; the JAM modifier has been eliminated.

Help properties (`help_screen`)
: The menu property, `mni_jam_help` (menu item Help property) is now `mni_help`.

The screen property (JAM Help property) and its corresponding mnemonic, `jam_help_screen` are now Help Screen and `help_screen`, respectively.

Style property (`style`)
: The screen subproperty of the Pixmap property now defaults to Tile instead of Center. This only effects newly created screens.

Application Properties

The `@jam` property shortcut for the application name has been replaced with `@app()`. `@jam` will continue to work for backward compatibility.

Text Selection

A new series of logical keys have been added for selecting text:

EXTFB	extend selection to start of field or list box
EXTFE	extend selection to end of field or list box
EXTL	extend selection with left arrow in text field
EXTLB	extend selection to start of line in text field

`EXTLE`	extend selection to end of line in text field
`EXTPD`	extend selection down one page in text field or list box
`EXTPU`	extend selection up one page in text field or list box
`EXTR`	extend selection with right arrow in text field
`EXTWL`	extend selection one word left in text field
`EXTWR`	extend selection one word right in text field
`SLALL`	select entire text field
`SLWRD`	select current word

In addition, `EXTD` and `EXTU` now also apply to text fields as well as list boxes.

New Library Functions

There are several new library functions for use in application building:

`sm_file_exists`
Checks whether a file exists.

`sm_file_move`
Copies a file and deletes its source.

`sm_file_remove`
Deletes a file.

`sm_ldb_fld_get`
Copy data from LDBs to specific fields.

`sm_ldb_fld_store`
Copy data from specific fields to LDBs.

`sm_l_open_syslib`
Opens a library as a system library.

`sm_list_objects_count`
Counts the widgets contained by an application object.

`sm_list_objects_end`
Destroys an object contents list.

`sm_list_objects_next`
Traverses the widgets contained by an application object.

`sm_list_objects_start`
: Constructs a list of widgets contained by a Panther object.

`sm_load_screen`
: Preload a screen into memory.

`sm_menu_change`
: Set a menu's properties.

`sm_mnitem_create`
: Insert a new item into a menu.

`sm_msg_del`
: Delete a message set from memory.

`sm_msg_read`
: Read messages from a memory block.

`sm_mw_PrintScreen`
: In Windows executables, print Panther screens, sending either the current Panther screen or all the screens in the MDI frame to the printer.

`sm_unload_screen`
: Unload a screen from memory.

Changed or Discontinued Functions

The following library functions have been changed or discontinued:

`sm_fi_open`
: Is no longer documented. `sm_fi_open` was used to find a file (along the Panther's search path) and open it in binary read-only mode. Use `sm_fi_path` instead to search along Panther's search path. Then call `fopen` (a standard C function) to open the file in any way you choose (it does not limit you to binary read-only mode).

`sm_inquire`
: A new parameter, `I_INERROR`, is available to determine if a message box is being displayed.

`sm_msgread`
: Has been replaced with the following new functions:

- `sm_n_msg_read`—Reads messages from a named file with standard file lookup protocol.

- `sm_d_msg_read`—Reads messages from the default message file (`SMMSGS` variable.
- `sm_msg_read`—Reads messages from a memory block.
- `sm_msg_del`—Deletes a message set from memory.

The message classes have also been updated; `FM_MSGS`, `JM_MSGS` and `JX_MSGS` messages are now located in `SM_MSGS`. The value for `WB_MSGS` has also been updated. Any instances of `sm_msgread` in a Panther application should be updated to the new message classes.

Database Interface

For additional information, refer to page 1-27, "Database Interface API Changes."

Improved SQL Processing

`DBMS SQL` statements that specify data modification and do not return data (`INSERT`, `UPDATE`, and `DELETE` statements) are executed by simply passing the SQL statement to the database immediately to process the statement quickly and efficiently. For SQL statements that return rows (`SELECT`), the process includes a prepare and execute cycle. This means that the database is first notified where to put the data (if any), and then tells the database to execute the SQL.

The method used to determine if a SQL statement returns rows is to execute the SQL statement and see if it returns rows. If it does, it goes through the prepare and execute cycle—essentially executing the `SELECT` statement twice. If the statement is a stored procedure which inserts a row and then selects back data, the stored procedure is executed twice and therefore causes two copies of the row to be inserted.

The new method of SQL statement processing includes two new DBMS statements which will improve `SELECT`-type processing and performance:

- `DBMS QUERY`—Executes the SQL statement based on the assumption that it may or may not return data.

- `DBMS RUN`—Executes the SQL statement immediately, which assumes that no data is returned from the database.

Performance is improved because:

- Data-modification (non-`SELECT`) statements are not executed twice.
- Since it can be determined ahead of time whether or not to expect fetched rows, it takes less time to execute a `SELECT` statement.

Specifying Variables in DECLARE CONNECTION

The new recommended syntax for `DBMS DECLARE CONNECTION` allows the values for the connection options to contain spaces or punctuation characters. Use the `WITH` keyword in the statement (instead of `FOR`) and connect the option and value with an equal sign in comma-separated pairs. Variables no longer need to be colon-expanded; strings must still be in quotation marks. The following example contains two variables for the user and password and a quoted string for the database path:

```
DBMS DECLARE c1 CONNECTION WITH \
    USER=user, PASSWORD=pword, \
    DATABASE="C:\Program Files\Prolifics\videobiz"
```

Since the variables are not colon-expanded in this variant, the values will not appear in error messages and trace statements.

Support for Long Filenames

`DBMS CATQUERY` now supports writing to a filename containing spaces or punctuation. Create a variable for the filename and use the variable in the new command syntax:

```
vars query1 = "query results"
DBMS CATQUERY TO FILENAME query1
```

This syntax does not replace `DBMS CATQUERY TO FILE`, which is supported unchanged.

Transaction Manager

For information on API changes (functions, properties, commands, slice events), refer to page 1-28, “Transaction Manager API Changes.”

Transaction Manager Common Model

The previous set of database-specific transaction models delivered with Panther have been replaced with smaller, more manageable models. In addition to the database-specific model, each engine also accesses a common transaction model, containing the functionality common to all of the database engines. The source code for the new database-specific transaction models is provided and can be modified to make global changes in transaction manager functionality. The common model should not be modified; however, the source code is available for reference.

Having a database-specific model expands the event processing in the transaction manager. As in previous versions, the transaction manager first checks to see if an event function has been specified for the event. If so, it is processed; otherwise, the transaction manager proceeds to the database-specific transaction model. If database-specific processing for the event is required, it must be contained in this model. Otherwise, the transaction manager proceeds to the common transaction model and performs the processing defined there.

To call the common model in addition to an event function and the database-specific model, have the event processing in the event function and the database-specific model return `TM_PROCEED`, which passes the processing to the next level. The common model is always called for `TM_START` and `TM_FINISH` events.

The common model provides plausible processing for every event known to the transaction manager. This includes default behavior for the database transaction events. While a majority of the database-specific transaction models set the mode to initial after a database transaction is committed, the common model does not, leaving this for the database-specific transaction models.

If you have revised an existing transaction model, the revised version can continue to serve as a database-specific transaction model. Since none of the previously distributed transaction models return `TM_PROCEED`, unless this return value has been explicitly coded, the transaction manager will only access the common model for the new transaction manager events.

If you have implemented a transaction manager event function and use one of the existing transaction models, there is no visible effect with the replacement of the old models with the new.

Web Application Development

For additional information, refer to page 1-35, "Web Application API Changes."

Initialization File Changes

Initialization File Settings

Web initialization files have the following new initialization variables:

- `NumServers`—The number of jserver processes, or concurrent users, for an application. This setting replaces the `MinServers` and `MaxServers` settings which are no longer available. Web applications from previous releases need to update their web initialization file to the new variable.
- `IdleServerTimeOut`—Number of seconds that a jserver process will wait for an incoming request before exiting.
- `EnableWebid`—Activates the caching process which uses the webid property to obtain the cache file so that it can be specified in the URL.
- `ImageDir`—Graphics can be fetched using the HTTP protocol, rather than the Panther web application server. In the web application's initialization file, specify a sub-directory of the HTTP server's document root directory in the

ImageDir variable. When development of the application is complete, copy the graphics to this sub-directory.

- `ListenQueueLength`—The length of the listen queue. This approximately represents the number of web requests that can be waiting for an available jserver.
- `PadOptionMenus`—For option menus, pads the text with trailing and HTML spaces (nbsp) if set to Yes. To pad only the first occurrence, set this option to First. Setting this option to Yes matches the behavior in previous versions of JAM and Panther.

One Initialization File

In previous releases, a Panther web application read the values from a global initialization file before reading the application's initialization file (`appName.ini`). This was done so that the `proweb.ini` or `jamweb.ini` could define any global values which are common among all web applications, but it caused problems for application maintenance. Therefore, the distributed proweb.ini file will no longer be read when you have an application initialization file. Only one initialization file is read for each Web application.

If you are using a global initialization file (`proweb.ini` or `jamweb.ini`) to set global parameters, merge all global values into your application-specific initialization file.

New Web Applications

A new Web-based utility, the Web Setup Manager, will help write and configure the files needed for your Web application.

For step-by-step instructions, refer to Appendix B, “Web Setup Manager,” in *Web Development Guide.*

HTML Template Changes

HTML templates behave as Panther screens, allowing you to have the flexibility of how the HTML is created tied in with the power of the Panther backend. In the HTML Template property, you specify the name of the HTML document to use in conjunction with the Panther screen.

HTML Template Caching

The cache data for a Panther screen utilizing an HTML template can be maintained, and the template will be updated dynamically to associate the cache file with it. The HTML template must contain the `{{form:info}}` template tag.

HTML Template Tags

The syntax for HTML template tags has changed from <<*`variable`*>> to `{{`*`variable`*`}}`. In addition, there are new tags for HTML templates:

`{{form:info}}`	Interpolates hidden data needed to submit the form.
`{{form:output}}`	Outputs the entire form in the HTML format Panther would normally use.
`{{form:script}}`	Generates JavaScript procedures based on edits and validations of widgets on the form.
`{{form:tag}}`	Generates the start `<FORM>` tag with `ACTION` and JavaScript attributes.
`{{value:`*`variable`*`}}`	Generates the value corresponding to the specified variable.
`{{emit:`*`object`*`}}`	Generates the HTML that Panther would normally output for the specified object.

`{{while:`*`condition`*`}}`
`{{if:`*`condition`*`}}`
`{{else:}}`
`{{elseif:`*`condition`*`}}`
`{{end:}}`
	Performs condition processing based on a JPL boolean expression.

`{{include:`*`filename`*`}}`	Include the specified file.
`{{eval:`*`statement`*`}}`	Evaluate a simple JPL statement.

Two application properties are associated with HTML loop processing: `html_max_loop` to limit the number of loop iterations and `html_max_nest` to limit the number of nesting levels.

New Syntax for Specifying Variables

The syntax for accessing Panther variables has changed from <<`variable`>> to {{`variable`}}. This new syntax can be used in JavaScript, VBScript, the Custom HTML properties (such as Prefix Markup and Suffix Markup), and HTML templates.

Web Entry Processing

When screens are submitted at runtime, Panther variables (`@web_action`, `@web_action_widget` and `@web_action_occurrence`) contain information about the push button that was pressed and the widget's object ID and occurrence number, if applicable. These variables can be accessed in `web_enter` processing. For more information, refer to “Web Entry Context Flags” on page 5-9 in *Web Development Guide*.

Caching Application State

The state of the application can now be obtained when performing a `GET` for Panther files. In previous releases, invoking screens and reports via a `GET` caused the state information to be lost. The following can now be accomplished:

- Hyperlinks—A hyperlink can be used to obtain a Panther screen that has access to application state information on the server.
- Frames—The HTML file which defines the frames would be set as an HTML template. Then, using the procedure for HTML templates, the <`FRAMESET`> can specify a series of screens sharing the same cache file.
- HTML Template – The name of the cache file can be encoded into the HTML template using the <<`widget_name`>> syntax. The template can call subsequent Panther screens via GET with this cache name specified. The called screens would then have access to the cache information.

To implement this caching behavior, two new application properties are available:

Previous Form (`previous_form`)
: Gets the screen name as stored in the current cache file. Typically, this would be the name of the last screen that was accessed.

WebID (`webid`)

Obtains the name of the next cache file to be generated.

To access the cache file, a new name=value pair can be encoded as part of the URL:

`@webid=cacheFile`

For more information, refer to “Getting Screens from the Server” on page 6-4 in *Web Development Guide*.

Requester Executables

If you are using an ISAPI- or NSAPI-compliant HTTP server, use the new ISAPI and NSAPI versions of the requester executable, instead of the CGI version, for faster processing of your HTTP requests.

Windows Servers

If you are using Windows as your Web application server:

- Use NTFS as the disk file system, not FAT, to improve your system performance.
- Use the new ISAPI and NSAPI versions of the requester executable, instead of the CGI version, for faster processing of your HTTP requests.
- Set your Web application to run as an service using the monitor utility. As an service, you can have the application automatically start when the server is rebooted. You can also specify that other services needed by the Web application, such as database access, be restarted first.

Running Java Servlets

A Panther web application can run as a Java servlet. For more information, refer to Appendix D, “Using Java Servlets,” in *Web Development Guide*.

Determining Mouse Location

Two JPL globals, `@web_image_click_x` and `@web_image_click_y`, contain the X and Y coordinates of the user's mouse click for use in JPL procedures.

Widget Positioning in Web Applications

In order to control widget positioning in Web applications better, a COLS attribute has been added to the table definition in the generated HTML. This will affect the widget positioning for screens built in previous versions of Panther. For new screens, there is a higher correlation between the GUI position and the HTML position.

Additionally if you create multiple boxes or grids which are aligned on two sides in the screen editor, they will now appear to be aligned in the Browser.

Note that one browser may create its widgets using different font families and sizes than another browser, or than the Panther screen editor. This results in screens that can appear slightly different (more or less space between widgets) from browser to browser, or from browser to screen editor.

Some helpful tips are:

- If you find that widgets are being pushed out further than expected in the generated HTML, place repeated design elements together in a box.
- If you want a group of widgets to be spaced close together, select all these widgets; then use Edit→Space→Custom to distance them at 0, either horizontally or vertically.
- Specifying font sizes in the Panther screen editor results in better positioning than using header tags, such as `H1`.
- HTML does not support overlapping widgets. In order to quickly detect overlapping widgets, borders were added to radio buttons, check boxes and labels to indicate whether they overlapped another widget. These borders do not appear at runtime. Additionally, to ensure that widgets do not overlap, use the new menu options which find overlapping widgets (Find→Overlapping Widgets or Options→Check Overlap on Screen Save).

Errors in Web Applications

Additional error text has been added for the requester, dispatcher, and jserver programs. Errors for the Web application server have also been added to the Panther message file.

Web Gallery Samples

The following changes have been made to the gallery of Web application samples:

ActiveX and VBScript

You can embed ActiveX controls in your Panther screens and write VBScript to manipulate them on the browser. This example contains JPL procedures that dynamically generate VBScript to populate an ActiveX control. This example also demonstrates how to write VBScript to get values from an ActiveX control and copy them to hidden Panther fields in order to send them back to the server.

Templates

Demonstrates the use of the Panther Template property to present data from a Panther screen using the format of a custom HTML file. This HTML file can also be submitted back to Panther for normal processing.

By clicking on the scroll buttons of the grid, the custom representation of the data is also updated at the bottom of the screen.

Web Wizard Defaults

For wizard-generated Web screens, the default values have changed for some properties. The Border, Title Bar, and System Menu properties now default to No.

Naming Conventions

With the change in naming conventions, smrepost.jam becomes `smrepost.scr.`

Reports

Converting ReportWriter 6 Reports

If you need to modify a report created with JAM/ReportWriter 6, you must first convert it to a Panther report file using the `rw6toprl` utility. From the command line, type:

```
rw6toprl [-fgkm] rw6Report pantherReport
```

`-f`
: Output file can overwrite existing file.

`-g`
: Use GUI coordinates when converting the report. This option positions widgets using the GUI decimal coordinates, instead of integer coordinates. Used this option if PostScript and proportional fonts have been specified in the JAM ReportWriter 6 source file.

`-k`
: Retain ReportWriter 6 widget types in the output file.

`-m`
: Merge included files name in input file into output report file.

The utility converts GUI coordinates to column and row (whole-number grid units, as though in character mode) coordinates to position widgets. To ensure GUI coordinates, run `rw6toprl` with the `-g` option.

Modifying Reports from Previous Versions

As of Prolifics 2.5, several changes were made inside the report editor. To edit report files created in previous versions, you need to manually rescale the grid in the report editor using either of the following methods:

- Set the report level font to a new value and then set it back to the original setting.

- Set the Grid Height and Grid Width properties manually (to the values already displayed in the Properties window). For example, if the Grid Height property is set to 0.17, enter: 0.17. If the Grid Width property is 0.10, enter: 0.10.

Setting Widget Size

To resize a widget in ReportWriter, do not drag the widget by its edges; set the widget size by setting the font size.

Printing PostScript

In Windows, if reports are generated using the `driver=postscript` option, those reports must be printed using a Windows print driver that supports PostScript. Some printer models have more than one printer driver; install the PostScript version for PostScript reports.

Report Utilities

The `r2asc` utility has been superseded by `f2asc`; therefore, to convert a report between binary and ASCII output, use `f2asc`. The `rinherit` utility has been superseded by `binherit`; therefore, to batch update all reports with inherited values from your application's repository, use `binherit`.

Upgrading to JetNet

JetNet, Panther's three-tier middleware product, is available for UNIX and Windows server. This section lists the additional features available in that product.

Editor

- On the File menu, there are menu options for creating, opening and saving service components.
- New Service properties are provided to define services to implement database access operations with the transaction manager. Table views can have the Insert, Update, Delete, and Select service properties set. Link widgets can have the Validation Service property set. Table view widgets on client screens are assigned service property values by the screen wizard to identify the services.

  If you use the screen wizard to create screens, these properties are automatically defined.
- Connection to the middleware is provided via a dialog box accessed by choosing File→Open→Middleware Session.

Screen Wizard

- The screen wizard now creates service components as well as client screens. The Application Model dialog prompts you to choose between two- or three-tier architecture, and whether to create a client screen, a service component, or both.
- Three-tier client screens created with the screen wizard do not have Continue (i.e., Next, Prev, First, Last) options built into the screen or menu bar. These operations are not available in a three-tier architecture because a server has to service multiple client requests and cannot keep track of the state of the database between requests, and also because there is no guarantee that the same server will handle repeated requests from a client.
- Screens created with the screen wizard use bitmaps. The bitmaps reside in the distributed client library `client.lib`.
- New transaction manager operation Service properties are automatically set to pre-defined services for the transaction manager operations. No coding is necessary.
- When selection screens are created for you in the screen wizard, the name extensions differentiate between two-tier screens and between client screens and service components in three-tier. The extension for two-tier screens remains the

same: `.itm`. In three-tier, the screen for the client is given the extension `.cit`. The extension for its corresponding selection service component is `.sit`.

- New server JPL code that implements service calls for database access is provided for you in the service component's JPL Procedures property and in `smwizsrv.bin`. `smwizsrv.bin` is distributed in the server library `server.lib`. The procedures contained in the service component's JPL procedure call common functions in the `smwizsrv.bin` module.

Menu Bar Editor

- Open and Save recognize local and remote libraries, not standalone files or screens. A library must be opened, and all menu bar scripts must be stored in a library.
- Connection to the middleware is provided. The menu bar editor becomes a client when the connection is made.

Styles Editor

- The styles editor recognizes libraries, not standalone disk files. A library must be opened, and styles files must be stored in it. The default style file, `styles.sty`, is distributed in `client.lib` and `server.lib`.
- Upon invoking the styles editor, all open libraries are searched for the file styles.sty.
- Style files must be saved to a library. If the file is new, the Save As Library Member dialog box opens where you can save it with a name to a library.

JIF Editor

- File menu options reflect storage of screen and JPL modules in local and remote libraries and support source control management if the library is under source control management.
- Connection to the middleware from the JIF editor. This permits interprocess communication, allowing access to remote libraries and automatic updating of

servers when the JIF is changed and saved. The JIF editor alerts servers when changes have been made to a service group that is advertised at server startup.

- Service options screen for specifying:
  - Transaction type; one of: select, insert, update, delete, or link validation
  - Asynchronous mode
  - Reply expected
  - Outside transaction
  - Exception handler
  - Unload handler
  - Priority
  - Service component caching choices: on advertise, on first call, or none
- Queue specifications for users of Oracle Tuxedo.
  - Queue menu for create, update, and delete queue screens.
  - View menu provides access to View Queues screen for viewing queues and queuespaces, as well as screens for View Service and View Groups.

Debugger

For JetNet/Oracle Tuxedo applications in order to debug your server, you must configure a debuggable server. For more information, refer to "Server Details" on page 3-22 in *JetNet/Oracle Tuxedo Guide*.

The debugger will only access compiled JPL code saved in libraries. If compiled JPL is put in a library outside of the editor (with the `jpl2bin` and formlib utilities), the binary JPL file must include the JPL source code. This means that you must not use the `-r` flag with `jpl2bin`. For more information on compiling JPL source code, refer to `jpl2bin` on page A-20 in *Application Development Guide*.

Service Components

Service components reside on the server, and are used to map data between client screens and services. In a running application they are not visible to the user. During development, if you execute a debuggable server, service components can be viewed in Test mode, or you can test it like a client screen if you have a direct connection to the database. They can be created using the screen wizard at the same time you create client screens.

JIF

For JetNet/Oracle Tuxedo applications, the JIF is a file that contains service information required by both the clients and servers of your application. A JIF is created and edited using the JIF editor and can be set in the environment using `SMTPJIF`.

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

Administration Utilities

Panther provides several utilities to configure and operate your three-tier application. For the Oracle Tuxedo middleware adapter, use utilities provided by Oracle Tuxedo that perform similar functions, such as `tpadmin`.

- `jetman`, the JetNet manager, is an interactive utility that performs all the functions necessary to configure, boot, monitor, and shutdown JetNet and your application's servers. `jetman` contains all the functionality of these utilities: `rbconfig`, `rbboot`, and `rbshutdown`.
- `rbboot` is used to start JetNet and boot up your servers.
- `rbshutdown` shuts down JetNet and your servers.
- `rbconfig` provides an alternative method for creating a minimal JetNet configuration file.
- `rblisten` allows application servers to run on multiple machines.
- `rb2asc` converts a JetNet configuration file to ASCII and vice versa.

Environment Variables

- A single declaration of `SMFLIBS` can now point to multiple libraries. Separate directories with a vertical bar (|) or use the convention used by your operating system for listing multiple directories in path. (For UNIX, this is a colon, and for Windows, it is a semi-colon.) The default setting automatically opens `client.lib`, `server.lib`, and common.lib.
- New middleware API-specific variables used to connect to the middleware: `SMRBCONFIG`, `SMRBHOST`, and `SMRBPORT`.
    - The JIF file for your application can be set in `SMTPJIF`.

Database Error Handling

Errors resulting from database access are handled differently when the server is connected to the database, as opposed to two-tier processing where the client is connected directly to the database. Default error handling on the server also depends on the environment: development or production.

There are new default database error handlers for servers. The DBMS `ONENTRY` default error handler logs each DBMS command in the development environment, but does nothing in a production environment. The DBMS `ONERROR` function default error handler logs all DBMS errors to the central user log file in both the development and production environments.

Team Development

In JetNet and Oracle Tuxedo executables, developers can have personal copies of screens in library files and services in the JIF in order to make and test changes during development.

A new menu option, Options→Service Alias allows you to specify a user identifier to use when testing services. For more information, refer to “Using Service Aliases to Test Services” on page 5-8 in *JetNet/Oracle Tuxedo Guide*.

To use this feature, the application server must have a service alias defined. Once defined, the library function `sm_tp_get_svc_alias` returns the value of the service alias for the application server.

Transaction Model

A JetNet-specific transaction model is provided for applications that use the transaction manager in the client side of your application: jetrb1. This database- and middleware-independent model is designed to process transaction manager events by requesting service calls, for example, when:

- You convert an existing two-tier application to three-tier (refer to the `clnt2svr` utility).
- You use the screen wizard to create your screens and service components.

progserv

A conversion server, progserv, is provided to process service requests made by client screens that use the transaction manager but do not have Service property specifications, such as screens created from a `clnt2svr` conversion.

JetNet and Oracle Tuxedo Event Handling

For JetNet/Oracle Tuxedo applications, there is a middleware layer of event handling. Several event types are defined, and built-in handlers are provided for both development and production executables. Default handlers are installed and can be overridden by using new, runtime, application-specific property settings.

For information on middleware API events and event handling, refer to Chapter 6, “JetNet/Oracle Tuxedo Event Processing,” in *JetNet/Oracle Tuxedo Guide*.

API Changes for JetNet and Oracle Tuxedo Applications

JPL Commands

There are several JPL commands associated with use of the JetNet and Oracle Tuxedo applications:

`advertise`	`notify`	`unload_data`
`client_exit`	`service_call`	`wait`
`client_init`	`service_cancel`	`xa_begin`
`jif_check`	`service_forward`	`xa_end`
`jif_read`	`service_return`	`xa_commit`
`log`	`unadvertise`	`xa_rollback`

Note that each new JPL command represents a potential name conflict with existing variable and field names in your application. All JPL commands are reserved keywords.

The receive command has been enhanced to provide middleware support. It is used to receive message data, for example by service routines to receive client data.

For more information on these commands, refer to *Programming Guide*.

Library Functions

There are several library functions associated with use of the JetNet and Oracle Tuxedo:

`sm_tp_free_arg_buf`	Frees up memory allocated by argument list generation functions.
`sm_tp_gen_insert`	Generates an argument list of fields for an `INSERT` operation.
`sm_tp_gen_sel_return`	Generates a list of fields for the returned select set of a `SELECT` or `VIEW` operation.
`sm_tp_gen_sel_where`	Generates a list of fields for the `WHERE` clause of a `SELECT` or `VIEW` operation.
`sm_tp_gen_val_link`	Generates a list of fields to be validated in a validation link operation.

`sm_tp_gen_val_return`	Generates a list of fields for the returned select set of a validation link operation.
`sm_tp_get_svc_alias`	Returns the value of the service alias for the application server.
`sm_tp_get_tux_callid`	Returns the Oracle Tuxedo-specific ID for a service request call.

Properties

There are several properties for JetNet and Oracle Tuxedo applications:

Table view service properties:

`delete_service`	`insert_service`
`select_service`	`update_service`

Runtime application properties:

`agent_type`	
`devserv_id`	`hdl_advertise`
`hdl_exception`	`hdl_jif_changed`
`hdl_message`	`hdl_post_request`
`hdl_post_service`	`hdl_pre_request`
`hdl_pre_service`	`hdl_request_received`
`hdl_server_exit`	`hdl_unadvertise`
`hdl_unload`	`tp_async_poll_interval`
`tp_block`	`tp_commit_return`
`tp_exc_code`	`tp_exc_msg`
`tp_exc_names`	`tp_mon_exc_code`

`tp_mon_exc_msg`	`tp_return`
`tp_severity`	`tp_severity_names`
`tp_signal_restart`	`tp_svc_cache_size`
`tp_svc_outcome`	`tp_svc_return`
`tp_this_call`	`tp_timeout`
`tp_tran_level`	`tp_tran_status`
`tp_unsol_poll_interval`	

Runtime service request properties:

`call_client`	`call_in_transaction`
`call_initial_text`	`call_no_reply`
`call_origin`	`call_priority`
`call_security_key`	`call_svc_name`
`call_text`	

Link widget service property:

`validation_service`

Migrating a JAM Transaction Manager Application

If you are upgrading a two-tier application that uses the transaction manager to three-tier–either a pre-Enterprise application or a two-tier Panther application– you must use the `clnt2svr` (client-to-server) utility provided with the Panther distribution.

The `clnt2svr` utility creates three-tier client screens and service components from the two-tier client screens in the library you provide as input. Upon completion, the service components, and optionally any JPL procedures associated with the original client screens, are placed in a library which can be used as the server library in a three-tier application. Also, the client screens are updated to use the default three-tier transaction model jetrb1, and stored in a new client library. The input library is left unchanged.

Certain property values that existed on the client screens are removed from the service components created from them in order to avoid unnecessary entry processing on the server.

To convert your two-tier JAM application to a three-tier Panther application:

1. If your two-tier screens are not in a library, run `formlib` and create a two-tier client library from all client screens and client JPL.
2. From the command line, type:

   ```
   clnt2svr inputLib
   ```

   A new client library (`cl.lib`) and a server library (`sv.lib`) are created from the input client library, which is left intact.
3. Create a JetNet configuration file using the JetNet manager. Include in the configuration a conversion server (progserv).
4. Set `SMFLIBS` on the client and server to recognize and open the new libraries on application startup.

   `clnt2svr` is described in more detail on page A-2 in *JetNet/Oracle Tuxedo Guide*.

Upgrading an Existing Application

If you are not ready to convert your existing application to three-tier, you need to put all the screens, modules and bitmaps in a library in order to use it with Panther. Run `formlib` and create a client library from all client screens and client JPL, or add them to the distributed client library (client.lib).

Consideration needs to be made for existing JPL code. JPL modules must be compiled before they are put in libraries. JPL can be compiled in two ways:

- Open the JPL module in the editor and save the JPL to an open library. The JPL is automatically compiled.
- Run `jpl2bin` on your existing JPL modules to compile them. If your JPL modules include a file extension, such as `.jpl`, you need to keep the extension

for the binary version. This is to ensure that public calls of the modules, such as public `mymodule.jpl`, will still work. To do this, use the `-e-` option to jpl2bin to preserve the existing file extension, but first copy the original ASCII file to another location so that it doesn't get overwritten. For example, to compile the JPL module mymodule.jpl and preserve the file extension, do:

```
cp mymodule.jpl old/mymodule.jpl
jpl2bin -b -f -e- mymodule.jpl
```

Upgrading to Panther for IBM WebSphere

Refer to *Panther for IBM WebSphere Developer's Studio* for information about:

- Configuring your Panther/WebSphere environment
- Building Enterprise JavaBeans in Panther
- Building client screens that call Enterprise JavaBeans
- Deploying Panther-built Enterprise JavaBeans on WebSphere Application Server

Documentation

Documentation Titles

The titles of some manuals have changed in this release:

JAM/Prolifics 2.5 Title	Panther Title
Administration Guide	*JetNet Guide* *Oracle Tuxedo Guide* *COM/MTS Guide* *WebSphere Developer's Studio*
Editors Guide	*Using the Editors*
Language Reference	*Programming Guide*
Tutorial	*Getting Started*

A new *Quick Reference* manual has been printed, containing a list of the property names, library functions, JPL commands, and transaction manager commands. The properties reference section of that manual is available online. For changes to the Quick Reference since its printing, refer to page 1-66, “Quick Reference Changes and Corrections.”

The *Application Development Guide* has been totally re-organized in order to illustrate a typical development process.

Online Documentation

Panther documentation is now available in HTML and PDF formats. For more information about Panther online documentation, refer to Appendix A, “Panther Online Documentation,” in *Installation Guide*.

Documentation Changes and Corrections

Other documentation changes not listed in previous sections are:

ASCII JPL modules
: Even though it is recommended that all JPL modules be placed in libraries, JPL modules can be in ASCII format on disk. This has been added back to the Programming in JPL chapter.

CGI variables
: In Web applications, the variables containing the HTTP header information, such as `@cgi_request_method`, are now referred to as HTTP server variables.

`cgi-bin` directory
: With the addition of ISAPI and NSAPI requester executables, the cgi-bin directory is now referred to as the program directory or scripts directory.

Reports
: The first release of Panther documentation incorrectly included descriptions of the Bar Height and Portable Placement properties. These properties are not in the Panther product.

`sm_card_val`
`sm_tw_val`
: The first release of Panther documentation incorrectly included these functions. For tab card validation, use `sm_validate`.

`sm_msg_del`
`sm_msg_read`
: Initial releases of the Panther documentation did not have an updated list of the message prefixes or message classes. `FM_MSGS`, `JM_MSGS` and `JX_MSGS` messages are now located in `SM_MSGS`. The value for `WB_MSGS` has also been updated.

`sm_prop_get_str`
: The documentation has been updated to say that this function stores the returned data in a pool of buffers that it shares with other functions so you need to copy or process this data immediately.

Traversal properties for table views
: Traversal properties for table views and server views in the transaction manager return the table view or server view name as a string, not as an object ID.

Quick Reference Changes and Corrections

The Panther Quick Reference which was printed in November 1999 does not have the following changes.

Configuration

Text Selection Keys - Windows

Refer to “Text Selection” on page 1-39 for the list of new logical keys.

`SMIBMVJAVA, SMPROVIDERURL, SMTPCLIENT, SMTPINIT, SMWSADMIN`

Refer to “WebSphere Variables” on page 1-19 for a description of the configuration variables in Panther for IBM WebSphere.

Functions

`sm_com*` Functions

The functions for components supersede the functions for COM components released in Panther 4.0 and 4.1.

COM Function	Panther 4.2 Replacement
`sm_com_call_method`	`sm_obj_call`
`sm_com_get_prop`	`sm_obj_get_property`
`sm_com_log`	`sm_log`
`sm_com_obj_create`	`sm_obj_create`
`sm_com_obj_destroy`	`sm_obj_delete_id`
`sm_com_onerror`	`sm_obj_onerror`
`sm_com_raise_exception`	`sm_raise_exception`
`sm_com_receive_args`	`sm_receive_args`
`sm_com_return_args`	`sm_return_args`
`sm_com_set_prop`	`sm_obj_set_property`

`sm_obj_sort`, `sm_obj_sort_auto`

New functions for sorting data in arrays and grids.

Properties

Column Click Action (`column_click_action`)
: For widgets in grids, under Format/Display, specify the action–sort or custom function–that occurs when a user clicks on the grid column header.

Column Click Function (`column_click_func`)
: For widgets in grids, under Format/Display, specify the custom function to invoke when a user clicks on the grid column header. For this property to be available, Column Click Action must be set to Custom.

Connection Pooling (`conn_pool_size`)
: In Panther/WebSphere applications, specify the number of concurrent database connections.

Current Component System (`current_component_system`)
: A runtime-only property that instantiates the type of component system currently in use. Before creating any service components, set this property to `PV_SERVER_COM` for COM components or `PV_SERVER_EJB` for Enterprise JavaBeans deployed under WebSphere Application Server.

HTML Max Loop (`html_max_loop`)
: For HTML templates using condition processing, specify the number of loop iterations to perform before terminating the process. The default setting is 1000.

HTML Max Nest (`html_max_nest`)
: For HTML templates using condition processing, specify the number of nesting levels. Each `if`, `while`, or `include` constitutes one level. The default setting is 20.

In Server (`in_server`)
: A new value for Panther/WebSphere applications, `PV_SERVER_EJB`.

Max Bundles (`max_bundles`)
: A runtime-only application property specifying the number of JPL bundles available for send and receive commands. It defaults to ten bundles (including the unnamed bundle) if unspecified.

Provider URL (`provider_url`)
: For Panther/WebSphere applications, a runtime-only application property specifying the location of the WebSphere application server machine. If

SMPROVIDERURL is set in the environment, the property is initially set to this value.

Runtime License (`runtime_license`)
: For ActiveX controls which support runtime licensing, if the Runtime License property exists, the control will be created using the license.

Sort Order (`sort_order`)
: Under Format/Display, specify the sort order to be used when the widget is in an array or in a grid. If the widget is in a grid, the Column Click Action property must also be set to Sort.

Sort Order Function (`sort_order_func`)
: Under Format/Display, specify the custom function to be invoked when Sort Order is set to Custom. The function can be either a JPL procedure or prototyped C function.

Utilities

`formlib`
: New `-m` option for compacting the library.

`makeejb`
: Panther/WebSphere utility for generating the Java files for EJBs from the service components in an application library.

CHAPTER

2 Using the JAM Upgrade Utility

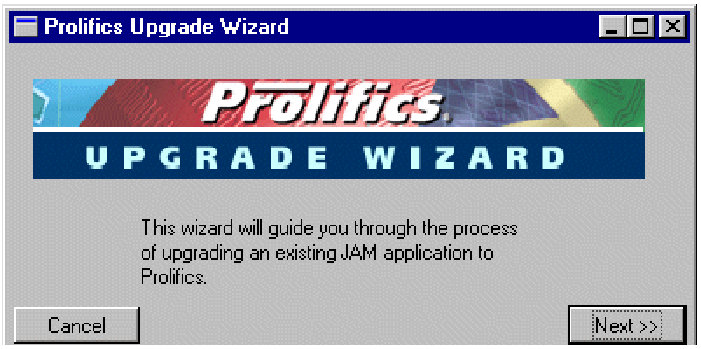
To order to help you upgrade a JAM application to a Panther application, the JAM to Panther utility bundles your loose JAM application files into libraries.

You can choose to update an existing JAM library to be your Panther application library or you can create a new application library. After the utility is complete, you can have the library open automatically by setting the `SMFLIBS` environment variable.

Running JAM to Panther

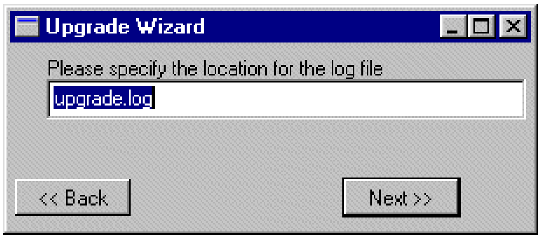
To run JAM to Panther:

- Windows: On the Start Menu, choose Panther→JAM Upgrade Utility.
- UNIX: On the command line, type `j2p`.



This wizard will guide you through the process of upgrading an existing JAM application to Prolifics.

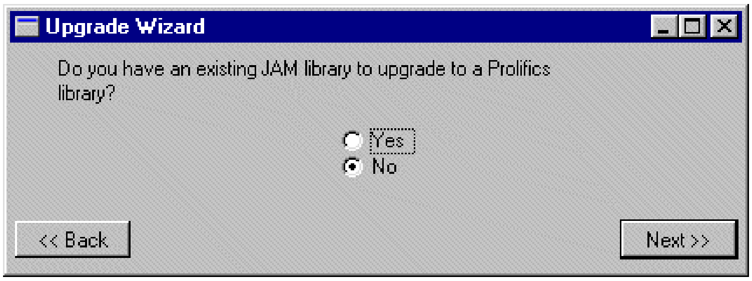
- Choose Next.



Please specify the location for the log file

upgrade.log

- Specify the path for the log file, and then choose Next. If unspecified, it defaults to `$SMBASE/util` for Windows and the current directory for UNIX.

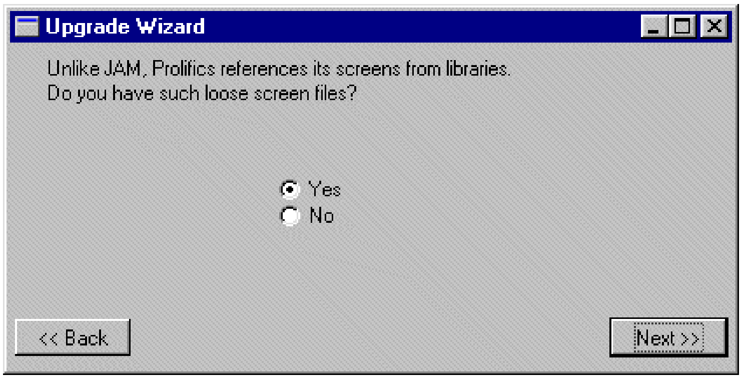


Do you have an existing JAM library to upgrade to a Prolifics library?

Yes [✓ No]

- If you have an existing JAM library, choose Yes. In the next screen, enter the library's location.

- If you did create a library for your JAM application, choose No.

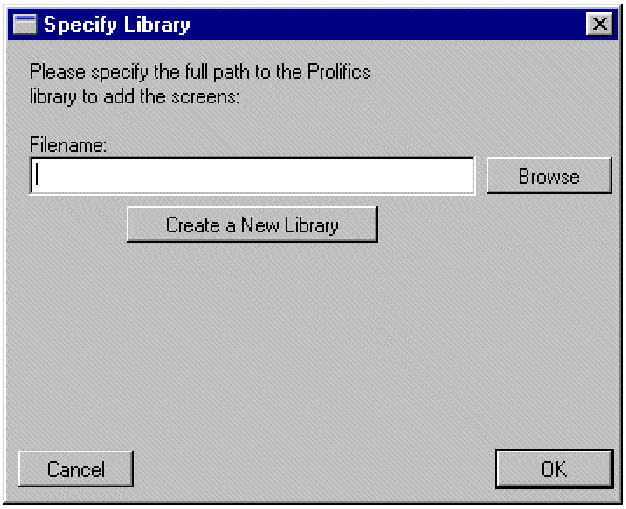


Upgrade Wizard

Unlike JAM, Prolifics references its screens from libraries.
Do you have such loose screen files?

[✓ Yes] No

- If you have screens not contained in a library, choose Yes and then Next.

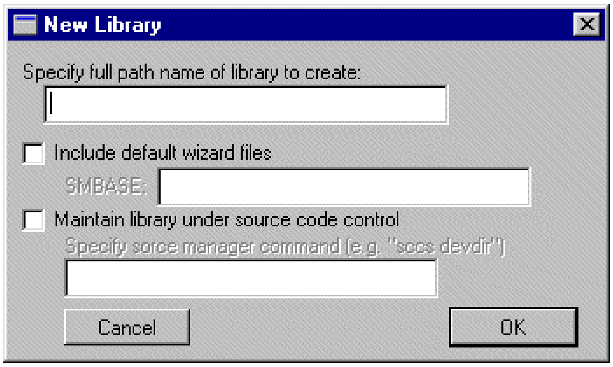


Specify Library

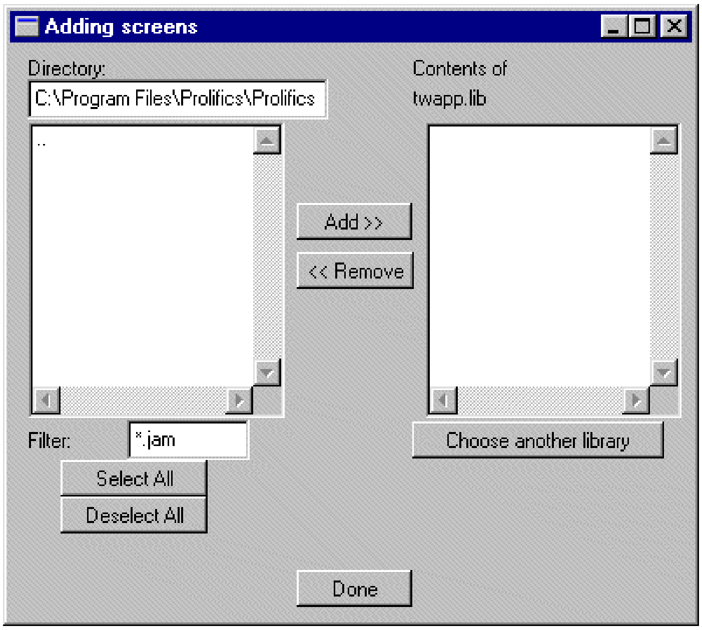
Please specify the full path to the Prolifics library to add the screens:

Filename:

- If you have not already specified a library, you can select a Panther library or create a new library. After entering the information, choose OK.

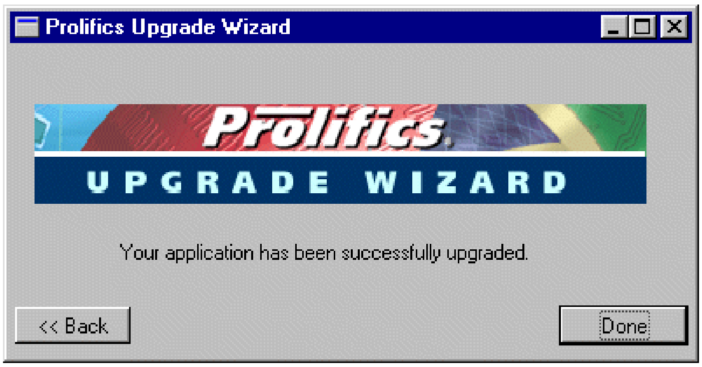

New Library
Specify full path name of library to create:
Include default wizard files
SMBASE:
Maintain library under source code control
Specify sorce manager command (e.g. "sccs devdir")
Cancel
OK

- Enter the path name of the new Panther library. If you want to include wizard files or maintain the library under source code control, specify the additional parameters. Choose OK.



- Select the directory containing the screens. If your screens do not end with a `.jam` extension, change the filter as needed.

- When the screens are displayed on the left, select the screens and choose Add. If you need to add additional screens from another library, select Choose Another Library.
- Choose Done when all screens have been added.
- The JAM Upgrade Utility then prompts you to add the following types of files to your library:
  - JPL files (filter is set to `.jpl`)
  - Reports (filter is set to `.jrw`)
  - Menu files (filter is set to `.mnu`)
  - Miscellaneous files, such as GIFs and JPGs for graphics.
- A repository can also be upgraded to the Panther format.
- After all steps are complete, the final screen displays:



Your application has been successfully upgraded.

- Choose Done.

CHAPTER

3 Upgrading to Oracle Tuxedo from JetNet

When you upgrade from the JetNet middleware adapter to Oracle Tuxedo, do not install the Oracle Tuxedo version over the JetNet version. The Oracle Tuxedo product should be installed in a separate directory.

Once the Oracle Tuxedo product is installed, you need to make the following changes in your application directory:

- Change the `SMBASE` setting to the Panther for Oracle Tuxedo installation. For UNIX application servers, `SMBASE` is set in `machine.env` and in `setup.sh`. For Windows application servers, it is set in `machine.env` and in the Windows System Environment.
- Because the following variables use `SMBASE` in their settings, their settings will also be updated:
  - `PATH`
  - `LD_LIBRARY_PATH, SHLIB_PATH, or LIBPATH`
- Update the server executables.
  - For UNIX, remove the existing links to the server executables and create new ones to the executables in the Oracle Tuxedo installation.
  - For Windows, remove the existing copies of the server executables and copy the ones from the Oracle Tuxedo installation.

CHAPTER

4 Upgrading to Panther from JAM 5

This chapter only discusses issues that pertain to upgrading screens in JAM 5 applications to Panther.

To further upgrade your application to Panther, refer to Chapter 1, “Upgrading to Panther from JAM 7.”

To upgrade your reports to ReportWriter 7, refer to page 1-52.

Upgrading From JAM 5

Although Panther differs considerably in its appearance from JAM 5, in its underlying functionality it remains closely related to JAM 5. The most obvious change is that JAM's interface has moved from a character to a graphical orientation. This is most evident in the graphical editor and in the more object-oriented terminology used throughout the product. This document discusses how to upgrade your applications from JAM 5 to Panther.

Warning: Be sure to back up your JAM 5 application before you start to upgrade it.

Upgrade Paths

There are three potential paths to follow when upgrading a JAM 5 application:

- Migration
- Utility conversion
- Full upgrade

These three paths are listed in ascending order of difficulty and effectiveness. Each path encompasses the previous one, but takes it several steps further.

Migration

Migration is the fastest and most direct way to get up and running with Panther. Migration means that your application remains virtually the same and runs with JAM 5 compatibility in the Panther environment. Once migrated, you can immediately begin to use the features of Panther to extend and modify your application. Although this compatibility will eventually be phased out of future releases, using it may buy you the time you need to plan a proper conversion. Experience has proven that migration works best for character-based applications, since graphical applications present more challenging issues. If you used JAM/P*i* to build your application, then migration is not recommended.

Utility Conversion

Conversion is potentially a more complicated process than migration. Its goal is to map your application's functionality into Panther constructs without altering it too much. A suite of conversion utilities, including a screen conversion utility, `\f5upg`, are provided to perform the bulk of this mapping. Once again, character applications convert more easily than graphical applications, but the utility does use a series of heuristics in an attempt to map JAM/P*i* constructs into Panther. The reason that graphical applications take more work to convert is that JAM/P*i* has undergone an extensive reworking as it was folded into Panther. The appearance of your graphical application after conversion will most likely be different than it was before, so you should examine each screen carefully and manually adjust them as necessary.

Full Upgrade

Finally, you might want to fully upgrade your application to Panther. Here, your application must be extensively modified to take full advantage of the advanced features provided in Panther. At this time, fully upgrading an application is a manual process.

Which Path is Best for My Application?

The best way to decide which path to follow is to read through the rest of this chapter, keeping in mind the specifics of your application and the upgrade options available. We first cover the steps that are common to all upgrade paths, and then branch into some of the more advanced upgrade steps.

Upgrading the Operating Environment

First of all, there are several steps common to each of the methods. These steps must be performed for all upgrades.

Update Your Configuration Files

Panther is distributed with new configuration files. If you modified any of these files in previous releases, you must reevaluate and transfer the changes into the new environment. You can edit the ASCII version of these files directly.

- keyfile
- message file

- setup or SMVARS file
- video file

Once you have made your changes, be sure to convert the file to binary using the appropriate Panther utility:

- key2bin
- msg2bin
- var2bin
- vid2bin

Table 4-1 shows the setup variables from JAM 4 that are no longer supported in Panther. References to these variables should be removed from your setup and configuration files.

Table 4-1 Obsolete setup variables

SMCHEMSGATT	SMCHFORMATTS	SMCHQMSGATT
SMCHSTEXTATT	SMCHUMSGAT	SMDWOPTIONS
SMEROPTIONS	SMFCASE	SMFEXTENSION
SMINDSET	SMMPOPTIONS	SMMPSTRING
SMOKOPTIONS	SMUSEEXT	SMZMOPTIONS

Be sure that your environment is pointing to the correct Panther configuration. This usually involves changing your SMVARS environment variable.

Update Your GUI Resource and Initialization Files

The Motif XJam file and the Windows ini file should be updated just like the configuration files. Any changes that you made to these files should be reevaluated and transferred to the Panther versions.

Color Aliases

Support for color aliases has now moved from the GUI resource and initialization files to the JAM configuration map file, `clrcmap`. Any color aliases that you added should therefore be moved to the configuration map file. Remember to use the utility `cmap2bin` to convert this file to binary format.

Update Your Data Dictionary into a Repository and LDB

While the format of most configuration files has not changed, an important exception is the Data Dictionary. In fact, both its role and its name have changed. Panther uses this file as a development tool; a repository of reusable widget definitions, stored in JAM library format. It is no longer used to create the runtime Local Data Block (LDB). You must use the `dd5upg` utility to convert your Data Dictionary to a repository. If your application relies on the LDB, use `dd5upg -l` to create a Panther library (`ldb.lib`) which is automatically loaded into the LDB at runtime. For details on using `dd5upg`, refer to "The dd5upg Utility" on page 4-21.

LDB Initialization

Panther no longer supports LDB initialization through ini files. We do however provide sample code that you can use to mimic JAM 5 behavior. You must choose whether to convert over to the new Panther conventions or simply link in the sample code.

An LDB in Panther is simply a library of screens. Panther performs initialization by either setting the initial text of the LDB widgets or by explicitly placing values into them through JPL statements. The sample code we provide, `sm5ldb.c`, enables ini files to work as they did in JAM 5, providing near seamless compatibility. Similar support is available for LDB scopes, Form structures, Data Dictionary records, and the old menu bar API.

Update Your Main Routines

Any changes that you made to the main routines in JAM 5, jmain.c and jxmain.c, should be reevaluated and transferred to the main routines provided with Panther. One way to locate these changes is to compare the distributed JAM 5 main routines with the JAM 5 main routines that you have been using. Then carry these changes forward to the Panther main routines, if appropriate.

Update Your Function List

Any functions that you added to the function list file funclist.c must be added to the new Panther funclist.c. You can use the same structure for declaring functions that you used in JAM 5.

Automatic Dereferencing

In looking at the function list in Panther, you'll notice that it includes prototypes for most of the JAM library functions. These are declared using a macro called `SM_INTFNC`. If you prototyped any of these functions for use in JAM 5, then you'll want to change these declarations to use the macro `SM_OLDFNC` instead.

The difference between the two macros is in their use of a new flag that has been added to the `intrn_use` member of the installation structure. This flag, `DEREF_ARGS`, allows function arguments to be automatically dereferenced without the need for colon expansion. If you were not in the habit of enclosing arguments in quotes when calling prototyped library functions in JAM 5, then you'll want to declare these library functions without the `DEREF_ARGS` flag. Declaring them with the `SM_OLDFNC` macro accomplishes this. `SM_OLDFNC`, `SM_INTFNC`, and `DEREF_ARGS` are all defined in the include file `sminstfn.h`.

Eliminate the Use of Release 4 Library Functions

Functions that were supported in JAM 5 for compatibility with JAM 4 are no longer supported. All usage of JAM 4 functions must be updated to use Panther functions. Table 4-2 lists the obsolete JAM 4 functions and their Panther equivalents.

Table 4-2 Obsolete JAM 4 functions and their Panther counterparts

JAM 4 Function	Panther Function
sm_ch_emsgatt	sm_option
sm_ch_form_atts	sm_option
sm_ch_qmsgatt	sm_option
sm_ch_stextatt	sm_option
sm_choice	sm_input
sm_cl_everyfield	sm_cl_unprot
sm_dw_options	sm_option
sm_er_options	sm_option
sm_fcase	sm_option
sm_fextension	sm_pset
sm_inbusiness	sm_inquire
sm_menu_proc	sm_input
sm_mp_options	sm_option
sm_mp_string	sm_option
sm_ok_options	sm_option
sm_openkeybd	sm_input
sm_plcall	sm_jplcall
sm_sdate	sm_sdtime
sm_stime	sm_sdtime
sm_smsetup	sm_option
sm_unsetup	sm_option
sm_zm_options	sm_option

Converting an Application

Converting a JAM 5 application to Panther is an iterative process that should be mapped out carefully.

The Conversion Toolkit

A suite of utilities are provided to aid in converting the various pieces of your application. These are summarized in Table 4-3.

Table 4-3 The upgrade utilities

Utility	Purpose
`dd5upg`	Converts JAM 5 data dictionaries to Panther repositories and, optionally, Local Data Blocks.
`f5upg`	Converts JAM 5 screens to Panther format.
`m2asc`	Converts JAM 5 menu bars to Panther format.
`dd2rec`	Converts records in a JAM 5 data dictionary to a format for use with sm5strct.c functions (see Table 4-4).

Sample source code is also provided that maps some JAM 5 features and function calls to Panther equivalents at runtime. These are summarized in Table 4-4. Detailed documentation for this code is provided in the source files themselves.

Table 4-4 Source code for backward compatibility

Function	Purpose
`sm5init.c`	Sets Panther library options for JAM 5 compatibility.
`sm5ldb.c`	Support for JAM 5 LDB scope and initialization functions.

Table 4-4 Source code for backward compatibility *(Continued)*

Function	Purpose
`sm5strct.c`	Support for JAM 5 bulk load/unload functions.
`sm5mbar.c`	Support for JAM 5 menu bar API.
`f2struct.c`	Converts screens to C structures for use with sm5strct.c functions.

The design and features in your particular application dictate which utilities and source modules are required in your conversion.

When a Feature is Missing...

As you work to establish a configuration and executable for your application you will undoubtedly notice that some features of JAM 5 are no longer supported in Panther. When this happens you must chose among converting to the new Panther conventions, finding a new approach, or linking in sample code we've provided to mimic the JAM 5 behavior.

As mentioned before, LDB initialization is a prime example of this. Panther supports this directly through the initial text property of an LDB widget. However, an alternative, and perhaps more flexible, approach is to explicitly place values into the widgets through JPL or C statements. Finally you can continue to use the ini files from JAM 5 by linking with the sample code we provide (`sm5ldb.c`) and calling `sm5_ldb_init` from your main routine. The choice is yours. Different applications and schedules require different choices.

Screens and Related Topics

Once you've built a Panther environment for your application, the next step is getting your screens to operate within it. Here the differences between migration and conversion become apparent.

All JAM 5 screens will execute under Panther without conversion. Panther recognizes that the screen was created in JAM 5 and translates fields, attributes and edits into widgets and properties. Widgets of this type are called “Release 5” widgets. You can expect these widgets for the most part to look and act as they did under JAM 5 control in both character and GUI environments. This is what we call migration, since with some subtle differences, your application remains the same.

The fact is, not everything translates automatically to Panther equivalents, and the appearance of certain fields/groups/messages will change under Panther. Since Panther is much more graphically oriented, you'll notice that many messages now appear in dialog boxes rather than on the status line. In addition, the border surrounding most windows looks slightly different by default. The rules governing the tabbing order between fields and groups have also changed slightly. Most of these differences can be resolved by either editing the screen or setting one or more of the option variables through your `SMVARS` file or source code provided in `sm5init.c`.

Users of JAM/P*i* will immediately note that none of the extensions to JPL are interpreted by default. Users of JAM/Reportwriter will find likewise. JAM/P*i* extensions and Reportwriter scripts have been fully integrated into widget properties for Panther. They can only be activated by passing the screen through the `f5upg` conversion utility. The utility supports a -5 option which performs these and other minor translations on the screen, avoiding a full conversion of the screen (`f5upg` can be used to continue the conversion at a later time).

Perhaps the most noticeable difference for GUI users is in widget positioning. The positioning algorithm in Panther places widgets at approximately the same positions they held under JAM/P*i*. For both migration and conversion, you will need to closely examine your screens under Panther and, more than likely, touch up the positioning of fields, boxes and lines. In most cases character users are spared this step.

Biting the Bullet

Most users will find that a full conversion to Panther is desirable. The f5upg utility defaults to performing a conversion and has many command line options available to help you in this process. Not all options are required for all screens and, indeed, some options are not desirable at all. We suggest that you look at all the available options and determine a set that most closely represents a conversion path for most of your screens. The remainder will need to be handled on an individual basis. The utility is based on heuristics controlled by command line options, so you may want to assess each screen and try different options.

Unless the -5 switch was used, screens that come out of f5upg consist entirely of Panther widgets. These may have different appearance and behavior than they had in JAM 5. The `f5upg` utility establishes each widget's type based on the rules established for JAM/P*i*. Plan on examining each screen to confirm that the conversion operated as intended and to resolve ambiguities that the utility was unable to handle.

Running Your Application for the First Time

For the most part, you can expect your application to run correctly. With minor exceptions, JAM events, JPL commands and library functions all perform as you remember. There are, however, several issues you may encounter.

As mentioned previously, there is a new function prototyping feature called field dereferencing that may cause your code to malfunction. Using this feature, JAM treats any string not surrounded by quotes as a field name, and attempts to extract a value from it which it then passes to the function being called. If you do not routinely place quotes around individual function arguments you will either need to correct your code or alter all the function prototypes in functlist.c to turn off the dereferencing feature. This can be accomplished by declaring the functions with the `SM_OLDFUNC` macro instead of `INTFNC`.

Users of JAM/P*i* should recall that widget types routinely changed from label to text and visible to invisible depending on protection settings and content. This behavior is no longer supported by Panther. The easiest way around this change in behavior is to settle on a single type for each widget for the duration of the screen. In lieu of this you'll have to rework your code to manage the display in two overlapping widgets of different types. In addition, you must to explicitly set the hidden property of widgets to get them to disappear, otherwise they merely deactivate.

Another potential issue deals with JAM 5 menus. Menu control strings are now a property of the menu fields themselves and, as a result, the control fields no longer have special meaning. The `f5upg` utility optionally migrates the content of the control fields to this new property, but if your code routinely reconstructs these fields, it will have to be changed to work with the properties instead. Please note that the old behavior continues to be supported by Release 5 widgets.

The Bottom Line

Plan on a full regression test of your application. Examine the behavior between and within screens. Undoubtedly you will like some new features and want to incorporate them right away; others you will want to avoid until later. Expect character applications to convert easier than GUI applications. Expect a majority of the code to convert easily, and some to be more difficult. Each application is different and it is important to treat them as such.

The f5upg Utility

Panther has an entirely new look and feel to its interface and its screens. This is a natural consequence of the changing computer software market and the rapidly expanding requirements for GUI technology and, even in character mode, applications that are as GUI-like as possible. In order to accommodate this new demand, we have added a significant amount of functionality to JAM and a new conceptual paradigm about JAM objects and their relationships to one another. In many cases, the old JAM 5 world will map with relative ease into the new Panther framework, but in order to bring the product into stricter compliance with GUI standards, there were some unique challenges that may make your upgrade from JAM 5 to Panther a tad more tricky than the JAM upgrades you have performed in the past.

The utility `f5upg` is designed purely for the purpose of converting JAM 5 screens to Panther screen format. Although Panther can read and process JAM 5 screens, certain JAM 5 features will be obsolete in future versions of the product, so it is strongly recommended that f5upg be used to convert all your JAM 5 screens to Panther.

If you are currently using JAM with a version earlier than 5, you must use the various utilities (`f3to5`, `f4to5`, etc.) to convert your screen binaries to JAM 5 format before you can use f5upg. If you use those utilities now or if you have used them in the past, you will notice that there is a rather large difference between those utilities and the f5upg utility.

The reason for this difference is that conversion of JAM screens in earlier versions of the product was a deterministic operation. In other words, there were no conceptual shifts in how JAM dealt with objects from version to version, so the conversion utilities merely had to map one binary representation of a concept into another. For the conversion from JAM 5 to Panther, the process is not strictly deterministic, but must use various heuristics specified by the user as command line options to effect the desired conversions. It is possible that you will need to use different combinations of command line options for different screens in the application you are converting, although in all likelihood most of your screens will be converted the same way.

Although it is possible with `f5upg` to specify the same file name for input and output and consequently overwrite your JAM 5 input screen binary with a Panther screen binary of the same name, we would strongly recommend that you not use this approach. Keep your JAM 5 screens available for the entire time you are upgrading your application until you are completely satisfied with the conversion. It might be that on looking at a particular converted screen, you could get something more along the lines of what you want by using a different combination of options to control the conversion heuristics.

Even if you did not use it, you should be aware of the product JAM/P*i*, or Presentation Interface, which was a layered product in the JAM 5 world and is now fully integrated with JAM in the Panther world. JAM/P*i* allowed JAM 5 users to create JAM applications in GUI environments. Much of the complexity of converting JAM 5 screens to Panther arises from the need to convert the JPL extensions used in JAM 5 to specify GUI-specific field and screen attributes to the Panther environment. If you did not use JAM/P*i* in JAM 5, much of the remaining part of this document may appear more complicated than necessary.

Invoking f5upg

The utility should be invoked from the command line with the following syntax:

```
f5upg [-5bcdfklmp] [-g type] [-v num] [-s color] JAM5_screen
JAM7_screen
```

Arguments and Options

`JAM5_screen`
: The name of the JAM 5 screen that you wish to convert.

`JAM7_screen`
The name that you would like the converted screen to have.

```
-5     Leaves fields as release 5 widgets by default.
-b     Forces border unless noborder is specified.
-c     Converts onscreen control fields to properties.
-d     Deletes onscreen control fields.
-f     Allows output file to overwrite existing file.
-k     Keeps JPL extensions on the output screen.
-l     Converts menu arrays to list boxes by default.
-m     Converts keyset designations to menu designations.
-p     Includes the GUI interface values for the hmargin, vmargin,
hbuffer, vbuffer properties in the converted screens.
-s     Assumes system colors (or scheme) by default.
```

`-g typeSets graphic conversion type:`

`N = none (default),`

`S = simple.`

`-v num Sets verbosity level. The default is 1.`

`-s colorDetermines color conversion method for screens and widgets:`

`X - For Motif applications, maintain JAM/Pi color scheme.`

`W - For Windows applications, maintain the JAM/Pi color scheme.`

`R - Reset colors according to Scheme in the Panther cmap file.`

If the option is not used, Windows' applications will convert the colors correctly only if color priority is undefined or not set to system. Colors for Motif applications may differ if the -sX option is not used.

General Behavior

`f5upg` assumes that the file name arguments are full path names, and treats them as such. There is no attempt to find screens along any particular path. First the utility ensures that the arguments are correct and consistent, then it attempts to find and open the input file as a JAM 5 screen. It then goes through the screen and convert JAM 5 screen and field objects and attributes to Panther screen and widget properties, according to some deterministic rules and a set of heuristics which are controlled by the command line options detailed below. Finally, the output file is written as a Panther screen.

Depending on the verbosity level set, informational status messages are printed to standard output and error messages are printed to standard error, and may be re-directed to log files by the operating system as desired.

When no command line options are set, all JAM 5 display text is converted to Panther Static Label widgets.

Any fields which are in JAM 5 groups are converted as follows:

- If they do not have JAM 5 check boxes assigned to them, they are converted to toggle button widgets.
- If they have JAM 5 check boxes but are JAM 5 radio buttons, they are converted to radio button widgets.
- If they have JAM 5 check boxes and are JAM 5 check boxes, they are converted to check box widgets.

Any fields unconverted so far are checked for the menu bit.

- If the menu bit is set, the fields are converted to Panther push buttons.

Any fields still unconverted are checked to see if they are protected from both data entry and tabbing:

- If not, they are converted to text widgets.
- If so, they are checked for protection against clearing, and if that is found they are made into dynamic labels.
- Otherwise they are converted to text widgets. This last set of fields, (those which are protected from both data entry and tabbing into) are hidden if the BLANK attribute is set, regardless of whether they were converted into text widgets or dynamic labels.

All JAM/P*i* extensions found in screen JPL modify properties of the screen as specified. All JAM/P*i* extensions found in field JPL modify the properties of the resulting widget as specified, including, possibly, the widget type.

Verbosity Level

The verbosity level is set with the -v command line option. The -v option is followed by a number ranging from zero to three. When 0 is specified, only critical errors are displayed to standard error. Verbosity levels 1, 2, and 3 all provide increasing levels of informational messages about the progress of the conversion, the heuristics being employed, and the use of features which may be rendered obsolete in future versions of JAM. These messages generally scroll to the terminal very fast, so if you need them it is recommended that output be directed to a log file.

By default, the verbosity level is set to 1.

Graphics Conversion

This is only of interest to JAM/P*i* users. In that product, boxes and lines could be specified as attributes of a given screen in that screen's JPL with the box, `hline`, and `vline` extensions. In Panther, boxes and lines are not attributes of screens, but screen objects in their own right. The `-g` option controls how these JAM/P*i* extensions are converted.

Specifying `-gN` merely throws away those graphic objects. Depending on the verbosity level, when `f5upg` encounters the JAM/P*i* extension for a box or line in a screen JPL, a message is printed to standard output warning the user that the extension could not be converted. This is the default.

Specifying `-gS` attempts a simple conversion of those graphic objects. In JAM/P*i*, boxes and lines were drawn between grid units on the screen for positioning purposes. In Panther, since these are objects in their own right, they take up grid units. As a result, this simple conversion will likely require subsequent manual manipulation in the editor to properly position the graphic element in question.

Keep JPL Extensions Around

By default, `f5upg` removes any JAM/P*i* extensions found in screen or field JPL before writing out the output screen. If the `-k` command line flag is specified, the utility leaves those extensions there. This does not harm the JPL, since the extensions are treated as comments by the JPL parser, but they really have no use. You should keep them around only if they serve as documentation to you. Most likely, they will just be a source of confusion.

The JPL extensions kept around are modified so that they are not reinterpreted if the target screen is run through `f5upg` a second time. They are modified so that the first angle bracket in the lead-in to each extension becomes an exclamation point. For example, `#<<nominimize>> <<font(myfont)>>becomes#!<nominimize>> !<font(myfont)>>`

Allow Output File to Overwrite an Existing File

By default, if the output file specified already exists, f5upg aborts without performing the conversion in question. However, if the -f option is specified, the output file is overwritten if it exists. Note that this means it is possible, in principle, to specify the same path for the input and output files. This is not recommended, however, because you will have lost your JAM 5 source file.

Assuming System Colors or Scheme by Default

f5upg provides a color option to specify how screen and widget colors will be converted. Depending on your use of the -s option, you can direct color conversion of your screen and widget colors in any of four different ways. The arguments to the color option work as follows:

W

Use this option to preserve widget and screen colors for Windows JAM/P*i* applications if color priority was set to system in the JAM.INI file.

X

Use this argument if you want your Motif application to maintain the same color scheme established in JAM/P*i*.

R

Convert all colors of screens and widgets according to the Scheme in the configuration map file. s option not specified For Windows applications, if color priority is not set to system—or is not defined—the colors will be unchanged. For Motif applications, the colors may differ if the color option is not set.

Conversion of Menu Arrays to List Boxes

By default, as mentioned above, JAM 5 menu fields are converted to push buttons. This is not necessarily optimal if your application had menu arrays, in particular scrolling menu arrays. If you specify the `-l` command line flag, all menu arrays are converted to list boxes rather than push buttons. Any single non-scrolling menu fields (not part of an array) are still converted to push buttons, even if `-l` is specified. (A non-scrolling list box with one element doesn't make sense).

Please be warned that the way JAM menu fields work means that their conversion to list boxes will likely cause them to operate in a way that is slightly different from most GUI standards. This is why the `-l` option is not the default behavior.

Note also that if there are JAM/P*i* extensions specifying widget type, this option will be overridden for that field.

Conversion of Borders

In JAM 5, screens were created without borders by default. JAM/P*i* screens, on the other hand, had borders unless the JAM/P*i* extension `noborder` was set in the screen JPL. JAM/P*i* users might, therefore, have created JAM screens without ever giving them borders and enjoyed the experience of the screens with borders nonetheless.

By default `f5upg` only designates a border for the Panther screen if there was a border specified in the JAM 5 character mode instantiation of the screen. However, if you set the `-b` command line flag, `f5upg` gives all screens a border even if the character mode JAM 5 screen did not have one. Only those screens marked as `noborder` will, in fact, not have a border.

Conversion of Onscreen Control Fields

Under JAM 5, the typical configuration of a menu involved a field (or array) with the menu bit set followed immediately on the screen by a corresponding field (or array) which held a control string. These control fields, usually hidden from view, were fields in their own right, and had field numbers, and could be accessed via the `sm_getfield` and `sm_putfield`. This was not the only way menu fields could be used, however.

Menu fields could have return values or submenus set, or the application might have managed the screen outside of the control of the JAM executive, such that a control field was not necessary.

In Panther, onscreen control fields are still supported, but they are on their way to obsolescence. Control strings under Panther are typically designated as properties of the push button or list box widgets in question, not as separate fields on the screen.

By default, when f5upg encounters a JAM 5 field with the menu bit set, it leaves it in place and generates a level 1 warning to the user stating that a menu was found and that an onscreen control field may have been used, and that this feature will soon be obsolete. Two command line flags can alter this behavior: the flags `-c` and `-d`. `-c` cause `f5upg` to copy control strings from onscreen control fields to the properties of the actual push button or list box widget derived from the initial menu field or array, and `-d` causes `f5upg` to delete the onscreen control fields from the screen.

Please note that there is no deterministic way for `f5upg` to know whether a given field did or did not have an associated control field in JAM 5. It depends on the application using that screen, which f5upg has no access to. `f5upg` can make certain good guesses which it does as follows:

- If the menu field or array is the last (or only) field on the screen, there can be no associated control field.
- If the menu field or array has a submenu attribute specified in JAM 5, `f5upg` assumes there is no corresponding control field.
- If the menu field or array has a return value attribute specified in JAM 5, `f5upg` assumes there is no corresponding control field.
- In all other cases, `f5upg` assumes that the field or array immediately following the menu field or array is an onscreen control field. These fields are either flagged as obsolete, copied to control string properties, and/or deleted depending on the presence or absence of the `-c` and/or `-d` command line flags.

Reasons not to use the `-c` command line flag:

- Your application uses C language or JPL routines to set the value of a control field. (If your application sets the value of the actual menu field, there is no problem.)
- Your application processes the screen in question (possibly every screen) without using the JAM executive. In other words, the screen is processed directly by using `sm_input`.

- The screen you are converting has (for obscure reasons) different dimensionality between the menu fields and their associated control fields. For example, there is a single array for a set of menu description fields, but each control field is a single field.

Reasons not to use the `-d` command line flag:

- All reasons not to use `-c` apply also to `-d`.
- You probably do not want to use -d independently of `-c` because you will lose information.
- Your application uses C language or JPL routines to inspect the value of a control field. (If your application inspects the value of the actual menu field, there is not a problem.)
- Your application makes use of field numbers to set the values of or otherwise access fields on the screen. All fields following the deleted control field would be re-numbered on the converted screen, leading to erroneous field processing.

Conversion of Keyset Designations

In JAM/P*i*, the designation of a keyset was reinterpreted as designating a menu script. Since `f5upg` has no way of knowing whether a given screen was designed for use under JAM/P*i* or not, use the `-m` option to move the designation of a keyset to the menu script property.

Protected Field Heuristics

`f5upg` converts any data entry fields that are protected from both data entry and tabbing into to Dynamic Label Widgets. In addition, if the field is marked as non-display, it is hidden. This heuristic is performed by default because it is assumed that the vast majority of screens would be best converted in this way.

Release 5 Widgets

`f5upg` attempts to convert every widget it can according to the heuristics discussed above under "General Behavior" on page 4-14. If, however, the -5 option is specified, the widget is left unconverted as a JAM release 5 field on a Panther screen. Panther attempts to deal with such fields to achieve a stricter backwards compatibility, but these fields are much harder to manipulate in the Panther editor, and are more difficult to convert to future releases of the product. You may wish to do the conversion with the `-5` option and later do a full conversion. Please note that the presence of a JAM/P*i* extension specifying widget type overrides this option, and the field is converted nonetheless.

The dd5upg Utility

The `dd5upg` utility converts JAM 5 data dictionaries into JAM/Panther repositories. The utility should be invoked from the command line with the following syntax:

```
dd5upg [-cflv] [-p prefix] [-e ext] JAM5dd repository
```

Arguments and Options

`JAM5dd`
: The name of the JAM 5 data dictionary.

`repository`
: The name of the JAM/Panther repository.

`-c`
: Create compact JAM/Panther screens.

`-f`
: Output file may overwrite an existing file.

`-l`
: Generate a screen library, ldb.lib, to be used as the LDB.

`-v`

Prints the name of each screen as it is generated.

`-p<`*`str`*`>`

Uses the specified prefix for generated JAM/Panther screens. The default is `R5dd`.

`-e<`*`str`*`>`

Uses the specified extension for generated JAM/Panther screens. The default is `SMFEXTENSION`.

Description

By default, the utility looks for the file `data.dic` in the current directory. That file is read and a file named `data6.dic`, a Panther library, is output. This library is intended to be used as a Panther repository.

Each field in the original data dictionary has a scope associated with it. Each scope produces one or more screens in the new library that contain the entries of that scope. By default, the screens are named:

```
R5dd
<scope>
<sequence#>
```

An output screen is populated with as many fields of a given scope as possible. If there are more fields of a scope than will fit on one screen, a new screen is started with the next sequence number tacked onto the name. Sequence numbers start at 0.

For example, if the input data dictionary has five entries of scope 1 and three entries of scope 2, the output library will contain the following screens:

```
R5dd10.jam  (has 5 fields)
R5dd20.jam  (has 3 fields)
```

If the `-l` option is used, `dd5upg` also creates the library `ldb.lib` to be used as the local data block. This library contains the same screens as `data6.dic`, except that they are named `ldb1.jam`, `ldb2.jam`, etc.

The m2asc Utility

The `m2asc` utility converts Panther menus between binary and ASCII formats. It also upgrades JAM 5 menus to Panther menus. To upgrade menus, the utility should be invoked from the command line with the following syntax:

```
m2asc -c [-fv] JAM5_menu [JAM5_menu ...]
```

Arguments and Options

`JAM5_menu`
: The name of the JAM 5 binary menu file to convert.

`-c`
: Converts JAM 5 binary file to Panther binary file.

`-f`
: Output file may overwrite an existing file.

`-v`
: Generates a list of the files as they are processed.

Description

When run with the `-c` option, this utility converts JAM 5 menu bar binaries to Panther menu bar binaries.

The dd2rec Utility

The `dd2rec` utility converts JAM 5 data dictionary records to programming language data structures. The utility should be invoked from the command line with the following syntax:

```
dd2rec [-flpv] [-o output_file | -o-] [-g language] dictionary
```

Arguments and Options

`dictionary`
: The name of the JAM 5 data dictionary to convert.

`-f`
: Overwrites an existing output file.

`-l`
: Converts record names to lower case.

`-p`
: Creates the output files in the same directory as the data dictionary.

`-v`
: Generates a list of the records as they are written.

`-o` *`output_file`*
: The name of the output file.

`-o-`
: Outputs to the screen.

`-g` *`language`*
: Generates records as an array of structures in the language specified. Currently only C is supported.

Description

The data structures generated by `dd2rec` can be used for backward compatibility via the `sm5strct.c` functions listed in on page 4-8 in Table 4-4.

CHAPTER

5 Conversion Summary from JAM 5 to Panther

The following tables summarize the issues that you may face in converting your applications. They are divided into three categories:

- All Applications (page 5-2)
- GUI Applications (page 5-10)
- Character Applications (page 5-13)

All Applications

Table 5-1 Configuration

Configuration Area	Notes
Data dictionaries	Run `dd5upg` to create a Prolifics data dictionary (`data6.dic`) and, optionally, an LDB (`ldb.lib`). Set `SMLDBNAME` = `new name` if you have not named it `ldb.lib`.
Groups	Groups are not loaded as part of repositories and LDBs. Hence, groups must be reestablished via the screen editor.
Key files	New keys have been added.
LDB Initialization Files	Perform one of these actions: ■ Compile and link `sm5ldb.c` to the application and call `sm5_ldb_init` in main. ■ Move `SMININAMES` from the setup file to either environment or `sm_ininames` call. ■ Edit the file by changing: “field” “value” to field=“value”; include this file in your startup JPL code. ■ Remove these fields from the data dictionary and make them global JPL variables.
Menu bar files	Run `m2asc -c` to convert to Panther format.
Message file	Fold in changes made to JAM 5 message file.
`SMVARS` and `SMSETUP`	Fold in changes made to JAM 5 files. `DBI4COMPATIBLE` is no longer supported.
Video file	While the format of these files remains the same, the content of the Panther files might be different, resulting in different appearance (e.g. graphic characters).

Table 5-1 Configuration *(Continued)*

Configuration Area	Notes
screen libraries	Extract all screens from the library. Run `f5upg` on each screen. Reconstruct the library.
Italics indicate options that are supported for backward compatibility only.	

Table 5-2 Executable

Area	Notes
Block mode	Not available in this release.
Memory-resident screens	Run `f5to6` on the original screen. Run `bin2c`. Re construct the header file.
`funclist.c`	Fold in changes made to your JAM 5 `funclist.c`. If your application does not routinely enclose each function argument in quotes, turn off the `DEREF_ARGS` flag in `SM_INTFNC`, `SM_STRFNC` and `SMDBLFNC` macros (see `SM_OLDFNC`) to turn off dereferencing of field names on function call lines.
Makefile	Use the supplied makefile, folding in changes made to your JAM 5 makefile. Compare your JAM 5 file with the distributed JAM5 file to discover your changes.
Main routine	Use the supplied `jmain.c`, folding in changes made to your JAM 5 jmain.c. Compare your JAM 5 file with the distributed JAM5 file to discover your changes.

Table 5-3 Library Functions

Function	Notes
`sm_bkrect`	Obsolete, but still supported.
`sm_blkinit`	Block mode not supported.

Table 5-3 Library Functions *(Continued)*

Function	Notes
`sm_blkreset`	Block mode not supported.
`sm_i_bitop`	Fails on array fields.
`sm_c_keyset`	Soft keys not supported.
`sm_dicname`	Link your application with `sm5ldb.c`.
`sm_do_region`	Obsolete, but still supported.
`sm_edit_ptr`	Use `sm_prop_get`.
`sm_emsg`	Use `sm_femsg`.
`sm_err_reset`	Use `sm_ferr_reset`.
`sm_getjctrl`	Use `sm_prop_get`.
`sm_ininames`	Link your application with `sm5ldb.c`.
`sm_initcrt`	Use `sm_jinitcrt`.
`sm_jplload`	Use `sm_jplpublic`.
`sm_keyset`	Soft keys not supported, use menu bars instead.
`sm_kscscope`	Soft keys not supported, use menu bars instead.
`sm_ksinq`	Soft keys not supported, use menu bars instead.
sm_kson	Soft keys not supported, use menu bars instead.
`sm_ksoff`	Soft keys not supported, use menu bars instead.
`sm_lclear`	Link your application with `sm5ldb.c`.
`sm_ldb_hash`	Link your application with `sm5ldb.c`.
`sm_ldb_init`	Link your application with `sm5ldb.c`.
`sm_ldbrevive`	Link your application with `sm5ldb.c`.
`sm_ldbsave`	Link your application with `sm5ldb.c`.

Table 5-3 Library Functions *(Continued)*

Function	Notes
sm_lreset	Link your application with sm5ldb.c.
sm_r_menu	Link your application with sm5mbar.c.
sm_d_menu	Link your application with sm5mbar.c.
sm_c_menu	Link your application with sm5mbar.c.
sm_mnadd	Link your application with sm5mbar.c.
sm_mnchange	Link your application with sm5mbar.c.
sm_mndelete	Link your application with sm5mbar.c.
sm_mnget	Link your application with sm5mbar.c.
sm_mninsert	Link your application with sm5mbar.c.
sm_mnitems	Link your application with sm5mbar.c.
sm_mnnew	Link your application with sm5mbar.c.
sm_mnutogl and jm_mnutogl	Menu and input modes are no longer relevant. Push buttons (formerly menus) always execute an action when activated, and can be tabbed into from other fields. To restore JAM 5 behavior, set AUTO_TOGGLE = AT_DISABLE in the setup file or call sm_option.
sm_perm_emph	No longer supported.
sm_putjctrl	Use sm_prop_set.
sm_qui_msg	Use sm_fqui_msg.
sm_rd_part	Link your application with sm5strct.c.
sm_rdstruct	Link your application with sm5strct.c.
sm_rrecord	Link your application with sm5strct.c.
sm_skinq	Soft keys not supported, use menu bars instead.
sm_skmark	Soft keys not supported, use menu bars instead.

Table 5-3 Library Functions *(Continued)*

Function	Notes
`sm_skset`	Soft keys not supported, use menu bars instead.
`sm_skvinq`	Soft keys not supported, use menu bars instead.
`sm_skvmark`	Soft keys not supported, use menu bars instead.
`sm_skvset`	Soft keys not supported, use menu bars instead.
`sm_submenu_close`	Use menu bars instead.
`sm_wrecord`	Link your application with sm5strct.c.
`sm_wrt_part`	Link your application with sm5strct.c.
`sm_wrtstruct`	Link your application with sm5strct.c.
Italics indicate options that are supported for backward compatibility only.	

Table 5-4 JPL

Area	Notes
Comments	The : character is no longer supported for comments.

Keywords	Panther introduces the following new keywords:
	`double`
	`global`
	`string`
	`send`
	`receive`
	`runreport`
	`tpi`
	For ease of recognition, avoid using these as program variables.

Loop variables	These are no longer floating point numbers (e.g. JAM 5 = 1.00; JAM 6 = 1).

Table 5-4 JPL *(Continued)*

Area	Notes
unload	An error is now generated when unload is issued for a file which was not previously loaded. Use call `sm_jplunload`... to restore JAM 5 behavior.
`sql`	Obsolete. Use `dbms query` and `dbms run` instead.

Table 5-5 Screens

Feature	Notes
Circular arrays	Arrowing down past the last occurrence places the cursor in the first element and displays the first occurrence there. Arrowing up past the first occurrence places the cursor in the last element and displays the last occurrence there. JAM 5 did not reposition the cursor.
Cursor up/down	Panther positions the cursor at the end of existing data by default when entering a widget. To restore JAM 5 behavior, set `IN_RESET` = `OK_NORESET` in the set up file or call `sm_option`.
Field function flags	`K_KEYS` always set to `K_NORMAL` on `K_ENTRY`.
Group navigation	In Panther, TAB and `BACK` position the cursor to the next field by default. To restore JAM 5 behavior, call `sm_keyoption` and set the `TAB` and `BACK` key bits to `VF_GROUP` | `VF_CHANGE`.
Group function flags	The MDT bit is not set when a member is selected. `K_KEYS` is always set to `K_NORMAL` on `K_ENTRY`.
Menus with control fields	Control fields are replaced by the control string property. There are two possible conversions via f5to6. The first is to use `f5to6 -5` to retain control field functionality via the Release 5 widget type. The second is to use `f5to6 -c` to set the Control String property from the control field data, provided that its number of occurrences matches that of the menu field. Code that modifies the control fields must be changed to call property API functions.
Menus with return codes	To push back a key, call `jm_keys` from the control string property.

Table 5-5 Screens *(Continued)*

Feature	Notes
Multiple range edits	New widgets may only have one range. Use the validation function for complex range evaluation.
Submenus	Use menu bars instead.
Tabbing order	Panther uses the `alt_next_tab_stop` property only when the field specified in its `next_tab_stop` property does not exist. If the next field is protected, then its designated next field is used. To restore JAM 5 behavior, set `JAM5_COMPATIBLE` = `JAM5_ON` in the setup file or call `sm_option`.
Scrolling date/time fields	All occurrences in a scrolling array with a system date/time edit will fill with data once they are allocated. In JAM 5, only the first element was filled.
Soft keys	Soft keys are not supported. `f5to6 -m` copies the keyset name into the menu bar property.
Zoom	If you map the NL key to `TAB`, you will not be able to add new lines at the end of an array. Use insert key instead. Remap NL to NL before invoking zoom.
Italics indicate options that are supported for backward compatibility only.	

Table 5-6 Processing

Area	Notes
`AUTO` key	In Panther, this event occurs after entering first field. In JAM 5 it occurred before.
Custom executive	If menu return codes are used, call `jm_keys` and `sm_keyoption` to change key routing.
Key remapping	Mouse clicks generate `NL` and `DELE` keys which will be remapped as if they had been entered from the keyboard. Use a default field function to reset remappings on entry and restore them on exit from menu fields.

Table 5-6 Processing *(Continued)*

Area	Notes
Menu vs. input mode	Menu and input modes are no longer relevant. Push buttons (formerly menus) always execute an action when activated, and can be tabbed into from other fields. To restore JAM 5 behavior, set `AUTO_TOGGLE = AT_DISABLE` in the setup file or call `sm_option`.
Screen entry/exit hooks	Activation on expose/hide is now the default. To re store JAM 5 behavior, set `EXPHIDE_OPTION` = `OFF_EXPHIDE` in the setup file or call `sm_option`.
Status line hook	Does not get called for messages now displayed via `sm_message_box`. Use `ERROR_FUNC` to trap all messages excluding `setbkstat`, `d_msg` and query.This event takes place before translation of % escapes.
Italics indicate options that are supported for backward compatibility only.	

Table 5-7 Menu Bar

Item	Notes
empty items	No longer supported. Panther displays the parent button when a child menu has no recognizable contents.
title	No longer supported. It will not appear on menus.

Table 5-8 Utilities

Utility	Notes
`dd2struct`	No longer supported. Use the `d2rec` utility to generate record definitions.
`f2asc`	Text format changed.
`f2struct`	No longer supported. Use `f2struct` supplied in sample code.
`f2tbl`	Not available in this release.

Table 5-8 Utilities *(Continued)*

Utility	Notes
`lstform`	No longer supported.
`menu2bin`	Replaced by `m2asc`. Text format changed.
`modkey`	Replaced by `showkey` utility. Since key file formats remain the same, the JAM 5 utility may still be used for those keys that aren't new.
`term2vid`	Not available in this release.
`text2form`	Not available in this release. Use the JAM 5 utility and load the generated screens directly into Panther or pass them through `f5to6` for conversion.
Italics indicate options that are supported for backward compatibility only.	

GUI Applications

Table 5-9 Configuration

Area	Notes
Environment variables	In Motif, `XNLSPATH` may be required for some implementations.
Color aliases	Color aliases currently defined in resource files must be moved to the JAM configuration map file. To mimic JAM 5 behavior, set the Color Type property for all widgets and screens to Scheme and either re move [Scheme] settings from the configuration map or set all BG and FG scheme colors to GUI equivalents (e.g. XJam.Background).

Table 5-10 Screen, general

Feature	Notes
Automatic widget type changes	The JAM/P*i* widget conversion algorithm formerly determined widget types based on protection. In Panther, widget types do not change at runtime. Therefore, changes in protection do not change widget type.

Blank push buttons (menu fields)	Blank push button widgets are no longer invisible. If you want widgets to disappear and reappear, use the following code:
	In JPL:
	`field-spec->hidden = PV_YES`
	`field-spec->hidden = PV_NO`
	In C:
	`id = sm_prop_id (field-spec);`
	`sm_set_prop (id, PR_HIDDEN, PV_YES);`
	`sm_set_prop (id, PR_HIDDEN, PV_NO);`
	A more GUI-like behavior is to make the button in active by toggling its active property.

Button label text	In Motif, the label is centered based on resource settings. In Windows, the label is always centered.

Table 5-11 Screen, Pi extensions

Feature	Notes
colorpriority	Sets the Color Type property of Foreground and Background to Scheme.
frame, box	Margin no longer supported.
hline, vline	May need to be repositioned.
hoff, voff	Not converted.
iconify	Iconifies a screen even if there is no icon specified.
list	Default horizontal and vertical scroll bars translated to no scroll bars.

Table 5-11 Screen, Pi extensions *(Continued)*

Feature	Notes
multiline	These are converted to push buttons, and the additional text is truncated.
multitext	Default horizontal and vertical scroll bars translated to no scroll bars.
noadj	Colon expansion is no longer supported.
nomove	No longer supported.
space	Colon expansion is no longer supported.
Colon expansion	Colon expansion of variables might not occur at the same time as in JAM/P*i*,
Widget position	The Panther positioning algorithm may change widget positions. Inspect your converted screens and make appropriate adjustments.

Table 5-12 Processing

Area	Notes
Changing field colors	Calls to `sm_chg_attr` that set field colors to match the screen colors (e.g. unhighlighted black) should be changed to set the colors to the expected color or to the container color (via the `B_INHERITED` attribute mask). JAM no longer automatically propagates screen colors to fields when the attributes match.

Table 5-13 Library functions

Function	Notes
`sm_widget`	In Motif, use `sm_xm_widget`. In Windows, use `sm_mw_widget`. Also, the include files required by these functions changed to `smmwuser.h` and `smxmuser.h`.

Table 5-13 Library functions *(Continued)*

Function	Notes
`sm_attach_drawing_func`	In Motif: `sm_xm_attach_drawing_func`.In Windows: `sm_mw_attach_drawing_func` Note that a 3rd argument for a user data buffer has been added to these functions. Also, the include files required by these functions changed to `smmwuser.h` and `smxmuser.h`.
`sm_drawingarea`	In Motif, use `sm_xm_drawingarea`. In Windows, use `sm_mw_drawingarea`. Also, the include files required by these functions changed to `smmwuser.h` and `smxmuser.h`.

Character Applications

Table 5-14 Screen

Feature	Note
Clear on input	Field is selected if data is present.

Table 5-14 Screen *(Continued)*

Feature	Note
Groups	The appearance of JAM 5 check boxes on Panther radio button and check box widgets has changed from _ to either ( ) or [ ]. The `f5to6` utility at tempts to reposition the group so that brackets appear around the old check box position. Because these extra characters are now considered part of the original field width, the initial text might be truncated by the conversion process, requiring you to manually reset the values. `f5to6` cannot automatically resize the widgets since it might cause unintentional overlapping of widgets or push widgets off the screen. Use `f5to6` -5 to retain the old appearance through a Release 5 widget type.
Menu Fields	These are now push buttons so they are not set to reverse video when the cursor enters. Use `f5to6 -5` to retain old behavior through the Release 5 widget type.

APPENDIX

A JAM Documentation: Alternative Scrolling

By default, storage of scrolling arrays is handled internally by Panther, which stores them in its own memory buffers. It is also possible for this data to be stored by the application, external to Panther–for example, in memory or disk. In this case, the application must install a scrolling driver which is called by Panther with an interface defined by Panther. Installation of a scrolling driver replaces Panther's default scroll driver. The driver is called to initialize the array, get and put occurrences, and so on.

You can write your own scrolling function; an alternative scroll driver can reduce application memory usage when used to control the scrolling of large arrays. Scroll drivers can be freely mixed on a screen. Each driver can be specified to manage any number of arrays and any number of drivers can be used at once.

Panther Interaction with Scrolling Drivers

When Panther initializes an application, it calls all scrolling drivers, and installs them. Similarly, when the application exits, Panther calls the driver for clean up. While the driver is active, Panther calls the routine once for each new scrolling array that Panther creates–for example, on screen open–telling it to initialize the array.

While the array is active, Panther moves data back and forth between itself and the driver and informs the driver of other changes to the array–for example, insertion or deletion of occurrences.

When the array is destroyed–for example, its window closes–the driver is called to release all the data associated with the array.

Normally, a non-scrolling widget can hold as many occurrences of data as the display allows. In character mode, this corresponds to the number of lines the widget occupies onscreen. The data held in non-scrolling widgets is kept as part of the normal screen data structure. However, by setting the Scrolling property to Yes, you can enter a number of occurrences that is equal to or greater than the number of visible occurrences.

Because the amount of data kept in scrolling widgets can grow large, the data for offscreen occurrences is managed separately from onscreen. Management of offscreen data is handled either by Panther's default scroll driver, or by a custom-written driver, which you can specify in the Alt Scroll Func property. If this property is left blank or the name is invalid, Panther uses its own scroll driver.

Installation

You can bundle multiple scrolling drivers into a Panther application. Scrolling drivers are installed in `funclist.c` in `sm_do_uinstalls`. Two types installations are important:

- A default scrolling driver, used when an array does not specify which driver to use or the specified driver is not installed.
- Installation of alternate drivers that are available in the executable.

The definition and installation of the default driver in `funclist.c` looks like this:

```
static struct fnc_data udfunc[] =
{
   SM_OLDFNC("virtmem", sm_vmbscroll),
};

static int udcount =
       sizeof (udfunc) / sizeof (struct fnc_data);
sm_install (DFLT_SCROLL_FUNC, udfunc, &udcount);
```

The definition and installation of the list of drivers in `funclist.c` looks like:

```
static struct fnc_data ufuncs[] =
{
   SM_OLDFNC("virtmem", sm_vmbscroll),
   SM_OLDFNC("dosmem", sm_mbscroll),
   SM_OLDFNC("dummy", adummy),
};
```

```
static int ucount =
            sizeof (ufuncs) / sizeof (struct fnc_data);
sm_install (SCROLL_FUNC, ufuncs, &ucount);
```

If no default scrolling driver is installed, Panther uses its own scroll driver.

Scroll Driver Interface

All scroll drivers have a single entry point. Panther passes the driver two parameters. The first is a pointer to an `altsc_t` structure; the second parameter is an operation code, indicating what action Panther needs the scrolling driver to perform. Following is an example definition of a scrolling driver:

```
int
scroll_driver (as_ptr, action_code)
altsc_t *as_ptr;
int  action_code;
```

The altsc_t Structure

A pointer to an `altsc_t` structure is the first parameter to a scroll driver. Table A-1 describes the `altsc_t` structure.

Table A-1 The altsc_t structure

Member name	Description	Type
Driver-maintained information:		
`scrolldata`	Pointer to structure maintained by driver.	`VOIDPTR`
Scrolling information:*		
`luid`	ID of the largest-used occurrence	`unsigned int`
`max_items`	Maximum number of occurrence	`unsigned int`
`shs`	Largest non-blank occurrence	`unsigned int`
Occurrence information:		
`item`	Occurrence number	`unsigned int`

Table A-1 The altsc_t structure *(Continued)*

Member name	Description	Type
`len`	Length of the occurrence	`unsigned char`
`attr`	Occurrence attributes	`attribute`
`valids[AS_VAL]`	Occurrence validation bits	`unsigned char`
`*text`	Occurrence data	`unsigned char`
Other parameters:		
`number`	Integer parameter	`int`
`vptr`	Pointer parameter–currently not used.	`VOIDPTR`

**This information is supplied on all calls except for INIT and RESET.*

The `altsc_t` structure serves two purposes:

- Lets the scrolling driver save information about an array between calls.
- Acts as a vehicle for passing data between the driver and Panther.

A scrolling driver can manage several arrays at once. The `scrolldata` member of the `altsc_t` structure serves the purpose of keeping tract of data for each array. When the driver is called to `AS_INIT_FUNC`, the driver should store a pointer in the `scrolldata` member, usually to an internal structure. Panther assumes that the driver uses the pointer in `scrolldata` to access the offscreen data. The `scrolldata` pointer can only be changed during the `INIT` call; all other calls pass back the same pointer saved from the INIT call.

The `altsc_t` structure also passes data into and out of the driver. The scrolling information parameters–`luid`, `max_items`, and `shs`–are passed so the driver knows the size of the array, current and potential. However, the `luid` and `shs` parameters are only completely accurate for the `AS_INSRT_FUNC` and `AS_DEL_FUNC` calls; otherwise, they are approximate. The occurrence information parameters–`item`, `len`, `attr`, `valids` and `text`–are used to pass information about specific occurrences into and out of the driver. The other parameters –`number`, `vptr`–are used only for specific calls.These members are discussed later in the description of each action.

Return Values

The scrolling driver generally returns 0 if the function is supported and it succeeds, -1 if the function is not supported, and non-zero value if the function fails. Exceptions are noted in the action code descriptions.

Scroll Driver Action Codes

The interface to the scrolling driver is through the mechanism of various action codes. Each action code represents one particular functionality. Many actions are required to update members of the `altsc_t` structure that is passed to the driver. Although the scrolling driver has one entry point, almost all drivers are implement ed as a giant switch statement that calls other routines.

For an example, refer to page A-9, "Scrolling Driver Example."

The following sections describe the calling protocols and functionality expected for the routines associated with the action codes.

`AS_CLEAR_FUNC`

Not currently used. Function should return -1.

`AS_DLT_FUNC, AS_INSRT_FUNC`

`AS_DLT_FUNC` and `AS_INSRT_FUNC` expect routines to delete and insert array occurrences, respectively. The routines typically manipulate indexes or lists to implement the deletion/insertion.New entries that are inserted into the array, or trailing occurrences that are left blank when occurrences are deleted, should have their attr, valids, and text parameters set from the values that are passed in the structure. A `NULL` pointer for text indicates an empty string.

The return value is the number of lines actually inserted/deleted; should equal number.

Input parameters (and their descriptions) are:

`scrolldata`	Pointer to the driver's internal buffer.
`item`	Occurrence number to start at–the first deleted occurrence or first newly inserted occurrence.
`number`	Number of occurrences to delete or insert.
`len`	Length of the occurrence.

`attr`	Display attribute to give to new occurrences.
`valids`	Validation bits to give to new occurrences.
`text`	Text to put into new occurrences.

`AS_GDATA_FUNC`

`AS_GDATA_FUNC` expects a routine that gets the data of an occurrence in a scrolling array. The routine might be called because the occurrence scrolled onscreen, or in response to a request for data from a C or JPL routine, for example, a call by `sm_getfield`.

Occurrence data contains text, display attributes, and validation flags. Panther allocates a buffer to hold this data and passes a pointer to the buffer in the text member. The routine should update the attr, text, and valids members in the structure with values previously passed in by the `AS_PDATA_FUNC` routine, although a clever driver can manufacture them.

Return value of 0 indicates success; -1 indicates failure.

Note: An empty occurrence must be blank-filled.

Input parameters (and their descriptions) are:

`scrolldata`	Pointer to the driver's internal buffer.
`item`	Occurrence number to get.
`len`	Length of the occurrence. Represents the size of Panther's buffer; the scrolling driver should not overrun this length.
`text`	Buffer to get the text.

Output parameters (and their descriptions) are:

`attr`	Display attribute of the occurrence.
`text`	The routine should fill the buffer pointed to by text with the text of the occurrence as passed to the routine by `PDATA`.

`valids`	Occurrence's validation bits; these are packed into a unique form.

`AS_GTSPC_FUNC`

`AS_GTSPC_FUNC` expects a routine that tells the driver the largest occurrence, or luid, that it must keep track of. Any buffers or resources currently allocated for keeping track of data above number occurrences can be freed.

The return value is the number of occurrences it has resources allocated for. Usually, the return is the same as number.

Input parameters (and their descriptions) are:

`scrolldata`	Pointer to the driver's internal buffer.
`number`	New `luid` to use.

`AS_INIT_FUNC`

`AS_INIT_FUNC` expects a routine that is called when Panther starts handling scrolling data for an array. The `max_items` and len parameters provide the driver with approximately how much data the array can hold.

Return value of 0 indicates success; -1 indicates failure.

Input parameters (and their descriptions) are:

`max_items`	Maximum number of occurrences in the array. If the value of `max_items` subsequently changes, Panther calls the `AS_GTSPC_FUNC` function to inform the driver of new limits.
`len`	Maximum width of the text of each occurrence.

Output parameter is `scrolldata`. Set to point to a driver-specific data structure that keeps track of the data for the array. This value is associated with the array; all future calls to the scrolling driver that refer to this array get the `scrolldata` value passed in the `altsc_t` structure.

`AS_INST_FUNC`

`AS_INST_FUNC` expects a routine that is called once when Panther starts up. If an application causes Panther to initialize and reset more than once, the function is called repeatedly. No buffer is passed to the driver; instead, it gets `NULL`.

This routine usually initializes the driver's global properties, allocates buffers, initializes virtual memory, opens files. However, it cannot perform tasks, especially if the initialization is performed for each array by `AS_INIT_FUNC`.

Return value of 1 indicates success; 0 indicates failure–driver is not installed.

`AS_NMUSD_FUNC`
Not currently used. Function should return -1.

`AS_PDATA_FUNC`

`AS_PDATA_FUNC` expects a routine that is called when Panther wants to update offscreen data in the driver. The routine should save the contents of the attr, valids, and text members.

Return value of 0 indicates success; -1 indicates failure.

Note: The routine cannot assume that text is null-terminated; it should save all len bytes of it.

Input parameters (and their descriptions) are:

`scrolldata`	Pointer to the driver's internal buffer.
`item`	Occurrence number to save.
`len`	Length of the occurrence.
`text`	Buffer to get the text.
`attr`	Display attribute of the occurrence.
`valids`	Validation bits of the occurrence.
`text`	Text to save.

`AS_RESET_FUNC`
`AS_RESET_FUNC` expects a routine to call when Panther terminates with `sm_resetcrt`. The routine can perform any desired final clean up, such as deleting temporary files, freeing memory allocated in the `INST` function, and so on. The routine has no input or output parameters and returns no value. It

should only be called by Panther when it has released all active arrays with `AS_RLS_FUNC`.

`AS_RLS_FUNC`

`AS_RLS_FUNC` expects a routine that is called when Panther destroys an array. The routine should then free any resources allocated for the array. The return value from this call is ignored. Thus, the scrolling driver must inform the user about any failure and take the appropriate action–for example, exit the program.

The input parameter is `scrolldata`, the pointer to the driver's internal buffer.

Return value of 0 indicates success; -1 indicates failure.

`AS_SCMAX_FUNC`

Not currently used. Function should return -1.

Scrolling Driver Example

The following code is an example of a scrolling driver:

```
/* Scroll buffers hold field data of scrolling. New buffers
are allocated as formerly empty offscreen elements are filled.
Each buffer can hold several items, determined by
i_per_buff = (SC_BUF_SIZE - SC_OVERHEAD) / (item_size + SC_BYTES)
where item_size is the maximum (shifting) length of a field.
The extra SC_BYTES hold VALIDED, MDT, and attribute information
for a field scrolled offscreen. The constants are defined below */

/* A buffer is allocated contiguously with its sc_buf_s.*/

#define SC_BUF_SIZE 1024
#define SC_LEN 0  /* length of an item if var.*/
#define SC_ATTR 0 /* offset of saved attribute*/
#define SC_BITS 2 /* offset of saved bits*/
#define SC_BYTES SC_BITS+AS_VAL /* extra bytes per item*/
#define SC_DATA SC_BYTES /* offset of field data*/

/* bits stored are:
   SELECTED  (0x10)
   VALIDED   (0x20)
   MDT       (0x40)
   OPROTECT  (0x80)
   PROTECT   (0x0f) */
```

```
typedef
struct sc_buf_s

{
    struct sc_buf_s *link; /* forward link */
    struct sc_buf_s *back; /* backward link */
    unsigned int first_item; /* first in this buffer */
    unsigned int last_item; /* last in this buffer */
    char item[SC_BYTES+1]; /* start of buffer */

} sc_buf_t;

#define SC_OVERHEAD     (int)(sizeof (sc_buf_t) - SC_BYTES - 1)

/* Every scroll header has an item altsc_id which is a pointer to
a scroll_t. scroll_t has an item scrolldata. A structure of
scrolldata depends on a scrolling method. If memory-based scrolling
is used, the structure of scrolling data is: */

typedef
struct scr_data_s

{
    sc_buf_t *first_buff;
    sc_buf_t *cur_buff;
    unsigned int i_per_buff;
} scr_data_t;

/* To store offscreen data, buffers are acquired as needed. Buffers
are linked into a chain whose head is stored in the member
first_buff in the structure above. Each buffer holds the same number
of items, determined by the formula above. */

/* FUNCTIONS IN THIS MODULE */

int sm_mbscroll PROTO((altsc_t *, int));
static int SMLOCAL mb_initscr PROTO((altsc_t *));
static int SMLOCAL mb_getitem PROTO((altsc_t *));
static int SMLOCAL mb_putitem PROTO((altsc_t *));
static int SMLOCAL mb_insitem PROTO((altsc_t *));
static int SMLOCAL mb_delitem PROTO((altsc_t *));
static int SMLOCAL mb_setluid PROTO((altsc_t *));
static int SMLOCAL mb_rlsscrl PROTO((altsc_t *));
static char * SMLOCAL mb_getptr PROTO((altsc_t *));
static sc_buf_t * SMLOCAL mb_addscbuf PROTO((altsc_t *,
       sc_buf_t *));

static int SMLOCAL mb_incluid PROTO((altsc_t *, scr_data_t *,
       unsigned int));
```

```
static sc_buf_t * SMLOCAL mb_getscbuf PROTO((scr_data_t *,
       int));

static void SMLOCAL mb_free_sc_bufs PROTO((sc_buf_t *));

int sm_mbscroll (as_ptr, option) altsc_t *as_ptr;

int option;

{
   switch (option)
   {
   case AS_INIT_FUNC:
      return mb_initscr (as_ptr);

   case AS_GDATA_FUNC:
      return mb_getitem (as_ptr);

   case AS_PDATA_FUNC:
      return mb_putitem (as_ptr);

   case AS_INSRT_FUNC:
      return mb_insitem (as_ptr);

   case AS_DLT_FUNC:
      return mb_delitem (as_ptr);

   case AS_GTSPC_FUNC:
      return mb_setluid (as_ptr);

   case AS_RLS_FUNC:
      return mb_rlsscrl (as_ptr);
   default:
      return 1;
   }
}

static int SMLOCAL
mb_initscr (as_ptr)
altsc_t *as_ptr;
{
   scr_data_t *scr_data;
   int retcode;
   scr_data = (scr_data_t *) sm_fmalloc (sizeof /
        (scr_data_t));
   if (scr_data && as_ptr->len > 0)
   {
      scr_data->first_buff = 0;
      scr_data->cur_buff   = 0;
      scr_data->i_per_buff =
            (unsigned)(SC_BUF_SIZE - SC_OVERHEAD) /
            (as_ptr->len + SC_BYTES);
        retcode = 0;
    }

    else
    {
        sm_qui_msg ("Memory based scrolling method
            cannot be initialized");
        retcode = -1;
    }
    as_ptr->scrolldata = (VOIDPTR)scr_data;
    return retcode;
}

static int SMLOCAL
mb_getitem (as_ptr)
altsc_t *as_ptr;
{
    char *ptr;
    if (!(ptr = mb_getptr (as_ptr)))
    {
        return -1;
    }
    memcpy ((VOIDPTR)as_ptr->valids, (VOIDPTR)&ptr[SC_BITS],
          AS_VAL);
    as_ptr->attr = UGET (ptr, SC_ATTR);
    memcpy ((VOIDPTR)as_ptr->text, (VOIDPTR)&ptr[SC_DATA],
         as_ptr->len);
    return 0;
}

static int SMLOCAL
mb_putitem (as_ptr)
altsc_t *as_ptr;
{
    char *ptr;
    if (!(ptr = mb_getptr (as_ptr)))
    {
        return -1;
    }
    memcpy ((VOIDPTR)&ptr[SC_BITS], (VOIDPTR)as_ptr->valids,
         AS_VAL);
    UPUT (ptr, SC_ATTR, as_ptr->attr);
    memcpy ((VOIDPTR)&ptr[SC_DATA], (VOIDPTR)as_ptr->text,
         as_ptr->len);
    return 0;
}
```

```
static int SMLOCAL
mb_insitem (as_ptr)
altsc_t *as_ptr;
{
   unsigned char *ptr;
   unsigned char valids[AS_VAL];
   attrib_t attr;
   int item;
   int item_to;
   int item_from;
   item = (int)as_ptr->item;
   if ((unsigned int)as_ptr->number >
       (as_ptr->luid - as_ptr->shs))
   {
      as_ptr->number = (int)(as_ptr->luid - as_ptr->shs);
   }
   item_from = (int)as_ptr->shs;
   item_to = item_from + as_ptr->number;
   ptr = as_ptr->text;
   attr = as_ptr->attr;
   memcpy ((VOIDPTR)valids, (VOIDPTR)as_ptr->valids, AS_VAL);
   as_ptr->text = (unsigned char *)sm_fmalloc (as_ptr->len);
   if (!as_ptr->text)
   {
      return -3;
   }
   for (; item_to >= item; item_to--)
   {
      if (item_from >= item)
      {
         as_ptr->item = item_from;
         if (mb_getitem (as_ptr))
         {
            if (ptr)
            {
               sm_ffree (as_ptr->text);
               as_ptr->text = ptr;
            }
            return -1;
         }
         item_from--;
      }
      else if (ptr)
      {
         sm_ffree (as_ptr->text);
         as_ptr->text = ptr;
         as_ptr->attr = attr;
         memcpy((VOIDPTR)as_ptr->valids,(VOIDPTR)valids,
              AS_VAL);
            ptr = 0;
        }
        as_ptr->item = item_to;
        if (mb_putitem (as_ptr))
        {
            if (ptr)
            {
                sm_ffree (as_ptr->text);
                as_ptr->text = ptr;
            }
            return -2;
        }
    }
    if (ptr)
    {
        sm_ffree (as_ptr->text);
        as_ptr->text = ptr;
    }
    return as_ptr->number;
}
static int SMLOCAL
mb_delitem (as_ptr)
altsc_t *as_ptr;
{
    unsigned char *ptr;
    unsigned char valids[AS_VAL];
    attrib_t attr;
    unsigned int item_to;
    unsigned int item_from;
    item_to = as_ptr->item;
    if (as_ptr->number > (int)(as_ptr->luid - item_to + 1))
    {
        as_ptr->number = (int)(as_ptr->luid - item_to + 1);
    }
    item_from = item_to + (unsigned int)as_ptr->number;
    ptr = as_ptr->text;
    attr = as_ptr->attr;
    memcpy ((VOIDPTR)valids, (VOIDPTR)as_ptr->valids, AS_VAL);
    as_ptr->text = (unsigned char *)sm_fmalloc (as_ptr->len);
    if (!as_ptr->text)
    {
        return -3;
    }
    for (; item_to <= as_ptr->shs; item_to++)
    {
        if (item_from <= as_ptr->shs)
        {
            as_ptr->item = item_from;
            if (mb_getitem (as_ptr))
            {
                if (ptr)
                {
                    sm_ffree (as_ptr->text);
                    as_ptr->text = ptr;
                }
                return -1;
            }
            item_from++;
        }
        else if (ptr)
        {
            sm_ffree (as_ptr->text);
            as_ptr->text = ptr;
            as_ptr->attr = attr;
            memcpy((VOIDPTR)as_ptr->valids,(VOIDPTR)valids,
                  AS_VAL);
            ptr = 0;
        }
        as_ptr->item = item_to;
        if (mb_putitem (as_ptr))
        {
            if (ptr)
            {
                sm_ffree (as_ptr->text);
                as_ptr->text = ptr;
            }
            return -2;
        }
    }
    if (ptr)
    {
        sm_ffree (as_ptr->text);
        as_ptr->text = ptr;
    }
    return as_ptr->number;
}
static int SMLOCAL
mb_setluid (as_ptr)
altsc_t *as_ptr;
{
    sc_buf_t *sc_buf;
    scr_data_t *scroll;
    unsigned int new_luid;
    unsigned int old_fuid;
    scroll   = (scr_data_t *) as_ptr->scrolldata;
    if (!scroll)
        return 0;
    new_luid = as_ptr->number;
    if (!new_luid)
    {
        mb_free_sc_bufs(scroll->first_buff);
        scroll->first_buff = scroll->cur_buff = 0;
        return 0;
    }
    sc_buf = mb_getscbuf (scroll, (int)new_luid);
    if (!sc_buf)
        new_luid = mb_incluid(as_ptr, scroll, new_luid);
    else if (sc_buf->link)
    {
        mb_free_sc_bufs(sc_buf->link);
        sc_buf->link = 0;
        scroll->cur_buff = sc_buf;
    }
    old_fuid = as_ptr->luid;
    as_ptr->luid = new_luid;
    while (++old_fuid <= new_luid)
    {
        as_ptr->item = old_fuid;
        mb_putitem (as_ptr);
    }
    return (int)new_luid;
}

/*

NAME: mb_rlsscrl - Free scroll buffers

SYNOPSIS: mb_rlsscrl (as_ptr)
altsc_t *as_ptr;

DESCRIPTION: Loops through field's scroll buffers, freeing them.
Then frees altsc_id and sets it to 0.

*/

static int SMLOCAL
mb_rlsscrl (as_ptr)
altsc_t *as_ptr;
{
    sc_buf_t *sc_buf;
    scr_data_t *scroll;
    scroll = (scr_data_t *) as_ptr->scrolldata;
    sc_buf = scroll->first_buff;
    mb_free_sc_bufs(sc_buf);
    sm_ffree ((VOIDPTR) scroll);
    return 0;
}
```

```
/*

NAME: mb_getptr - Get a pointer to offscreen item in memory buffer

SYNOPSIS:

   ptr = mb_getptr (as_ptr);
   char *ptr;
   altsc_t *as_ptr;

DESCRIPTION: Searches forward or backward from current scroll
buffer to find buffer that contains specified occurrence. If the
occurrence previously existed, returns pointer to data.

RETURNS: ptr = pointer to offscreen data; 0 if occurrence could not
be found.

*/

static char * SMLOCAL
mb_getptr(as_ptr)
altsc_t *as_ptr;

{
   scr_data_t *scroll;
   unsigned int occur;
   sc_buf_t *sc_buf;
   if (!(scroll = (scr_data_t *)as_ptr->scrolldata) ||
      !(sc_buf = mb_getscbuf(scroll, (int)(occur =
         as_ptr->item))))
      return 0;
   return sc_buf->item + ((unsigned)occur -
        sc_buf->first_item) * (as_ptr->len + SC_BYTES);
}

/*

NAME: mb_addscbuf - Add another sc_buf_s to the list specified.

SYNOPSIS:

   new_scroll_buffer = mb_addscbuf (as_ptr, scroll_buffer);
   sc_buf_t *new_scroll_buffer;
   altsc_t *as_ptr;
   sc_buf_t *scroll_buffer;

DESCRIPTION: Mallocs and initializes new scroll buffer. If scroll
buffer passed is null, the first_buff and cur_buff entries of scroll
header are set to the new buffer.

RETURNS: new_scroll_buffer = pointer to new buffer; 0 if malloc
failed.
```

```
*/

static sc_buf_t * SMLOCAL
mb_addscbuf (as_ptr, sc_buf)
altsc_t *as_ptr;
sc_buf_t *sc_buf;

{
   unsigned int f_item;
   unsigned int l_item;
   unsigned int buf_size;
   sc_buf_t *new_buf;
   scr_data_t *scroll;
   scroll = (scr_data_t *) as_ptr->scrolldata;
   l_item = 0;
   if (sc_buf)
       l_item = sc_buf->last_item;
   f_item = l_item + 1;
   l_item += scroll->i_per_buff;
   if (l_item > as_ptr->max_items)
       l_item = as_ptr->max_items;
   buf_size = (l_item - f_item + 1) * (as_ptr->len +
      SC_BYTES) + SC_OVERHEAD;
   new_buf = (sc_buf_t *)sm_fmalloc(buf_size);
   if (new_buf == 0)
   {
      char   buf[80];
      sprintf (buf, "There is no more memory for new
         occurrence # %i", as_ptr->number);
      sm_qui_msg (buf);
      return 0;
   }
   memset((VOIDPTR)new_buf, 0, buf_size);
   new_buf->link = 0;
   new_buf->first_item = f_item;
   new_buf->last_item  = l_item;
   new_buf->back = sc_buf;
   if (sc_buf)
      sc_buf->link = new_buf;
   else
   {
      scroll->first_buff = new_buf;
      scroll->cur_buff   = new_buf;
   }
   return(new_buf);
}

/*
NAME: mb_getscbuf - Get pointer to scroll buf that contains the
```

```
specified item.

SYNOPSIS:
    scroll_buffer = mb_getscbuf (sc_data, item);
    sc_buf_t *scroll_buffer;
    scr_data_t *sc_data;
    int item;

DESCRIPTION: Searches scroll buffers for specified item. Uses the
cur_buff pointer to save time. Updates cur_buff pointer (to save
time next time). If desired item is in first buffer, don't bother
with cur_buff.

RETURNS: scroll_buffer = pointer to desired buffer; 0 if item is
not in any buffer.

*/

static sc_buf_t * SMLOCAL
mb_getscbuf(sc_data, item)
scr_data_t *sc_data;
int item;
{
    sc_buf_t *sc_buf;
    sc_buf_t *ret_buf;
    if (!(sc_buf = sc_data->cur_buff))
        return 0;
    if (item <= (int)sc_buf->last_item)
    {
        sc_buf_t *f_buf;
        if (item >= (int)sc_buf->first_item)
            return sc_buf;
        f_buf = sc_data->first_buff;
        if (item <= (int)f_buf->last_item)
        {
            if (item < (int)f_buf->first_item)
                return 0;
            return f_buf;
        }
        if (item >= (int)((sc_buf->first_item + \
            f_buf->last_item) / 2))
        {
            while ((ret_buf = sc_buf->back) &&
                item < (int)(sc_buf = ret_buf)->first_item)
            {
            }
            sc_data->cur_buff = sc_buf;
            return ret_buf;
        }
        sc_buf = f_buf;
    }
    while ((ret_buf = sc_buf->link) &&
        item > (int)(sc_buf = ret_buf)->last_item)
    {
    }
    sc_data->cur_buff = sc_buf;
    return ret_buf;
/*

NAME: mb_incluid - Ensures sufficient allocated space to hold
desired item.

SYNOPSIS:

    new_luid = mb_incluid (as_ptr, scroll, desired_new_luid)
    int new_luid;
    altsc_t *as_ptr;
    scr_data_t *scroll;
    unsigned int desired_new_luid;

DESCRIPTION: Calls mb_addscbuf in a loop to increase
largest_used_item_id until it meets or surpasses desired value.

RETURNS: new_luid = smaller of desired_new_luid and amount of space
which could be allocated.

*/

static int SMLOCAL
mb_incluid (as_ptr, scroll, new_luid)
altsc_t *as_ptr;
scr_data_t *scroll;
unsigned int new_luid;
{
    sc_buf_t *new_buf;
    sc_buf_t *last_buf;
    unsigned int retval;
    retval = new_luid;
    last_buf = scroll->cur_buff;
    do
    {
        if (!(new_buf = mb_addscbuf(as_ptr, last_buf)))
        {
            if (last_buf)
                retval = last_buf->last_item;
            else
                retval = 0;
            break;
        }
    }
    while (new_luid > (last_buf = new_buf)->last_item);
    scroll->cur_buff = last_buf;
    return (int)retval;
}
/*

NAME: mb_free_sc_bufs - Frees scrolling buffers starting with one
passed.

SYNOPSIS:

    mb_free_sc_bufs (scroll_buffer)
    sc_buf_t *scroll_buffer;

DESCRIPTION: Loops through on link freeing buffers. Caller is
responsible for valid arguments and setting pointers to the freed
buffer to NULL. Argument can be NULL.

RETURNS: None.

*/

static void SMLOCAL
mb_free_sc_bufs(sc_buf)
sc_buf_t *sc_buf;
{
    char *ptr;
    while (sc_buf)
    {
        ptr = (char *)sc_buf;
        sc_buf = sc_buf->link;
        sm_ffree(ptr);
    }
}
```

APPENDIX

B JAM Documentation: Internal I/O Processing

This chapter describes the following:

- How keyboard input is processed—including which library functions are called to carry out such processing.
- How Panther and Panther applications under character-mode use the information encoded in the video file to process output.

All user input to a Panther application is processed through a keyboard translation file or table before being handled by Panther. All output to a character-mode display monitor is processed through a video mapping table.

This translation of input and output is done to avoid code specific to particular displays or terminals, and thereby preserve terminal independence. Panther and Panther applications can run on a variety of terminals, provided that the appropriate keyboard and video configuration files are identified. These configuration files are used by the application at initialization to establish the keyboard and video translation.

Processing Keyboard Input

Keystrokes are processed in three steps:

1. The sequence of characters generated by one key is identified.
2. The sequence is translated to an internal value, or logical character.
3. The internal value is either acted upon or returned to the application (key routing).

All three steps, described in this section, are table-driven. Hooks are provided at several points for application processing; refer to Chapter 44, “Installed Event Functions,” in *Application Development Guide*.

Logical Keys

Panther processes characters internally as logical values, which usually correspond to the physical ASCII codes used by terminal keyboards and displays. Panther uses the key translation file to map specific physical keys or sequences of physical keys to logical values, and the video file's `MODE` and `GRAPH` entries to map logical characters to video output. For most keys, such as displayable data characters, no explicit mapping is necessary. Certain ranges of logical characters are interpreted specially by Panther:

`0x0100` to `0x01ff`	Operations such as tab, scrolling, cursor motion
`0x6101` to `0x7801`	Function keys `PF1` to `PF24`
`0x4101` to `0x5801`	Shifted function keys `SPF1` to `SPF24`
`0x4103` to `0x5a03`	ALT keys `ALTA` to `ALTZ`
`0x6102` to `0x7802`	Application keys `APP1` to `APP24`

Key Translation

The first two steps in Panther's processing of keyboard input–identification and translation–are controlled by the binary key translation file, loaded at initialization. Panther finds the file's name in a setup file or in the environment (refer to the *Configuration Guide* for details). The binary file is derived from an ASCII file that you can modify with any text editor.

Panther assumes that the first input character of a multi-character key sequence is a control character in the ASCII chart (`0x00-0x1f`, `0x7f`, `0x80-0x9f`, or `0xff`) and attempts to translate the character to a single logical key. Characters outside this range are assumed to be displayable characters and are not translated.

Note: This algorithm assumes that a timing interval (`KBD_DELAY` entry in the video file) has not been specified. For more information, refer to `KBD_DELAY` on page 7-24 in *Configuration Guide*.

On receiving a control character, the keyboard input function `sm_getkey` searches the key translation file for a sequence beginning with that character. If no match is found on the first character, Panther accepts the key without translation. If a match is found on the first character, an exact match, `sm_getkey` returns the indicated value. The search continues through subsequent characters until one of the following conditions is true:

- An exact match on n characters is found and the nth+1 character in the file is 0, or n is 6. In this case, the value in the file is returned.
- An exact match is found on n-1 characters but not on n. In this case, `sm_getkey` attempts to flush the sequence of characters returned by the key.

The latter condition is of some importance: if the user presses a function key that is not defined in the file, Panther must know where the key sequence ends. The following algorithm is then used:

- The file is searched for all entries that match the first n-1 characters and are of the same type in the nth character, where the types are digit, control character, letter, and punctuation. The smallest of the total lengths of these entries is assumed to be the length of the sequence produced by the key.
- If there is no entry matches by type at the nth character, the shortest sequence that matches on n-1 characters is used. Hence, `sm_getkey` can distinguish, for example, between the sequences `ESC O x, ESC [ A, and ESC [ 1 0 ~`.

With Timing Interval Set

If you have a KBD_DELAY entry in your video file, you can specify key sequences in the key translation file that are substrings of other sequences. For example, the sequences ESC and ESC [C can both have logical values, even though one is a substring of the other. In this case, Panther waits the specified timing interval, as indicated in the KBD_DELAY entry, between processing characters to determine if a character is a single keystroke or belongs to a sequence of keystrokes.

Key Routing

The third step in processing keyboard input is handled by the library function sm_input. This function calls sm_getkey to obtain the translated value of the key. It then decides what to do based on the following rules:

Value Greater Than 0x1ff

If the logical value is greater than 0x1ff, sm_input returns the value as the return code.

Value Between 0x01 and 0x1ff

If the value is between 0x01 and 0x1ff, the key is translated via the key translation file. The processing of the key is then determined by a routing table. You can alter the default behavior of keys (cursor control) within this range with the library function sm_keyoption as well as set the routing information for a particular key. The routing value consists of two bits, examined independently, so four different actions are possible:

- If neither bit is set, the key is ignored.
- If the EXECUTE bit is set and the logical value is in the range 0x01 to 0xff, it is written to the screen (as interpreted by the GRAPH entry in the video file, if one exists). If the value is in the range 0x100 to 0x1ff, the appropriate action (Tab, field erase, etc.) is taken.
- If the RETURN bit is set, sm_input returns the logical value to the caller; otherwise, sm_getkey is called for another value.
- If both bits are set, the key is executed and then returned.

The default settings are ignored for ASCII and extended ASCII control characters (`0x01` - `0x1f`, `0x7f`, `0x80` - `0x9f`, `0xff`), and `EXECUTE` only for all others. The default setting for displayable characters is `EXECUTE`. All other ASCII and extended ASCII characters are ignored. The function keys (`PF1` to `PF24`, `SPF1` to `SPF24`, `APP1` to `APP24`, and `ABORT`) are not handled through the routing table. Their routing is always `RETURN`, and cannot be altered. All other function keys (`EXIT`, `SPGU` etc.) are initially set to `EXECUTE`.

Changing Key Actions at Runtime

You can program your application to change key actions at runtime by using `sm_keyoption`. For example, to disable the Backtab key, you execute the following function:

```
call sm_keyoption(status, BACK, KEY_ROUTING, KEY_IGNORE giving
status)
```

To make the field erase key return to the application program, use:

```
call sm_keyoption(status, FERA, KEY_ROUTING, RETURN giving status)
```

Logical key mnemonics are defined in the `smkeys.h` file in the include directory.

Processing Terminal Output

Panther defines a set of logical screen operations (such as positioning the cursor) and stores the character sequences for performing these operations in a video file specific to the display. Logical display operations and the coding of sequences are described in Chapter 7, “Video File,” in *Configuration Guide*.

This section describes the ways in which Panther uses and ultimately, the way applications use the information encoded in the video files to determine how and what output your terminal displays.

How Panther Handles Output

Panther uses a delayed write output scheme to minimize unnecessary and redundant output to the display. No output at all is done until the display must be updated, either because keyboard input is solicited or the library function `sm_flush` is called. Instead, the runtime system does screen updates in memory and keeps track of the display positions. Flushing begins when the keyboard is opened; but if you type a character while flushing is in progress, the runtime system processes it before sending any more output to the display. Therefore, you can type ahead on slow displays. You can force the display to be updated by calling `sm_flush`.

Graphics Characters and Alternate Character Sets

Many terminals support the display of graphics or special characters through alternate character sets. Panther provides 8-bit alternate character sets–for example, those that translate from IBM PC extended character to Latin-1. These tables can be installed by calling the library function `sm_xlate_table`. Control sequences switch the terminal among the various sets, and characters in the standard ASCII range are displayed differently in different sets. Panther supports 8-bit to 7-bit translations via the `MODEx` and `GRAPH` entries in the video file.

The seven `MODEx` sequences (where x is 0 to 6) switch the terminal into a particular character set. `MODE0` must be the normal character set. The `GRAPH` command maps logical characters to the mode and physical character necessary to display them. It consists of a number of entries whose form is logical value = mode physical-character.

When Panther needs to output logical value it first transmits the sequence that switches to mode, then transmits physical-character. It keeps track of the current mode to avoid redundant mode switches when a string of characters in one mode (such as a graphics border) is being written. `MODE4` through `MODE6` switch the mode for a single character only.

APPENDIX

C Obsolete Functions

The functions in this section were made obsolete in either the Panther, JAM or Prolifics products.

Function Name	Made Obsolete In	Description
sm_achg	JAM 7	
sm_apply_prop	JAM 7	
sm_ascroll	JAM 7	
sm_base_fldno	JAM 7	
sm_bitop	JAM 7	
sm_chg_attr	JAM 7	
sm_com_call_method	Panther 4.2	Calls a method of a COM component (Recommended alternate: sm_obj_call)
sm_com_get_prop	Panther 4.2	Gets a COM component property setting (Recommended alternate: sm_obj_get_property)
sm_com_log	Panther 4.2	Writes a message to a log file (Recommended alternate: sm_log)
sm_com_onerror	Panther 4.2	Installs an error handler (Recommended alternate: sm_obj_onerror)
sm_com_set_prop	Panther 4.2	Sets a property for a COM component (Recommended alternate: sm_obj_set_property)

Function Name	Made Obsolete In	Description
sm_com_obj_create	Panther 4.2	Instantiates a COM component (Recommended alternate: sm_obj_create)
sm_com_obj_destroy	Panther 4.2	Destroys a COM component (Recommended alternate: sm_obj_delete_id)
sm_com_raise_exception	Panther 4.2	Sends an error code back to the client (Recommended alternate: sm_raise_exception)
sm_com_receive_args	Panther 4.2	Receives a list of in and in/out parameters for a method (Recommended alternate: sm_receive_args)
sm_com_return_args	Panther 4.2	Returns a list of in/out and out parameters for a method (Recommended alternate: sm_return_args)
sm_create_id	JAM 7	
sm_destroy_id	JAM 7	
sm_finquire	JAM 7	
sm_fldno	JAM 7	
sm_fldno	JAM 7	
sm_fset	JAM 7	
sm_ftype	JAM 7	
sm_getcurno	JAM 7	
sm_get_prop	JAM 7	
sm_gp_inquire	JAM 7	
sm_iselected	JAM 7	
sm_n_bld_fldno	JAM 7	
sm_length	JAM 7	
sm_max_occur	JAM 7	

Function Name	Made Obsolete In	Description
`sm_name`	JAM 7	
`sm_novalbit`	JAM 7	
`sm_num_occurs`	JAM 7	
`sm_oshift`	JAM 7	
`sm_prop_errno`	JAM 7	
`sm_protect`	JAM 7	
`sm_query_msg`	JAM 7	
`sm_rscroll`	JAM 7	
`sm_sc_max`	JAM 7	
`sm_set_prop`	JAM 7	
`sm_sibling`	JAM 7	
`sm_size_of_array`	JAM 7	
`sm_t_scroll`	JAM 7	
`sm_t_shift`	JAM 7	
`sm_unprotect`	JAM 7	
`sm_viewport`	JAM 7	

sm_com_call_method

Calls a method of a COM component or ActiveX widget

```
char *sm_com_call_method(char *method_spec);
```

`method_spec`
A string specifying the method and its parameters consisting of the following:

`object_id`
An integer handle identifying the `COM` component whose method you want to call. The handle is returned through `sm_com_obj_create` for `COM` objects, `sm_prop_id` for ActiveX controls.

`method`
The name of the method. Periods are allowed as part of the method specification, as in: `Application.Quit`

`p1, p2, ...`
(Optional) A comma-delimited list of the method's parameters. Unused parameters are allowed to be omitted, as in:

```
sm_com_call_method ("TreeView, \"Add\" , , , , 'First node'")
```

If the COM object’s typelib cannot be used to determine the parameter's type, `@obj can` be used to specify the object ID of the parameter. Generally, this syntax will not be necessary.

Environment Windows, Web

Scope Client

Returns On Windows platforms:

- The value returned by the COM component, converted to a string.
- A null string if an error occurred. For the error code, call `sm_com_result`. Error codes are defined in winerror.h.

On UNIX platforms for ActiveX controls:

0	Success.
-1	Failure.

Description

sm_com_call_method calls methods that are part of COM components' inter faces. Usage of this function is no longer recommended; use sm_obj_call instead to ensure component portability.

To find which methods are available, refer to the documentation supplied with the COM component, use the Panther AxView utility, or use the View→Component Interface in the editor for service components.

This function returns a string; the COM component itself can return different types of data.

If you get a “type mismatch” error, refer to the component documentation and check that all the parameters are of the correct type. @obj may be needed if any of the parameters must be passed as objects.

Example

```
// This C function calls the InsertNode method of the
// ActiveX treeview control.

char method[100];
char *parent;
char *child;
int id;

id = sm_prop_id ( "treeview" );
sprintf( method, "InsertNode(%s, \"Child node\"", parent) );
child = sm_com_call_method( id, method );

// This is the JPL call for this method. Single quotation
// marks are used surrounding the method in order to pass
// double quotation marks to the method itself.

vars parent
vars child

child = sm_com_call_method \
   ( treeview->id, "InsertNode", :parent, "Child node")


// These JPL procedures instantiate the cCustomers COM
// component and call its GetCustomer method.

vars id

proc entry
```

```
id = sm_com_obj_create("cCustomers")
return

proc GetCustomer
call sm_com_call_method(id, "GetCustomer", \
   CompanyName, CustomerID, Phone)
return

// This JPL procedure closes down Microsoft Excel
// that is running as a COM component.

proc close
call sm_com_call_method(ExcelID, "Application.Quit")
return
```

See Also `sm_obj_call`

sm_com_get_prop

Gets the value of a property from a COM component

```
char *sm_com_get_prop(int obj_id, char *prop);
```

obj_id
: An integer handle that identifies the COM component whose property you want to get. The handle is returned through `sm_com_obj_create` for COM objects, `sm_prop_id` for ActiveX controls.

prop
: The designated COM property. For indexed properties, use brackets to specify the occurrence.

Environment Windows, Web

Scope Client

Returns

- The property's current value, returned as a string.
- A null string if an error occurred. For the error code, call `sm_com_result`. Error codes are defined in `winerror.h`.

Description `sm_com_get_prop` retrieves property values from a COM component. Usage of this function is no longer recommended; use `sm_obj_get_property` instead to ensure component portability.

Example

```
#include <smuprapi.h>
{
id = sm_prop_id( "spinner" );
value = sm_com_get_prop ( id, "prop" );
}

// For an indexed property:
{
id = sm_prop_id( "spinner" );
value = sm_com_get_prop ( id, "prop[5]" );
}
```

See Also `sm_obj_get_property`

sm_com_log

Writes a message to an error log

```
int sm_com_log(char *msg);
```

msg

Message to be printed to the log.

Environment	Windows
Scope	Server

Description

`sm_com_log` writes messages to an error log named server.log in the COM component's application directory. Usage of this function is no longer recommended; use `sm_log` instead to ensure component portability.

When this file is activated, in addition to the messages logged with this function, messages are automatically logged when service components are created or destroyed. All messages that would normally appear on the message line or message window are also logged.

During development you should always enable error logging by creating server.log. In production server.log should not be present as logging is a substantial load on the system.server.log must exist in the application directory for logging to take place.

See Also `sm_log`

sm_com_obj_create

Instantiates a COM component

```
int sm_com_obj_create(char *name);
int sm_c_com_obj_create(char *clsid);
```

clsid

Specify the `CLSID` (Class ID) of the COM component.

name

Specify the name of the COM component.

Environment	Windows
Scope	Client
Returns	1 Success: the component's object ID.

Description

`sm_com_obj_create` instantiates COM components. Usage of this function is no longer recommended; use `sm_obj_create` instead to ensure component portability. The COM component must support the `IDispatch` interface (the “automation interface”) so that properties can be set and methods can be called. For returns greater than zero, the call has succeeded and the object ID will be used to access the component.

For ActiveX controls, you do not call this function. The ActiveX container in the Panther editor specifies the name and `CLSID` of the component, and makes this call on your behalf.

Example

```
//The following JPL procedures create a variable
// and instantiate the control.

vars id

proc entry
id = sm_com_obj_create("cCustomers")
return
```

See Also `sm_obj_create`

sm_com_obj_destroy

Removes a COM component

```
int sm_com_obj_destroy(int obj_id, int force);
```

obj_id

An integer handle that identifies the COM component you want to delete, obtained through `sm_com_obj_create`.

force

Specify the type of removal.

If force is zero, only the specified component will be destroyed. Any components obtained as values of a property of the component or returned by the component's method calls will be retained.

If force is zero, then these derived components will also be destroyed.

Environment	Windows
Scope	Client
Returns	0 Always.

Description

`sm_com_obj_destroy` removes COM components. Usage of this function is no longer recommended; use `sm_obj_delete_id` instead to ensure component portability. If force is zero, only the specified COM component will be destroyed. Any components obtained as values of a component property or returned by method calls will be retained. If force is non-zero, then these derived components will also be destroyed.

the COM object will be removed. Otherwise, the COM component will continue to exist until the last of the dispatch interfaces is deleted by a call to `sm_com_obj_destroy`.

For example, the following JPL code creates a treeview control and calls its properties for nodes, node and font:

```
vars tree, nodes, node, font
tree = sm_com_obj_create ("treeview")
nodes = sm_com_get_prop (tree, "Nodes")
```

```
node = sm_com_get_prop (tree, "Item[1]")
font = sm_com_get_prop (node, "FontName")
```

To destroy the COM component `tree` but not nodes, node or font:

```
call sm_com_obj_destroy (tree, 0)
```

To destroy `node` and `font` (but not `tree`):

```
call sm_com_obj_destroy (nodes, 1)
```

To destroy the COM component `tree` and all derived components (`node` and `font`):

```
call sm_com_obj_destroy (tree, 1)
```

Example

```
// This JPL procedure destroys the component
// referenced by the variable objID on screen exit.

proc exit
{
call sm_com_obj_destroy (objID, 1)
}
```

See Also `sm_obj_delete_id`

sm_com_onerror

Installs an error handler

```
void sm_com_onerror(char *handler);
```

handler
: The name of the error handler. This C function or JPL procedure will be passed three parameters:

  errorNumber
  : The error number as an integer. Use this value to test for errors.

  errorHexadecimal
  : The error number as a string in hexadecimal format, as in `0x80000307`. (This parameter is displayed by the default error handler.)

  errorMessage
  : The text description of the error. (This parameter is displayed by the default error handler.)

Environment Windows

Scope Client

Description `sm_com_onerror` specifies the error handler that will be called if a COM operation returns a negative exception. The return from the handler is ignored. Usage of this function is no longer recommended; use `sm_obj_onerror` instead to ensure component portability.

An error handler will only be fired on negative exception codes; use `sm_com_result` to retrieve positive exceptions.

If you do not want an error handler, you must install your own error handler that simple returns.

To restore the default error handler, use `sm_com_onerror(0)`.

Example

```
// This JPL module specifies an error handler
// similar to the default error handler.

call sm_com_onerror (handler)

proc handler (errnum, errhex, errmsg)
msg emsg "COM Error: " errhex " " errmsg
return
```

See Also `sm_obj_onerror`

sm_com_raise_exception

Sends an error code back to the client

```
int sm_com_raise_exception(int error);
```

error

Error code to be returned to the client.

Environment	Windows
Scope	Server
Description	`sm_com_raise_exception` sends an error code back to the client. The client's error handler then decides what to do based on the value sent. Microsoft defines some conventional exception codes for use in COM programming; see `winerror.h`. Usage of this function is no longer recommended; use `sm_raise_exception` instead to ensure component portability.
See Also	`sm_raise_exception`

sm_com_receive_args

Receives a list of in and in/out parameters for a method

```
int sm_com_receive_args(char *text);
```

`text`
: List of in/out and out parameters, separated by commas, of field names and global JPL variables.

Environment: Windows

Scope: Server

Returns:

0	Not an COM method.
1	Success

- Otherwise, a value from `smerror.h`

Description: `sm_com_receive_args` receives a list of arguments, and writes them to the in and in/out parameters of a method. The arguments are written to the parameters in the order received. Usage of this function is no longer recommended; use

`sm_receive_args` instead to ensure component portability.

Example: See the example under the JPL verb `receive_args`.

See Also: `sm_receive_args`

sm_com_return_args

Returns a method's in/out and out parameters

```
int sm_com_return_args(char *text);
```

`text`
List of in/out and out parameters, separated by commas.

Environment	Windows
Scope	Server

Returns

0	Not an COM method.
1	Success

- Otherwise, a value from `smerror.h`

Description

`sm_com_return_args` is passed a list of arguments to be written to the in/out and out parameters of a method. Usage of this function is no longer recommended; use `sm_return_args` instead to ensure component portability.

Example

See the example under the JPL verb `receive_args`.

See Also

`sm_return_args`

sm_com_set_prop

Sets the value of a property of a COM component

```
int sm_com_set_prop(int obj_id, char *prop, char *val);
```

`obj_id`
: An integer handle that identifies the COM component whose property you want to set. The handle is returned through `sm_com_obj_create` for COM objects, `sm_prop_id` for ActiveX controls.

`prop`
: The designated property. Refer to the Description for information about available properties.

`val`
: The value to set for the specified property or property item. Panther converts the value to the type expected by the component.

: If the value needs to be sent as an object ID, `@obj()` can be used to specify the object ID.

Environment Windows, Web

Scope Client

Returns On Windows platforms:

- The `HRESULT` from the most recent COM component function call. Refer to `winerror.h` for values; 0 is the value for `S_OK`.

On UNIX platforms:

0	Success.
-1	Failure.

Description `sm_com_set_prop` sets the value of the specified COM component property. Usage of this function is no longer recommended; use `sm_obj_set_property` instead to ensure component portability.

When setting properties for ActiveX controls, this function can be used on all platforms for `CLSID` and Control Name and only on Windows for properties of the ActiveX control itself.

The properties for a COM components can be determined through the AxView utility, the Properties window for ActiveX controls, and the Component Interface window for service components.

In property specifications, periods are allowed. For example:

```
sm_com_set_prop (Excel_Sheet,
   "ActiveSheet.Cells(1,1).Value", text)
```

For indexed properties, use brackets to specify the occurrence. For example:

```
sm_com_set_prop (id, "prop[5]", value)
```

Example

```
#include <smuprapi.h>
int id;
int retcode;

{
id = sm_prop_id( "spinner" );
retcode = sm_com_set_prop ( id, "Value", "40" );
}
```

See Also `sm_obj_set_property`

Index