

## Contents:

### About This Document

#### 1. Overview of Panther Web Applications

How the Internet Works .....	1-1
Components of a Panther Web Application .....	1-5

#### 2. Web Application Setup

Web Application Components .....	2-1
Setting Up the Web Application Server .....	2-5
Creating a New Web Application.....	2-9
Setting Web Browser Options .....	2-12
Firewalls .....	2-12

#### 3. Setting Properties for Web Applications

Screen Properties .....	3-1
Widget Types.....	3-4
Widget Properties .....	3-5
Font Properties.....	3-16
Application Properties .....	3-18

#### 4. Opening Screens

Processing Screen Requests .....	4-2
Transmitting Screens Securely .....	4-6

#### 5. Web Events

Web Event Hooks .....	5-1
Web Application Events.....	5-5
Controlling Entry Processing .....	5-7

#### 6. Preserving Application State

Caching Data .....	6-2
--------------------	-----

---

Saving State Data in Cookies .....	6-7
Unpreserved State Information.....	6-7
<b>7. JPL Globals in Web Applications</b>	
Application Globals .....	7-1
Context Globals .....	7-2
Transient Global Variables.....	7-3
<b>8. Customizing HTML Generation</b>	
Setting Custom HTML Properties.....	8-2
Using HTML Templates.....	8-4
Using Hyperlinks.....	8-11
Setting Target Windows .....	8-13
Specifying the Browser's Title Bar.....	8-14
Using Graphics .....	8-15
Using the FRAME Extension.....	8-18
Using Style Sheets .....	8-19
Creating Headings .....	8-20
Drawing Horizontal Rules .....	8-20
Using Cookies.....	8-21
Embedding Java Applets .....	8-22
Refreshing Screens in a Web Browser .....	8-24
Using ActiveX Controls .....	8-24
Embedding Sound.....	8-29
<b>9. Using JavaScript and VBScript</b>	
Browser Events.....	9-2
<b>10. Accessing Databases</b>	
Connecting to the Database .....	10-1
Initializing the Panther Client.....	10-2
Using Database Cursors.....	10-3
Database Transactions .....	10-3
Fetching Multiple Rows .....	10-4

---

## 11. HTTP Variables

Definitions .....	11-2
-------------------	------

## 12. Web Initialization Options

Setup Variables.....	12-1
Behavior Variables .....	12-4
Sample Initialization File .....	12-9

## 13. Deploying Web Applications

How to Configure a Panther Web Application.....	13-1
---	------

### A. Web Application Utility

### B. Web Setup Manager

Using the Web Setup Manager.....	B-2
----------------------------------	-----

### C. Setting Up an NSAPI Web Server

Configuring Your NSAPI-Compliant Server .....	C-1
Accessing the Panther Web Application .....	C-2
A Sample Obj.conf File .....	C-3

### D. Using Java Servlets

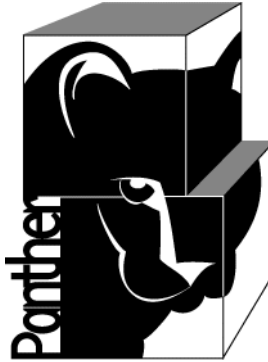
Installing Java Servlet Support.....	D-1
Accessing the Panther Web Application .....	D-2
Panther's Java Servlet Classes .....	D-3

### E. Sample Web Applications

General Applications .....	E-1
Feature-Specific Examples .....	E-3

## Index





# Panther

## Web Development Guide

***Prolifics.***

Release 5.51

Document 0404

March 2017

## Copyright

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

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

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

CORBA is a trademark of the Object Management Group.

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

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

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

INGRES is a registered trademark of Actian Corporation.

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

Linux is a registered trademark of Linus Torvalds.

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

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

Mozilla and Firefox are registered trademarks of the Mozilla Foundation.

Netscape is a registered trademark of AOL Inc.

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

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

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

VeriSign is a trademark of VeriSign, Inc.

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

Send suggestions and comments regarding this document to:

Technical Publications Manager

Prolifics, Inc.

24025 Park Sorrento, Suite 405

Calabasas, CA 91302

<http://prolifics.com>

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

(800) 458-3313

© 1996-2017 Prolifics, Inc.

All rights reserved.

# Contents:

## About This Document

Documentation Website .....	xi
How to Print the Document.....	xii
Documentation Conventions .....	xii
Contact Us! .....	xiv

## 1. Overview of Panther Web Applications

How the Internet Works .....	1-1
Retrieving Documents with URLs .....	1-2
HTTP.....	1-3
Formatting Documents With HTML.....	1-3
Viewing HTML in Web Browsers .....	1-4
Using the Internet and Intranets .....	1-4
Components of a Panther Web Application .....	1-5
Two-Tier Processing .....	1-6
Three-Tier Processing .....	1-6
Web Application Server Processes .....	1-7

## 2. Web Application Setup

Web Application Components .....	2-1
Requester Program .....	2-2
Application Directory.....	2-3
Cache Directory.....	2-4
Configuration Directory .....	2-4

---

Log Files .....	2-4
Logging Application Errors .....	2-5
Setting Up the Web Application Server .....	2-5
Configuring the Requester Program .....	2-6
Using CGI .....	2-6
Using ISAPI .....	2-6
Using NSAPI .....	2-6
Using Java Servlets .....	2-6
Configuring Middleware Access .....	2-7
Configuring Library Access .....	2-7
Using Remote Libraries .....	2-7
Configuring Database Access .....	2-7
Configuring a Windows Server .....	2-8
Creating a New Web Application .....	2-9
Using the Web Setup Manager .....	2-9
To run the Web setup manager: .....	2-9
Starting Your Panther Web Application .....	2-10
Using monitor .....	2-10
Starting as a Service .....	2-10
Accessing Your Panther Web Application .....	2-11
Setting Web Browser Options .....	2-12
Firewalls .....	2-12

### **3. Setting Properties for Web Applications**

Screen Properties .....	3-1
Web Options Properties .....	3-3
Widget Types .....	3-4
Widget Properties .....	3-5
Web Options Properties .....	3-5
Property Usage in Web Applications .....	3-7
Push Buttons .....	3-9
Using push buttons to perform processing in the browser .....	3-10
Selection Groups .....	3-10
Grid Widgets .....	3-10
Dynamically Resizing Grids .....	3-11



---

Selecting and Modifying Grid Data .....	3-11
Scrolling Grids .....	3-13
Widget Positioning .....	3-14
Overlapping Widgets .....	3-14
Horizontal and Vertical Anchors .....	3-14
Snap to Grid .....	3-15
Spacing of Widgets .....	3-15
Positioning Regions .....	3-15
Maximum Usage of Space .....	3-16
Fonts .....	3-16
Font Properties.....	3-16
Font Name .....	3-16
Font Size.....	3-17
Application Properties .....	3-18

## 4. Opening Screens

Processing Screen Requests .....	4-2
Specifying a URL.....	4-2
Encoding Parameters in the URL.....	4-3
Example .....	4-4
Encoding ASCII Characters.....	4-4
Transmitting Screens Securely .....	4-6
Setting Screen Properties .....	4-6
Determining Screen Sequence .....	4-6
Configuring Your Web Application.....	4-7

## 5. Web Events

Web Event Hooks.....	5-1
Specifying Web Event Hooks .....	5-4
Web Application Events.....	5-5
Web Application Server Events .....	5-5
Screen Events .....	5-5
Controlling Entry Processing .....	5-7
Screen Entry Context Flag .....	5-7
Screen Entry Processing and Option Menus .....	5-8

---

Web Entry Context Flags .....	5-9
<b>6. Preserving Application State</b>	
Caching Data .....	6-2
Posting Screens Back to the Server .....	6-2
Getting Screens from the Server.....	6-4
Cached Data.....	6-4
Saving State Data in Cookies .....	6-7
Unpreserved State Information.....	6-7
LDB .....	6-7
Window Stack .....	6-8
Property Changes.....	6-8
<b>7. JPL Globals in Web Applications</b>	
Application Globals .....	7-1
Context Globals .....	7-2
Transient Global Variables.....	7-3
<b>8. Customizing HTML Generation</b>	
Setting Custom HTML Properties .....	8-2
Screen Custom HTML Properties .....	8-2
Widget Custom HTML Properties .....	8-3
Using HTML Templates.....	8-4
HTML Template Tags.....	8-5
HTML Template Document.....	8-7
Conditional Processing.....	8-7
Passing Database Values .....	8-8
Submitting a Form.....	8-10
Using Hyperlinks.....	8-11
Creating Hyperlinks.....	8-11
Setting the Default Link Property .....	8-11
Setting the Item Link Property .....	8-11
Calling sm_web_invoke_url .....	8-12
Placing an Action List Box Inside a Grid .....	8-12
Using Hyperlinks in Reports .....	8-12
Setting Target Windows .....	8-13

---

Setting the Window for a Specific Hyperlink .....	8-14
Setting the Default Window for All Screen Hyperlinks .....	8-14
Setting the Window for a Screen .....	8-14
Specifying the Browser's Title Bar .....	8-14
Using Graphics .....	8-15
Setting Graphics Size .....	8-16
Graph Widgets.....	8-16
Image Maps .....	8-16
Creating Image Maps in JPL.....	8-17
Loading Graphics at Runtime .....	8-17
Using the FRAME Extension .....	8-18
Using Style Sheets .....	8-19
Creating Headings .....	8-20
To create a heading: .....	8-20
Drawing Horizontal Rules .....	8-20
To create a horizontal rule: .....	8-21
Using Cookies .....	8-21
Retrieving Cookie Values .....	8-22
Embedding Java Applets .....	8-22
To embed a Java applet into your Panther screen:.....	8-23
Refreshing Screens in a Web Browser .....	8-24
To specify a META tag:.....	8-24
Using ActiveX Controls .....	8-24
Using ActiveX Controls in Web Browsers .....	8-25
Signing Your ActiveX Controls .....	8-26
Submitting Data to the Web Application Server.....	8-26
Submitting Data using JavaScript .....	8-27
Submitting Data using VBScript.....	8-27
Generating HTML for ActiveX Controls.....	8-27
Embedding Sound .....	8-29
To embed a sound file in your application:.....	8-30

## **9. Using JavaScript and VBScript**

Browser Events.....	9-2
JavaScript Event Properties.....	9-2

---

Screens .....	9-2
Widgets.....	9-3
Setting Event Properties .....	9-4
Writing JavaScript and VBScript Functions .....	9-4
How to Define a JavaScript or VBScript Function .....	9-4
Accessing Widget Values.....	9-6
Accessing Widgets in JavaScript and VBScript.....	9-6
Automatic JavaScript Generation.....	9-7
<b>10. Accessing Databases</b>	
Connecting to the Database .....	10-1
Initializing the Panther Client.....	10-2
Using Database Cursors.....	10-3
Database Transactions .....	10-3
Fetching Multiple Rows .....	10-4
<b>11. HTTP Variables</b>	
Definitions .....	11-2
<b>12. Web Initialization Options</b>	
Setup Variables.....	12-1
Required Settings.....	12-2
Required for JetNet/Oracle Tuxedo Applications .....	12-2
Optional Settings .....	12-3
Database Information .....	12-4
Behavior Variables .....	12-4
Required Settings.....	12-4
Optional Settings .....	12-5
Sample Initialization File.....	12-9
<b>13. Deploying Web Applications</b>	
How to Configure a Panther Web Application.....	13-1
<b>A. Web Application Utility</b>	
monitor .....	A-2

---

## **B. Web Setup Manager**

Using the Web Setup Manager.....	B-2
To create a new Panther Web application:.....	B-2

## **C. Setting Up an NSAPI Web Server**

Configuring Your NSAPI-Compliant Server .....	C-1
Accessing the Panther Web Application .....	C-2
A Sample Obj.conf File .....	C-3

## **D. Using Java Servlets**

Installing Java Servlet Support.....	D-1
To install Java servlet support:.....	D-2
Accessing the Panther Web Application .....	D-2
To access the Panther web application using Java servlets: .....	D-2
Panther's Java Servlet Classes .....	D-3
Methods.....	D-3

## **E. Sample Web Applications**

General Applications.....	E-1
Feature-Specific Examples .....	E-3

## **Index**



# About This Document

The *Web Development Guide* contains information about developing and deploying Panther web applications. Using Panther, you have two methods to use to deploy your web applications:

- You can take an existing Panther GUI application and move it to the web environment by generating HTML at runtime for your client screens.
- You can write HTML documents that call screens in your Panther application in order to get and update data.

For an introduction to taking your Panther application to the web, refer to the *Getting Started 2-Tier* manual.

---

## Documentation Website

---

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

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

---

# How to Print the Document

---

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

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

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

---

# Documentation Conventions

---

The following documentation conventions are used throughout this document.

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



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

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

---

---

## Contact Us!

---

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

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

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

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

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

# 1 Overview of Panther Web Applications

Panther lets you easily build fully transactional, database-oriented applications to deploy on the Internet or an intranet. To understand how Panther works, this chapter presents:

- An introduction to the Internet: see [page 1-1](#), “How the Internet Works.”
- How the Panther software components operate in a web environment: see [page 1-5](#), “Components of a Panther Web Application.”

---

## How the Internet Works

---

The Internet is a global collection of computer networks. Because each computer on this network is able to communicate with any other, any resource that is on one is available to all. A resource can be a document, a picture, a sound file, a video clip—any information that can be stored and presented electronically.

Computers on this global network can communicate for these reasons:

- They all use the same routing protocol, or language, called the Internet Protocol (IP).
- Each computer has a unique 32-bit address, called the IP address.

The Internet's popularity vastly increased with the advent of the World Wide Web, originally designed for scientists to share information online. This information is deployed as hypertext documents that can contain hyperlinks (pointers) to other documents on the system.

The World Wide Web system uses these conventions:

- URL (Uniform Resource Locator)—Specifies the location of an Internet resource.
- HTTP (HyperText Transfer Protocol)—Specifies the protocol for communication between the client and server. HTTPS (HTTP Secure) is similar to HTTP but uses encoding to help make the communications more secure.
- HTML (HyperText Markup Language)—Defines the structure of information within a web document.

Using a client/server model, information is stored on any of the computer servers that make up the web. That information can be accessed by a variety of clients or browsers—from an ASCII terminal to a PC running a graphical web browser.

In order to access documents on the web, a user starts a browser program. Every browser program can perform these tasks:

- Specify a URL for a document, including the protocol that is used to retrieve the document.
- Interpret HTML commands and display the document.

## Retrieving Documents with URLs

Each document on the web has a unique URL, which allows other browsers to retrieve it. A URL contains these components:

- The name of the protocol, or language, to use to retrieve the document. For most documents, this is HTTP. Other common protocols which are also accessible in a browser include HTTPS, FTP, WAIS, and Gopher.

- The domain name of the HTTP server where the document is stored. The name can include two or more levels.
- The port address for the HTTP server, if needed. Many servers use the default port number, 80.
- The directory location of the document, including the document name.

The following example of a URL specifies that `http` is the protocol, the domain name is `docs.prolifics.com`, and the document named `overview.htm` is located in the `panther/html/web_html` subdirectory of the HTTP server's document directory:

```
http://docs.prolifics.com/panther/html/web_html/index.htm
```

## HTTP

HTTP (HyperText Transfer Protocol) was developed specifically for transferring hypertext documents. Computer servers on the Internet, because they have programs installed to process HTTP requests, are called HTTP servers.

An HTTP server can also take data sent from the client and pass it on to programs on the server for further processing. These server programs are called gateway programs because they act as a bridge between the HTTP server and other local resources, such as databases. The interaction between the HTTP server and these programs is part of the CGI (Common Gateway Interface) specification.

## Formatting Documents With HTML

HTML (HyperText Markup Language) is used to construct web documents. Each HTML document contains a series of tags. These tags identify each element of information in the document.

The web supports different types of clients, or browsers, to access the same information. HTML tags indicate the logical structure of a document; the actual implementation of these tags depends on the given browser, which has its own way to display the various parts of the document structure. So, presentation of HTML document data can differ in each browser. For example, on an ASCII terminal a heading might be centered on the terminal and appear all in uppercase characters. The same heading on a graphical web browser might start on the left side of the page and appear in mixed case, bold characters.

Each HTML tag is enclosed in a set of angle brackets (< >). The first instance of the tag is called the start tag. The second instance, in which a slash precedes the tag (</ >), is called the end tag. For example, the start tag for the body of a document appears as <BODY>. At the end of a document, the end tag appears as </BODY>. For some tags, the matching end tag is optional and is often omitted.

## Viewing HTML in Web Browsers

Different browser programs can have different physical capabilities—for example, support for tables and graphic formats can vary. Furthermore, the browser program itself might permit the user to customize presentation of HTML elements; so two users who are using the same web browser program might see the same HTML document appear differently.

Browser presentation is especially susceptible to the base font that is specified for the browser program. A number of Panther widgets are resized according to the base font—text fields, static labels, dynamic labels, and labels on push buttons, radio buttons, and check boxes. Because text fields are also specified in HTML with a size property based on the number of characters, the resulting field changes in length according to the font size specifications in the browser.

## Using the Internet and Intranets

If you have an Internet connection, you start your browser program and enter a URL. The browser then displays the resource specified in the URL. At that time, you can enter another URL; or, more typically, click on a hyperlink. A hyperlink's definition includes the URL of another web resource, which can be located anywhere on the Internet. When you click on a hyperlink, it opens that resource in your browser.

A web can be viewed as a collection of hyperlinked documents that exist either on the Internet or an intranet. Intranets and the Internet are different only in the scope of the network: an intranet only provides resources for a single company or organization.

The web is designed for easy access from one resource to another; it is not designed to retain information from previous resources—that is, it is stateless. Neither client nor server stores information about the other. After the server returns a resource to the browser, the connection between them ends. Each new request requires a new connection. The restrictions inherent in this stateless environment present a significant challenge to deploying transactional database applications on the web.

With Panther, however, you can build web applications that meet complex businesses requirements quickly and easily.

Panther has built-in components to store the state information needed for your application.

---

## **Components of a Panther Web Application**

---

For a Panther web application, like any Panther application, you first develop your application screens and service components in the Panther editor. With its repository, screen wizard, and graphical, drag and drop environment, you have the tools to build your application quickly.

After the application's libraries are constructed, you can deploy the application on the Panther web application server, which works with your HTTP server software. Popular HTTP servers include Microsoft's Internet Information Server and Apache. The screens in your application can be requested from the HTTP server by entering the URL for the screen in a web browser.

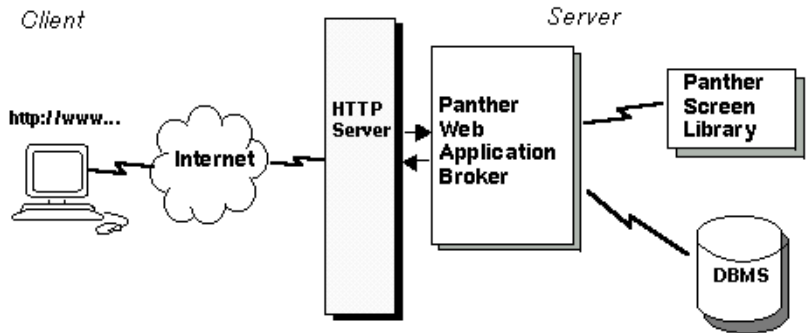
When a browser request comes in for a Panther screen, the HTTP server passes the requested screen name to the Panther web application server. This is called a GET in the HTTP protocol. The web application server opens the screen, generates HTML for the screen, and passes the HTML back to your HTTP server, which transmits the data back to the browser that requested it.

In an HTML document, the Panther screen is translated into an HTML form. Embedded in this form is the screen's URL. After the user interacts with the screen and presses one of its push buttons, the screen is submitted back to the HTTP server. In the HTTP protocol, this is called a POST. The embedded URL tells Panther which screen to reopen. At that time, data from the HTML form is mapped into the Panther screen, and any processing attached to the activated push button is performed.

When these steps are complete, Panther generates HTML for the current screen and sends that data to the HTTP server, restarting the cycle. This cycle of browser requests occurs for all Panther web applications. However, the role of the web application server varies for two-tier and three-tier applications.

## Two-Tier Processing

In traditional two-tier client/server processing, the web application server is responsible for making the database connections, opening the screens, mapping the browser and data cache values onto the screens, fetching data from the database, and generating HTML.



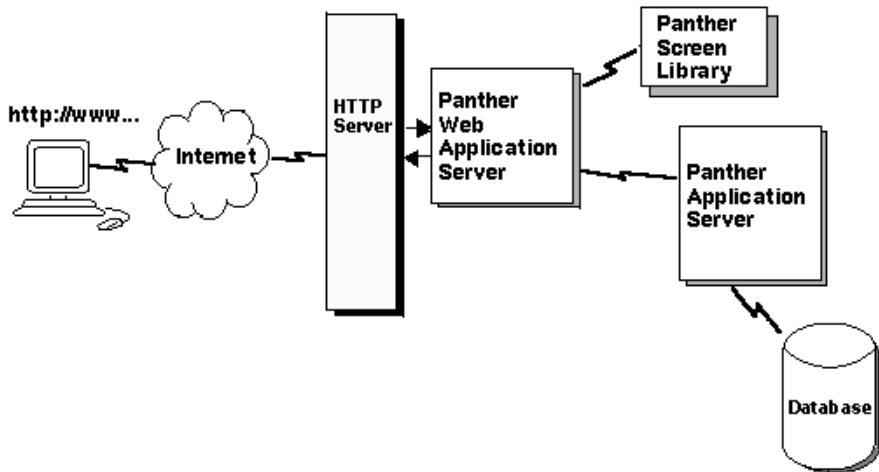
**Figure 1-1** When the HTTP server receives a request from a browser for a Panther screen, the request is sent to Panther for processing.

## Three-Tier Processing

In three-tier distributed processing, the Panther application server establishes and maintains connections to the database. When the web application requires data, the web application server, acting in the role of client, sends the service request to the Panther application server. The Panther application server processes this request and returns the desired data to the web application server.

In a three-tier environment, the web application server remains responsible for opening screens, mapping browser and data cache values, generating HTML.





**Figure 1-2** In three-tier processing, the web application server is a Panther client.

## Web Application Server Processes

When a web browser requests a Panther screen, the HTTP server passes the screen name to the web application server via the CGI, ISAPI, or NSAPI interface. The web application server opens the screen, generates HTML, and passes the HTML back to the HTTP server, which transmits the data back to the browser.

A web application server consists of three processes: requester, dispatcher, and jserver. By dividing its work among these separate processes, a web application server optimizes response time to each request and provides efficient service and options for high-volume applications.

### Requester

Accepts a request from the HTTP server, passes it to a jserver process, and then waits for and transmits a response. There are three versions of the requester program, one for each type of interface: CGI, ISAPI, and NSAPI.

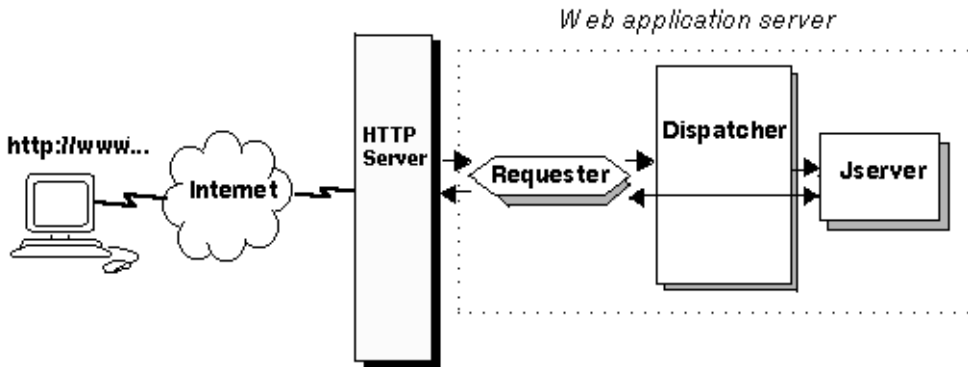
### jserver

After receiving the request, performs all application-specific processing and, in three-tier applications, for routing service requests to the Panther

application server. Unlike the requester, which shuts down after servicing a request, the jserver is always running. This allows Panther to provide prompt service to concurrent requests.

Dispatcher  
Manages the pool of jservers.

Figure 1-3 represents the interaction between the web application server processes:



**Figure 1-3** The user request (URL) to the HTTP server initiates the startup of the web application server processes.

Panther's web application server architecture efficiently allocates server processes on a per-server basis, not a per-user basis, and reuses them for each request. This feature significantly reduces the load on your server.

# 2 Web Application Setup

This chapter describes:

- The components of a Panther web application.
- How to set up the web application server
- How to create a new web application.

---

## Web Application Components

---

A complete Panther web application has these components:

- Libraries that contain the application's screens, service components, reports, JPL modules, and graphics files.
- An initialization file that specifies the application's configuration settings.
- Server executables: requester, dispatcher, and jserver.
- Panther libraries and configuration files that are required by the web application server.

These components, created during installation and with the web setup manager, are distributed among different directories. This section describes these components and their location.

## Requester Program

Each Panther web application needs a copy of, or link to, the distributed requester executable in the HTTP server's program directory. Using the web setup manager to create your web application automatically copies the requester executable for your application.

There are three different versions of the requester program:

- CGI (Common Gateway Interface)—The default name is `proweb` or, for Windows, `proweb.exe`; the application's version is `application-name[.exe]`. This version uses the CGI protocol to pass the requester information to the jserver.
- ISAPI (Microsoft's Internet Information Server API)—The default name is `proweb.isa`; the application's version is `application-name.isa`.
- NSAPI (Netscape's Web Server API)—The default name is `proweb.nsa`; the application's version is `application-name.nsa`.

Common names for the program directory on the HTTP server are `cgi-bin` or `scripts`.

**Notes:** For the Apache HTTP server, use the directory specified for Standard CGI in the Apache Administration dialog. Typically, this is `cgi-bin` or `scripts`, not `cgi-win`.

The application name, which is used in the file name for the initialization file as well as the requester executable, must be unique and follow the naming conventions for the operating system.

The permissions on the requester program must allow the program to be executed by the HTTP server.

Alternatively, the Panther web application can run as a Java servlet. For more information, refer to Appendix D, “Using Java Servlets.”

Each Panther web application must have an initialization file whose file name matches the name of the requester executable in the format application-name.ini. The name of the distributed initialization file is `proweb.ini` and the name of the requester executable is `proweb[.exe]`. Using the web setup manager to create your web application will automatically create the initialization file for your application.

The location of the initialization file is different for Windows and UNIX:

- The UNIX installation creates an ini subdirectory in the home directory of the `proweb` user. The initialization files are located in that ini directory. Permissions on the initialization file must allow it to be read by the HTTP server. The permissions on the ini subdirectory and its parents must allow the directory to be read and executed by the HTTP server.
- The Windows installation copies the default initialization file `proweb.ini` to the `%WINDDIR%` directory where Windows is installed.

For more information about initialization file settings, refer to Chapter 12, “Web Initialization Options.” For instructions on using the web setup manager, refer to Appendix B, “Web Setup Manager.”

## Application Directory

The application directory contains the files and libraries for Panther screens and reports. All references to Panther files should be relative to this directory. This directory can also contain JPL modules, graphics files, and other files referenced by screens and reports. If these files are not in this directory, they must be in a directory specified by the Panther environment variable `SMPATH`.

The `AppDirectory` variable in the initialization file specifies the location of the application directory on your HTTP server. On startup, the `jservlet` changes its working directory to the application directory.

Panther opens files named in URLs relative to this directory and uses this directory as the basis for any relative path name references in Panther. For security purposes, Panther strips the parent directories from the path name. This means that CGI variables, such as `PATH_TRANSLATED`, do not contain the absolute path specified in this variable.

## Cache Directory

In order to preserve application values during a series of browser requests, Panther caches data whenever it transmits a screen that can be posted back to the web application server. Generally, the data is stored in a cache file on the web application server. The `CacheDirectory` variable determines the cache directory location; other variables determine how the cache files are removed.

On UNIX, the directory permissions must allow the HTTP server to read, write, and execute this directory.

**Warning:** When you create this directory, make sure that the period character (.) does not appear anywhere in the path name. (This restriction is imposed by the Graftsman software that manages Panther graphs.)

## Configuration Directory

The application must know the location of `smvars.bin` in the `config` directory of the web application server distribution. Generally, the web application server determines the location of this file by looking at the setting for `SMBASE`, one of the application variables. If the directory specified in `SMBASE` contains the `config` subdirectory, this is the only specification needed. Otherwise, the `SMVARS`, environment variable must specify the location of `smvars.bin`.

The initialization file can contain application-specific settings for any of the environment variables named in `smvars.bin`. The setting in the initialization file takes precedence.

## Log Files

Errors occurring on the web application server are written to the log file set by the `ErrorFile` variable in the initialization file. Additional client and server information is written to the files specified by the `ClientLog` and `ServerLog` variables. In order to better track your web application server usage, you can specify that the `ClientLog` and `ServerLog` variables write information to the same file. Client logs contain information about the requester process. Server logs list configuration information and jserver usage.

**Note:** Application errors are displayed in the browser and are not logged to the error file.

Using client and server logs does affect performance. You may choose to run client and server logs to test your web application, but not use them during deployment.

In three-tier applications, the time entered in the error logs is based on the server clock and will not correspond to the ULOG entries which are based on UTC (Coordinated Universal Time).

## Logging Application Errors

In a Panther web application, errors can be logged to the web application server by calling `sm_web_log_error` in an error handler. The error messages are appended to the file named in the `ErrorFile` variable. Since the error message is not displayed to the user, you must make a separate call to `sm_message_box`, `sm_femsg`, or their JPL msg equivalents for any user messages.

The following error handler displays a message to the user and logs a separate message to the web error file:

```
proc error_def
  msg emsg "Error: File not found"
  call sm_web_log_error ("Unable to find file.")
  return
```

---

# Setting Up the Web Application Server

---

The Panther web application server must be installed on a machine that is running an HTTP server. After installing the software, the web application server must be configured.

## Configuring the Requester Program

As discussed in previous sections, the Panther web application server has three components, requester, dispatcher, and jserver, and there are three different versions of the requester program for the various protocols.

Since CGI uses a separate process for each request if the requests happen concurrently, the performance is slower for the CGI version of an application. Using the HTTP server's APIs, like ISAPI or NSAPI, will improve performance.

The requester executables distributed with the web application server are located at:

```
$SMBASE/util/proweb*
```

Copy the appropriate version to the HTTP server's program directory. Common names for the program directory are cgi-bin or scripts.

### Using CGI

To use the CGI (Common Gateway Interface) version of the requester executable, copy `proweb[.exe]` to the program directory.

### Using ISAPI

To use the ISAPI (Microsoft's Internet Information Server API) version of the requester executable, copy `proweb.isa` to the program directory.

### Using NSAPI

To use the NSAPI (Netscape's Web Server API) version of the requester executable, you must configure the NSAPI server and copy `proweb.nsa` to the program directory. For more information, refer to Appendix C, "Setting Up an NSAPI Web Server."

### Using Java Servlets

Alternatively, the Panther web application can run as a Java servlet. For more information, refer to Appendix D, "Using Java Servlets."



## Configuring Middleware Access

In a three-tier processing model, the web application server acts as a client of the application server. The web application server can connect to the enterprise application either as a local or remote client:

- As a local client that resides on either the master or non-master machine, the web application server connects to the enterprise via the local environment's `SMRBCONFIG` setting.
- As a remote client, the web application server's `jserver` component must use `jserver.ws` as its executable, set in the Web initialization file's `Server` variable; and the `SMRBPORT` and `SMRBHOST` variables, also in the Web initialization file, must be set to the application server's machine.

## Configuring Library Access

### Using Remote Libraries

In JetNet and Oracle Tuxedo applications, you cannot open a remote library on startup using `SMFLIBS= server_id!lib_name`. If a client library cannot be placed on a shared file system, connect to the middleware on application startup using `client_init` and execute the Panther library function `sm_l_open` to open a remote library.

## Configuring Database Access

For two-tier UNIX web applications, verify that:

- The user specified in the `DBMS DECLARE CONNECTION` statement is configured for database access.
- The web application's initialization file contains the environment variables needed for database access. (At runtime, the web application runs under an `http` process.)

For two-tier Windows web applications, verify that:

- The user that installs the web application as a service is configured for database access. That user can be specified as part of the monitor—install command or in the Services section of the Control Panel.
- The user specified in the `DBMS DECLARE CONNECTION` statement is configured for database access.
- The PC has the database vendor's client program installed.
- The web application's initialization file contains the environment variables needed for database access.

If the three-tier Windows application includes a web application client, the web application can be installed as a service. Verify that:

- The user that installs the web application as a service is configured for database access. That user can be specified as part of the monitor—install command or in the Services section of the Control Panel.

Verify that the service has database access by:

- Logging into the machine as the user (owner) of the service.
- Starting a database vendor's program.

Some database engines have special installation instructions. For example, Informix requires that you run `setnet32` to add the service user and then run the demo login program to check that the user was added correctly.

## **Configuring a Windows Server**

HTTP servers on Windows can start a web application as a Windows service. For more information, refer to [page 2-10](#), “Starting as a Service.”

---

# Creating a New Web Application

---

To create a new web application:

- Run the Web setup manager to create the initialization file and the requester program.
- Start the application using the monitor utility.
- For Windows servers, start the application as a Windows service.

## Using the Web Setup Manager

The Web setup manager utility creates the files necessary for a new web application and allows you to modify the settings of existing web applications.

### To run the Web setup manager:

Start your Web browser and enter the following URL:

`http://serverName/programDirectory/websetup`

The Web setup manager consists of a series of screens; enter the information needed on each screen. You will need:

- The application name.
- The path of the application directory.
- The path of your Panther Web installation.
- The path of the HTTP server's program directory (usually called cgi-bin or scripts).

If the Web Setup Manager utility is unable to write the files to the correct directories, generally due to incorrect file permissions, it prompts for a temporary location, by default `/tmp`. At the end, the utility prompts you to move the files to their correct locations using provided script, `application-name.cfg`.

For step-by-step instructions in using the web setup manager, refer to Appendix B, “Web Setup Manager.” For more information on web initialization options, refer to Chapter 12, “Web Initialization Options.”

## Starting Your Panther Web Application

### Using monitor

You can start your web application (when it is not a service) with the `monitor` utility using the syntax:

```
monitor -start appname [appname ...]
```

and stop the web application using the syntax:

```
monitor -stop appname [appname ...]
```

### Starting as a Service

On Windows, it is recommended that the Panther web application be installed as a service. Use the `monitor` command in conjunction with the `-install` option. The full syntax for that option is:

```
monitor -install application-name  
  [-display display-name]  
  [-{automatic|manual|disabled}]  
  [-user {\domain\user|.user}]  
  [-password password]  
  [[-depend service-name]...]
```

Once an application is installed as a service, `monitor -start` and `monitor -stop` must not be used to start and stop the web application. `net start` and `net stop` can be used for manual control, or use the Services section of the Control Panel.

It is also recommended that the services needed by the Panther web application start in a specific order:

- Database engine
- Oracle Tuxedo IPC Helper (for JetNet and Oracle Tuxedo applications)
- HTTP server
- Panther Web application

## Accessing Your Panther Web Application

The type of requester program determines the syntax of the URL needed to access your application. The general syntax is:

```
http://serverName/programDirectory/applicationName/screenName
```

In the following URLs, the application name is inventory and the screen is main.scr.

To use the CGI executable to access the application:

- For Windows, copy `proweb.exe` to `inventory.exe`, start the application, and access the screen using the following URL:

```
http://myhost.com/cgi-bin/inventory.exe/main.scr
```

- For UNIX, copy `proweb` to `inventory`, start the application, and access the screen using the following URL:

```
http://myhost.com/cgi-bin/inventory/main.scr
```

To use the ISAPI executable to access the application, copy `proweb.isa` to `inventory.isa`, start the application, and access the screen using the following URL:

```
http://myhost.com/cgi-bin/inventory.isa/main.scr
```

To use the NSAPI executable to access the application, copy `proweb.nsa` to `inventory.nsa`, start the application using `monitor` or `net start`, and access the screen using the following URL:

```
http://myhost.com/panther/inventory.nsa/main.scr
```

Your HTTP server may support another syntax; refer to the HTTP server documentation.

If the web application is installed as a Java servlet, access the screen using the following URL:

`http://myhost.com/proweb/inventory/main.scr`

For more information on using Java servlets, refer to Appendix D, “Using Java Servlets.”

---

## Setting Web Browser Options

---

Caching should be enabled in the browser program. In Microsoft Internet Explorer, select Tools→Internet Options in the Browsing History Settings dialog, choose one of the Automatically or the Every time I visit the web page options.

---

## Firewalls

---

A firewall is the system that administers the access policy between two networks. The firewall determines which data is allowed network access and which data is blocked. A firewall lets users on your internal network access the global Internet, and prevents Internet users from accessing your internal network.

Panther is designed to work within whatever firewalls that are on your network. The security issues for Panther are the same issues encountered for all CGI programs. To be sure that your application operates within your company's firewall policy, check with your network or system administrator.

Your HTTP server vendor is the best source of information about how to set up the HTTP server to operate within a firewall and what security precautions are advised for CGI programs.

A Panther web application requires the CGI directory to contain the application's requester program. The remaining files in your application have configuration variables whose specifications are in the application's initialization file.

For example, the `AppDirectory` variable specifies the location of screens, reports, JPL modules, and graphics files. When Panther embeds the URL for the screen, it uses the relative path name based on the `AppDirectory` variable, not the absolute path name.





# 3 Setting Properties for Web Applications

This chapter discusses screen and widget properties and behavior that are specific to Web applications. In most respects, screens and widgets have the same properties and behavior in Web applications as on other platforms. You use the same tools—screen editor, screen wizard, debugger—to create and test Web screens. For information about these tools and common properties, refer to the *Using the Editors*.

---

## Screen Properties

---

Table 3-1 lists screen properties by category and describes their relevance in Web applications.

**Table 3-1 Screen properties and Web applications**

<b>Property Category</b>	<b>Behavior</b>	<b>Ignored properties</b>
Identity	Same	Dialog 3D Mnemonic Position
Geometry	n/a	All
Positioning	Same	
Color	Same	
Font	Same if browser supports font specification. See <a href="#">page 3-16</a> , “Font Properties.”	
Focus	Same	Menu Name Menu Script File
Help	If the browser is enabled for JavaScript, the Status Line Text entry displays in the browser's status message area  Starting in Panther 5.50, if <code>@app()-&gt;web_use_tooltips</code> is set to <code>PV_YES</code> and <code>@app()-&gt;web_alt_as_tooltips</code> is set to <code>PV_NO</code> , the Tooltip Text property is used for images.	All other help options
Display	Same	Style Border* Icon Pointer Title Bar* System Menu*
Transaction	Same	Fetch Directions
Web Options	Refer to Table 3-2	

**Table 3-1 Screen properties and Web applications**

Property Category	Behavior	Ignored properties
<i>* Set to No in order to maximize usage of browser window space.</i>		

## Web Options Properties

Table 3-2 lists the Web Options properties for screens that have property settings that are relevant only to Web applications:

**Table 3-2 Web Options properties for Web application screens**

Web Options property	Purpose	More information
Secure POST	If set to Yes, enable secure transfer of sensitive data.	“Transmitting Screens Securely” ( <a href="#">page 4-6</a> )
Display Window	Set the window or frame used to display the screen.	“Setting the Window for a Screen” ( <a href="#">page 8-14</a> )
Target Default	Set the window or frame to use as the default target for hyperlinks that are contained by the screen.	“Setting the Window for a Specific Hyperlink” ( <a href="#">page 8-14</a> )
Stylesheet Source	Set the type and location of the stylesheet for the screen.	“Using Style Sheets” ( <a href="#">page 8-19</a> )
Browser Options	Each of these properties specifies the JavaScript or VBScript function to execute when the corresponding event occurs.	Chapter 9, “Using JavaScript and VBScript”
HTML Options	The properties under Custom HTML let you add elements or attributes to the HTML that Panther generates for the screen.	“Screen Custom HTML Properties” ( <a href="#">page 8-2</a> )

---

# Widget Types

---

Table 3-3 lists Panther widget types whose presentation or behavior is different in Web applications:

**Table 3-3 Widget-type conversions and behavior in a Web application**

Widget type	Description
Push Button	Executes an action or procedure and submits the screen back to the HTTP server.
Combo Box	Converted to text widgets.
Toggle Button	If part of a multiple selection group, converted to a check box group. Single toggle buttons are converted to radio buttons.
Scales	Converted to text widgets.
List Box	Selection list boxes in grid widgets are automatically converted to columns of radio buttons or check boxes. Action list boxes in a grid widget submit the screen back to the server. They may appear as either hyperlinks or push buttons.
Tab Cards Tab Decks Vertical Lines	Not supported in browser.

---

# Widget Properties

---

Widgets have several properties that are relevant only to Web applications. A number of other properties are implemented differently in Web applications or are ignored altogether. Three widget types—push buttons, selection groups, and grid widgets—are discussed individually; widget positioning in an HTML document is also discussed separately.

## Web Options Properties

Table 3-4 lists Web Options properties that can be set for widgets used in Web applications.

**Table 3-4 Web Options properties for Web application widgets**

Web Options property	Purpose
Auto Expand	If set to Yes, specify that the grid size is sized dynamically to match the number of returned rows.
Custom HTML properties	The properties under Custom HTML let you add elements or attributes to the HTML tag that Panther generates for the widget.
Export to HTML	Generate an HTML tag for the widget even if the widget is hidden at runtime to enable scripting access in a Web browser.
Image Map	Divide the image on a dynamic label into a separate sections, each with its own hyperlink.
Ins/Del Buttons	If set to Yes, Insert and Delete buttons appear at the bottom of the grid widget if one of the grid members is unprotected.

**Table 3-4 Web Options properties for Web application widgets (Continued)**

<b>Web Options property</b>	<b>Purpose</b>
JavaScript	Write JavaScript functions that can be specified for any of the widget's Browser Event properties.
VBScript	Write VBScript functions that can be specified for any of the widget's Browser Event properties.
Browser Options	Each of these properties specifies the JavaScript or VBScript to execute when the corresponding event occurs.
Keep Image Size	If set to Yes, the GUI dimensions of the widget's image is used when it is positioned in the generated HTML.
Default Link	Designate a dynamic label or graph widget to act as a hyperlink by specifying the desired URL.
Item Link	In an array, designate a separate hyperlink for each occurrence.
Submit	If set to No, clicking on the push button will run the JavaScript or Java attached to the button and not submit the document back to the server.
Scroll Buttons	If set to Yes, scroll buttons appear to the left of the grid if the number of rows is greater than the size of the grid widget.
Target	Set the browser window or frame to use in order to display the resource that is invoked by a widget's hyperlink.

## Property Usage in Web Applications

Table 3-5 shows which properties are implemented differently in a Web application.

**Table 3-5 Properties and their behavior in Web applications**

Property	Widget type	Behavior
<b>Identity properties</b>		
Default/Cancel	Push button	Ignored.
Hidden	Any	If a widget's Hidden property is set to Yes or Always, no HTML tag is generated for the widget and it is stored in the cache file. If set to Always, no space is reserved for un hiding the widget at runtime.  To generate an HTML tag for hidden widgets, set its Export to HTML property to Yes.
Label	Dynamic/static label	Displayed in the browser's default font and point size.
Mnemonic Position	Push button	Ignored.
Name	Any	Use the widget name in JPL procedures and C functions. For JavaScript, use the HTML tag.
<b>Geometry properties</b>		
Length	Dynamic/static label, push button	Ignored; the browser uses the label's length to determine the widget's size.
Max Data Length	Multiline text	Ignored.
Scrolling	Multiline text	Ignored.
Size to Contents	Dynamic/static label, push button	Ignored; the browser uses the label length to determine the widget's size.
<b>Positioning properties</b>	Any	Same
<b>Color properties</b>	Any	Depends on browser.

**Table 3-5 Properties and their behavior in Web applications (Continued)**

<b>Property</b>	<b>Widget type</b>	<b>Behavior</b>
<b>Font properties</b>	Any	Same, if browser supports font specification.
<b>Focus properties</b>		
<b>Focus Protection</b>	Single line, multiline text	If set to Yes, widget's contents are displayed in bold text.
	Option menu, list box	Ignored.
Next/Prev Tab Stop	Any	Ignored.
<b>Help properties</b>	Any	Except for Status Line Text, help options are ignored.
Status Line Text	Any	If JavaScript is enabled, status line text for a widget displays in the browser's status message area.
<b>Input properties</b>	Single line, multiline text	If JavaScript is enabled, Panther automatically generates JavaScript functions when many input properties are set.
Edit Mask	Single line text	Only with JavaScript; otherwise, ignored even when the widget cannot receive focus.
Input Protection	Single line, multiline text	If set to Yes, widget's contents are displayed in bold text.
<b>Validation properties</b>		
Calculation	Any	Ignored.
Double Click	Any	Ignored.
<b>Format/Display properties</b>		
Active Pixmap	Push button	If a pixmap is set, onClick JavaScript event is not activated on the Web browser.
Border	Multiline text	



**Table 3-5 Properties and their behavior in Web applications (Continued)**

Property	Widget type	Behavior
Customer Drawn	Push button	Ignored.
Justification	Any	Ignored.
Password Field	Single line text	If set to Yes, text entered in the HTML form is displayed as asterisks (*). The JavaScript function that converts case cannot be specified for password fields.
Password Char	Single line text	Ignored.
Word Wrap	Multi line text	Word wrapping varies in each browser. All browsers honor hard returns and newline characters. Browser programs that support the WRAP attribute insert soft returns in the text string.

## Push Buttons

Push buttons are used in Web applications to submit the form back to the server and perform the processing associated with that button. Inactive push buttons use the Inactive Pixmap property setting; if it is not set, the push button is drawn as display text with a border.

The size of a push button is controlled by the browser, which typically uses the label length; the push button's Length and Size to Contents properties are ignored.

JavaScript or VBScript can be specified for push buttons using the onClick event; however, do not specify pixmaps in this case. In the HTML specifications, the pixmap specification takes precedence, and the JavaScript or VBScript function for the push button is not activated.

## Using push buttons to perform processing in the browser

To perform the processing attached to a push button without submitting the form back to the server, set the Submit property on the push button to No. One use for this property is when Java or JavaScript is specified for a push button. If the Submit property is No, activating the push button runs the Java/JavaScript on the Web browser without additional interaction with the HTTP server.

**Warning:** Do not use push buttons in Web applications to execute shell commands whose output is displayed back to the user. If you execute a shell command, redirect the output to a file, then display that file to the user.

## Selection Groups


You can use either radio button or check box widgets in a Web application to record the user's selection. If an application contains toggle buttons that are part of a multiple selection group, Panther converts them to a check boxes. Single-selection toggle buttons are converted to radio buttons.

If a selection group's member has its Active property set to No, the widget is removed; however, its text remains displayed.

## Grid Widgets

Three types of widgets can be included in grid widgets—dynamic labels, single line text, and list boxes. A Web application displays these widget types as follows:

- Dynamic labels appear as display text to indicate that they do not allow user data entry.
- Single line text widgets appear as input text fields if they are not protected; otherwise, they appear as bold display text.
- List boxes are displayed according to the setting of the Listbox Type property. If the Listbox Type is set to Action, the values appear as hyperlinks inside the grid. If the Listbox Type is set to Select Any, the values are converted to columns of radio buttons or check boxes—radio buttons if the group's Number of Selections property is set to 0 or 1 or 1; check boxes if Number of Selections is set to Any.

	A	B	C	D
<input checked="" type="radio"/> 1	dynamic label	text	<input type="checkbox"/> selection listbox	action listbox
<input type="radio"/> 2			<input type="checkbox"/>	
<input type="radio"/> 3			<input type="checkbox"/>	
<input type="radio"/> 4			<input type="checkbox"/>	
				

**Figure 3-1** A grid widget that contains four members: dynamic label, single line text, list box set to Select Any, and list box set to Action.

## Dynamically Resizing Grids

By default, the grid widget is created with a fixed size that is calculated from its Onscreen Rows property. If the number of rows in a grid is greater than this property, Panther inserts a column of scrolling push buttons to enable access to the unseen data (refer to [page 3-13](#), “Scrolling Grids.”). You can specify that a grid's size be set dynamically by setting its Auto Expand property (under Web Options) to Yes. When data returns, the size of the grid matches the number of returned rows. The screen size limit of 255 lines is not enforced with expandable grids.

When an expandable grid widget's size changes, all other widgets on a screen maintain their position relative to its borders. For example, when the grid expands, the widgets below it shift down. If widgets are aligned to the bottom of the grid, their positions remain unchanged, and they stay aligned to the bottom of the grid.

For sample usage, refer to the Panther Web gallery example *Expandable Grids*.

## Selecting and Modifying Grid Data

If one or more grid members is unprotected, Panther, by default, modifies the grid in two ways:

- Includes push buttons at the bottom of the grid for inserting and deleting a row (if the Ins/Del Buttons property is set to Yes).

- Inserts a column of radio buttons in the leftmost position of the grid widget (if the Radio Buttons property is set to Yes). These radio buttons visually indicate the current row selection.

If all grid members are protected, you can allow users to select a row by setting the grid widget's Stripe Current Row property to Yes. A grid thus set contains a column of radio buttons that enable user selection.

When the screen is submitted back to the server, Panther sets the `grid_current_occ` property to the selected row and performs all grid row entry and exit functions.

	ID	Name	Type	Rent Cat
Select row <input checked="" type="radio"/>	56	After Hours	COM	G
<input type="radio"/>	1	Airplane!	COM	G
<input type="radio"/>	2	Aliens	SCFI	G
<input type="radio"/>	3	All of Me	COM	G
<input type="radio"/>	60	All of Me	CLAS	G

**Figure 3-2** When the Radio Buttons property is set to Yes, radio buttons appear in the left column of the grid if one of the grid members is unprotected or if the grid's Stripe Current Row property is set to Yes.

If the Ins/Del Buttons property is set to No, the buttons to insert and delete rows are not generated. If the Radio Buttons property is set to No, the radio buttons indicating the current selection are not generated, even though the current selection is still tracked.

When screens are submitted at runtime, Panther variables contain information about the grid push button that was pressed and the grid's object ID and occurrence number. For more information, refer to [page 5-9](#), "Web Entry Context Flags."

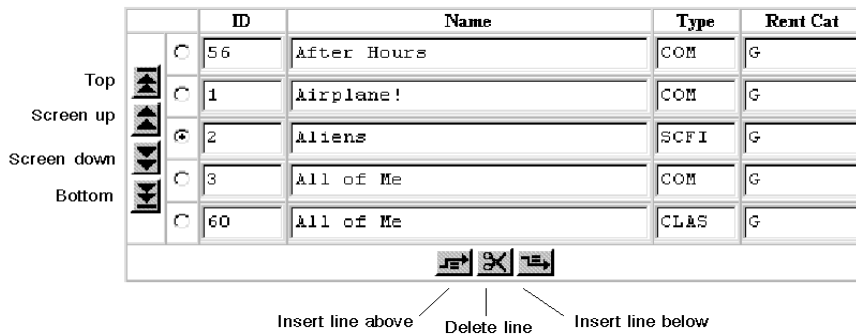
The grid push buttons derive their GIF images from a Panther library:

Push button	GIF file
Insert Above	gridsi.gif
Insert Below	gridsa.gif

Push button	GIF file
Delete Row	gridsd.gif

## Scrolling Grids

If a grid widget's size is fixed and it contains more rows than its onscreen size allows, a column of push buttons are inserted in the leftmost position of the grid widget to enable scrolling:



**Figure 3-3** A grid with unseen rows contains push buttons for scrolling all data into view.

This default behavior can be changed by setting the Scroll Buttons property on the grid widget to No.

When screens are submitted at runtime, Panther variables contain information about the grid push button that was pressed and the grid's object ID and occurrence number. For more information, refer to [page 5-9](#), “Web Entry Context Flags.”

The grid push buttons derive their GIF images from a Panther library:

Push button	GIF file
Bottom	gridbb.gif
Screen Down	griddn.gif

Push button	GIF file
Screen Up	gridup.gif
Top	gridtt.gif

The *Panther Gallery* contains sample screens entitled Action & Selection in Grids and Scrolling Grids, and is accessible from the Web application server:

`http://server-name/cgi-bin/jwsamp/main`

## Widget Positioning

To maintain widget positions within an HTML document and maximize usage of window space, the following position-specific properties and design options are provided.

### Overlapping Widgets

HTML does not support overlapping widgets. In order to quickly detect overlapping widgets, radio buttons, check boxes and labels have borders to indicate whether they overlap another widget. These borders do not appear at runtime. Additionally, to check for overlapping widgets while editing a screen, choose Edit→Find→Overlapping Widgets. To check for overlapping widgets when a screen is saved, choose Options→Check Overlap on Screen Save.

### Horizontal and Vertical Anchors

Each widget has two anchor properties—Horizontal Anchor and Vertical Anchor—to help align widgets. These properties can be set manually in the Properties window, but they are also set automatically when the screen editor's alignment feature is used. Thus, if you left-align a group of widgets, you are automatically setting their horizontal anchor properties to be left. Panther uses these anchor properties to determine how widgets are aligned with other widgets on the screen.

Each widget has one vertical and one horizontal anchor property, so you can control alignment once in each direction. If you align a widget one direction and then realign it the other way, only the latter setting of the anchor property is in effect. So, you can align a large widget like a pixmap or a grid widget relative to a widget on the left or on the right, but not both.

For example, a screen has a pixmap with several widgets to one side of it. The pixmap's Vertical Anchor property is set to middle.

Panther uses that middle point to determine where the pixmap lies relative to the widgets to the side of it. With this setting, you cannot directly control whether the topmost widget is above or below the upper border of the pixmap. To ensure that the topmost widget lies below the upper border of the pixmap, specify the pixmap's Vertical Anchor to be top and make sure that the topmost widget's vertical anchor is below the upper border of the pixmap.

## Snap to Grid

In the screen editor, select Options→Snap to Grid. With this option enabled, a new widget is automatically positioned at the nearest grid point. This makes it easier to visually align widgets.

## Spacing of Widgets

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.

## Positioning Regions

If you find that widgets are being pushed out further than expected in the generated HTML, place repeated design elements together in a positioning region. Widgets inside a positioning region keep the same relative distance to each other when the HTML is generated. A positioning region is defined through a box widget (refer to “Using Boxes as Positioning Regions” on [page 22-6](#) in *Using the Editors*); a screen can have multiple positioning regions.

## Maximum Usage of Space

Three screen properties that are otherwise ignored can affect how closely widgets can be positioned next to the browser window's borders: Border, Title Bar, and System Menu. Set all three properties to No in order to allow widgets to abut on the browser window's borders without any intermediate space.

## Fonts

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. Specifying font sizes in the Panther screen editor results in better positioning than using header tags, such as H1.

---

# Font Properties

---

If the Font Name or Point Size property for a screen or widget is set to a value other than Default, the generated HTML specifies fonts and sizes in this format: `<div style="font-size:relative-fontsize;font-family:font-family-names ">`

## Font Name

If a screen has its Font Name property set, a `style` tag that specifies this setting is generated for each widget whose Font Name property is set to Default. If a widget has its own Font Name property setting, the HTML contains a `style` tag that specifies the widget's font.

If font-name is supported by the browser, this setting overrides the browser's proportional and fixed font settings.



The font name must map to a font that the browser supports in order to have effect. For maximum flexibility, set the Font Name property to a font alias whose definition in the Web configuration map file `webcmap` specifies one or more fonts. For example, this entry maps Prolifics Helvetica to three comma-delimited sans serif fonts:

```
Prolifics Helvetica=Helvetica,Arial,Geneva
```

When HTML is generated, all Font Property settings that specify Prolifics Helvetica are mapped to `Helvetica,Arial,Geneva` in a `style` tag. Then the HTML is rendered, the browser will use the first font in the list that it supports.

When HTML is being generated by `prorun` or `prodev`, if the Configuration Map file has the `[HTML Fonts]` section, the font aliases specified are used instead of those in the `[Display Fonts]` section.

## Font Size

If a screen has its Point Size property set, a `style` tag that specifies this setting is generated for each widget whose Point Size property is set to Default. If a widget has its own Point Size property setting, the HTML contains a tag that specifies the widget's relative size.

Point Size property settings for screens and widgets are converted into equivalent HTML font sizes, as shown in Table 3-6:

**Table 3-6 HTML Font Sizes**

Point size	Relative font size
6, 7, 8, 9	60%
10, 11, 12	80%
13, 14	100%
15, 16, 17	120%
18, 19, 20	150%
21, 22, 23	200%
24, 26, 28, 36, 48, 72	300%

---

# Application Properties

---

Web applications also have access to the following runtime properties:

`in_web`

Determine whether the application is running as a Web application, so that appropriate actions can be taken. It is possible to create screens and write code that is common to two-tier, three-tier, and Web applications; this property determines the current environment.

Values: `PV_YES/`, `PV_NO`

Constraints: Read-only.

The following section of a JPL procedure would hide the push buttons needed for Web processing in other types of applications:

```
if ( @app()->in_web=PV_NO )
{
  pb_details->hidden=PV_YES
  pb_back->hidden=PV_YES
}
```

The Panther Web Gallery contains a documentation sample entitled Web and GUI Application which uses `in_web`. The Panther Web Gallery is accessible from the Web Application Server at:

`http://server-name/cgi-bin/jwsamp/main`

`previous_form`

The current screen as specified in the cache file. (Refer to [page 6-2](#), “Caching Data.”)

`webid`

Determine the name of the next cache file to be generated by the web application server. (Refer to [page 6-2](#), “Caching Data.”)

# 4 Opening Screens

A Panther Web application can specify to open a screen in several ways:

- Enter the screen name in a URL.
- Create a hyperlink to the screen.
- Open the screen through application processing.

With the first two methods, the URL specified in the browser or hyperlink contains the name of the HTTP server where the Panther screen is located. The HTTP server passes the URL request to Panther using CGI's `GET` method.

Panther then performs any processing, generates the HTML for the screen, and sends the data to the browser. If the user activates a push button on the screen, the browser submits the screen back to the HTTP server. This time the HTTP server uses CGI's `POST` method to relay the submitted screen.

The protocol used to send the screen data is determined by the URL. For many requests on the Internet, the HTTP protocol is used, but to transfer data securely, the HTTPS protocol must be specified.

This chapter describes how to open Panther screens with a URL and which protocol to use in order to transfer the data.

---

# Processing Screen Requests

---

When the URL reaches the HTTP server, the server passes information about the request to the CGI program named in the URL. This information is contained in a series of environment variables. Three variables facilitate data transfer:

- REQUEST\_METHOD
- QUERY\_STRING
- CONTENT\_LENGTH

The value of REQUEST\_METHOD indicates how data in your form is submitted back to the server. Form data can be submitted with two methods: GET and POST. The FORM tag in the HTML document contains a METHOD attribute that indicates which method is used.

If the GET method is used, all form data is passed in the URL. Data input appears as a *name=value* pairs, which are put into CGI's QUERY\_STRING variable. The form's input field name must match the Panther widget or variable name.

Because GET passes all data to the CGI program in a URL, the amount of data that can be passed is limited. For this reason, most forms are submitted back to the server with POST. With POST, the form data appears in standard input and CGI's CONTENT\_LENGTH variable determines the amount of data to process. This is the method that Panther uses.

A sample screen in the Panther Gallery uses the GET method to send data to a search engine. This sample entitled *Search Engine* is accessible from the web application server:

```
http://server-name/cgi-bin/jwsamp/main
```

## Specifying a URL

After a Panther web application is started using the monitor command or started as a Windows service, a screen in the application is typically opened by:

- Entering the screen name in a URL.

- Creating a hyperlink to the screen.

Both methods require the screen's URL. URLs (Universal Resource Locator) use the naming convention developed by the World Wide Web to uniquely identify Web resources. The URL that opens a Panther screen contains this information:

- The protocol to use in accessing the server—for example, `http` or `https`.
- The Internet domain name for the HTTP server where the resource is stored.
- The port number of the HTTP server. If the port number is 80 (the default value) the port number is not displayed in the URL.
- The programs directory that contains the Panther requester program.
- The name of the Panther requester program. The default name for the CGI version is `proweb` (or `proweb.exe`).
- The name of the screen or report.

In the following URL, the HTTP protocol is used to access the server at `prolifics.com`. The CGI directory, `cgi-bin`, contains Panther's CGI requester program, `vidbiz`. This application directory contains the screen entitled `main.scr`:

```
http://prolifics.com/cgi-bin/vidbiz/main.scr
```

## Encoding Parameters in the URL

The GET method lets you add information in a URL beyond the resource name. Because certain ASCII characters have a specific use in a URL, you might need to encode the additional information so that it is correctly interpreted.

To format the URL correctly, apply the following rules:

- A question mark (?) follows the resource name, before any additional parameters.
- A space in the URL string is indicated by a plus sign (+).
- A percent sign (%) precedes any special characters encoded in hexadecimal notation.
- An ampersand (&) delimits each *name=value* pair from the next.

- An equal sign (=) separates the input field name from its value in each *name=value* pair. If nothing has been entered in the input field, the URL string will contain the entry "name=".

## Example

The following URL retrieves the vid.scr screen in the vidbiz application:

```
http://myserver.com/cgi-bin/vidbiz/vid.scr
```

To pass data values in the URL, use the GET method to submit the form. The data in the form's input fields are included in the URL as *name=value* pairs. To map those values to a screen, the name of the input field must match the widget name. The following URL sends the values from the `title_id` and `copy_id` fields to the vid.scr screen:

```
http://myserver.com/cgi-bin/vidbiz/vid.scr?title_id=9&copy_id=2
```

To specify the occurrence number of an input array, place the occurrence number inside a pair of square brackets. The following URL sends the values to the fifth occurrence from a widget of the same name:

```
http://myserver.com/cgi-bin/vidbiz/vid.scr?title_id[5]=9
```

## Encoding ASCII Characters

If the characters used to format the URL appear in the URL for any other purpose, they must be encoded. Table 4-1 lists common special characters and the hexadecimal value for each character.

**Table 4-1 Special characters in URLs**

Special Characters	Hexadecimal Representation
#	%23
/	%2F
?	%3F
=	%3D
&	%26

In addition to special characters, there are several ASCII characters that can only be present in an encoded format. As with the special characters, the ASCII characters must also be preceded by the percent sign (%). Table 4-2 lists these disallowed ASCII characters along with their encoded format.

**Table 4-2 ASCII characters that are disallowed in URLs**

<b>Character</b>	<b>Hexadecimal representation</b>
TAB	%09
SPACE	%20
"	%22
<	%3C
>	%3E
	%5C
^	%5E
{	%7B
}	%7D
[	%5B
]	%5D
`	%60
	%7C
~	%7E

# Transmitting Screens Securely

---

Information over the Internet travels through a network of many computer systems, each with the capacity of intercepting the information. In order to add a layer of security to this system, the SSL (Secure Sockets Layer) protocol was developed. SSL provides the features, such as server authentication and data encryption, that are needed to transfer data securely.

## Setting Screen Properties

To use SSL in a Panther Web application, you must have a secure HTTP server, specify that HTTP server in the application's initialization file in the `HTTPSHOST` variable, and set the `Secure POST` property in Panther screens.

When Panther finds the `Secure POST` property set to `Yes`, that screen is submitted using the HTTPS protocol. Panther continues to use the HTTPS protocol until the application opens a screen that has the `Secure POST` property set to `No`.

## Determining Screen Sequence

Graphical browsers use an icon to indicate whether the HTTPS protocol is in use. For example, Mozilla Firefox displays a lock icon in the address bar when the transmission is secure. However, this icon does not tell how the current screen is to be submitted back to the server; it only shows which protocol was used for the previous submission.

Given this limitation, you might want to build a certain screen sequence into your application surrounding the transmission of secure information. For example, an order entry application has a payment screen where the user enters their credit card information. To transfer this data securely, you might create this screen sequence:

1. After making selections, the user presses a push button to enter payment information.



2. A screen opens explaining that the user is making a secure transmission. The Secure POST property in this screen is set to Yes.
3. The push button on this screen opens the payment screen. The Secure POST property in this screen is also set to Yes.
4. The user enters the payment information and submits this screen back to the server.

By opening an introductory screen whose Secure POST property set to Yes, the user can visually see the secure icon on the payment screen.

## Configuring Your Web Application

A Panther application's initialization file must include the variables for screens that are submitted with the HTTPS protocol, which transmits data to a secure server, and the HTTP protocol, which transmits data to an unsecured server.

### HTTPHOST

Specifies the server for the HTTP protocol. This variable must be set only if both secure and unsecured servers are operational.

### HTTPSHOST

Specifies the server for the HTTPS protocol. This variable must be set in order for the HTTPS protocol to be available.

If the two types of servers are on different hosts, the entries in an `.ini` file might appear as follows:

```
HTTPHOST=apphost.com
HTTPSHOST=safeserve.com
```

**Note:** Port numbers are not necessary if the default port is used—80 for http, 443 for https.

If the two types of servers are on different domains of the same host, the entries in the `.ini` file might appear as follows:

```
HTTPHOST=apphost.myserver.com
HTTPSHOST=safeserve.myserver.com
```

In order to use server caching, both servers must be able to access a common cache directory. Otherwise, the application must use browser caching.



# 5 Web Events

Application processing in Panther occurs by executing a series of events. Panther supports several event hooks that are specific to Web applications. This chapter describes the following event types:

- Web Event Hooks ([page 5-1](#))
- Web Application Events ([page 5-5](#))
- Screen Entry Context Flag—`K_WEBPOST` ([page 5-7](#))
- Web Entry Context Flags ([page 5-9](#))

---

## Web Event Hooks

---

Panther has four Web application event hooks where you can attach JPL procedures. Four events are defined: `web_startup`, `web_enter`, `web_exit`, and `web_shutdown`. When one of these events occurs, Panther searches for a JPL procedure that has the corresponding event name.

`web_startup`

Called when the Panther Web application starts up. This event hook performs start up processing or specifies application-wide settings. You can use a `web_startup` procedure to create global variables, to establish database connections for two-tier applications, or to establish request broker connections for three-tier applications.

```
proc web_startup //2-tier
DBMS ENGINE jdb
DBMS DECLARE jdb_conn1 CONNECTION FOR DATABASE "videobiz"
return

proc web_startup //3-tier
client_init
return
```

### web\_enter

Called after the Panther screen has been opened and the data from the browser has been mapped into the screen. This hook can be used to validate the contents of screens and to verify data before acting upon a request.

When a screen is submitted, values are written to Panther variables which can be used in `web_enter` processing. These variables tell why the screen was submitted, the object ID of the widget involved, and the occurrence number, if applicable. For more information, refer to [page 5-9](#), “Web Entry Context Flags.”

The following `web_enter` procedure from the Panther Gallery specifies a different graphic and hyperlink if the documents are displayed using frames.

```
proc web_enter
if frames=1
{
// If using frames, use jlogomin.gif.

    gallery->active_pixmap = "jlogomin.gif"
    gallery->default_link = ""
}
else
{
// If frames unavailable, use Gallicon.gif and set hyperlink.

    gallery->active_pixmap = "Gallicon.gif"
    gallery->default_link = "main"
}
return
```

### web\_exit

Called just before the HTML generation. This hook offers a last chance to change a screen before

Panther generates the HTML for presentation in the browser. Screen exit processing occurs after the HTML is generated; therefore, screen exit processing cannot affect the generated HTML.

The following `web_exit` procedure unhides the grid containing the address information if the JPL global variable `show_details` is set to `PV_YES`. This variable is created during screen entry and is set to `PV_YES` on POST events.

```

proc screen_entry (data, flags)
if !(flags & K_WEBPOST)

// If this is a GET, create variable
// and set to No
{
    global show_details = PV_NO
}
else // If this is a POST, create variable and set to Yes
{
    global show_details = PV_YES
}
return

proc web_exit ()
if show_details = PV_YES // If show_details set to Yes for
                        // a POST, grid is unhidden
{
    address_grid->hidden = PV_NO
}
else // hide grid
{
    address_grid->hidden = PV_YES
}
return

```

`web_shutdown`

Called when a Panther Web application shuts down. Use this hook to close the database and request broker connections that are created in `web_startup`. You can also use it to perform any other necessary clean up.

Note that this procedure is only called if the application shuts down cleanly, such as with the monitor utility.

```

proc web_shutdown //2-tier
DBMS CLOSE CONNECTION jdb_conn1
return

proc web_shutdown //3-tier
client_exit
return

```

## Specifying Web Event Hooks

When a Web application event occurs, Panther searches for a JPL procedure with the event name and, if found, executes it.

Panther executes the first JPL procedure that it finds with this name, using this search algorithm:

1. If the call is issued from a JPL module, a named procedure in that module.
2. A named procedure in the current screen's module.
3. A named procedure in a public module. If the procedure name exists in more than one public module, Panther uses the procedure in the most recently loaded module.
4. A memory-resident module.
5. A named procedure in the JPL module named in the application variable, `SMINITJPL`. Because Panther automatically publics this module as part of its initialization, the default processing for each of the Web event hooks can be defined here.

Given this search order, a Web application can have multiple JPL procedures for the same event. For example, you can define the `web_enter` procedure and issue the `public` command on it at application startup; this procedure performs default processing for all screens. However, a specific screen can have its own `web_enter` procedure; when this screen is open, Panther finds its `web_enter` procedure first and executes it. The default `web_enter` procedure is executed when subsequent screens open that lack their own `web_enter` procedure.

---

# Web Application Events

---

## Web Application Server Events

After Panther is installed and configured on your system, each Web application server performs the following steps when it is initialized:

1. Loads any JPL modules specified by `SMINITJPL` in `smvars.bin` or in the Web application's initialization file.
2. Calls the JPL procedure `web_startup`.

If the Web application server receives a request to terminate from the monitor utility, it performs the following steps:

1. Executes the JPL procedure `web_shutdown`.
2. Performs normal Panther shutdown processing and exits.

## Screen Events

For each screen request or submission, Panther performs these steps:

1. Restores any open bundles (created by the `send` command) and global JPL variables that come back from the browser or have been cached on the server.
2. Opens the screen by calling `sm_r_form`. This causes the following screen entry processing:
  - Executes the unnamed JPL procedure.
  - Initializes the transaction manager which issues a `START` command.
  - Executes the default screen function.
  - Executes the screen entry function.
  - Merges with the LDB.

- Executes the `AUTO` control string.
3. On a `POST`, restores the contents of fields as specified by data from the browser and from the server cache.
  4. On a `GET`, processes variable assignments from the `QUERY_STRING`.
  5. If a button was pushed, calls `sm_gofield` for that button.
  6. Calls the JPL procedure `web_enter`.
  7. If a button was pushed, calls `sm_ungetkey` for the `NL` logical key.
  8. Invokes the Panther executive to perform normal processing until key input is required. This includes:
    - Execution of the default field function.
    - Execution of the field entry function for the first field.
    - Processing the `NL` key from Step 7.
    - Execution of the application processing defined in the screen.Error messages, instead of requiring a user response, are processed by automatically executing the processing assigned to the default push button.
  9. Calls the JPL procedure `web_exit`.
  10. Generates `HTML` for the screen and sends that data to the browser.
  11. Closes all open windows, which executes normal screen exit processing. Note that this cannot affect the already generated `HTML`.
  12. Deletes appropriately marked JPL variables.
  13. If a C web event hook function was installed by calling `sm_web_set_onevent`, this function is called.



---

# Controlling Entry Processing

---

## Screen Entry Context Flag

`K_WEBPOST` is a context bit flag that is set to true when Panther opens a screen because of a browser's `POST` command. When you press a push button on a browser screen, the data is sent back to the HTTP server with the `POST` method and the `K_WEBPOST` flag is set. If part of that screen's processing is to open another screen on the Panther server, `K_WEBPOST` flag is not set for the second screen. Only after you post the second screen back to the server does the `K_WEBPOST` flag get set.

For example, your application screen selects data from the database on screen entry and displays that data to the user so that the user can make a selection. At this point, Panther performs these actions:

1. Receives the URL request for the screen (using the `GET` method).
2. Opens the screen.
3. Executes screen entry processing which selects the data.
4. Generates HTML for the screen.
5. Sends the data to the browser making the request.

The user then makes a selection and submits the screen back to the server where Panther performs these actions:

1. Receives the screen data (using the `POST` method).
2. Reopens the screen.
3. Reexecutes screen entry processing which reselects the data, wiping out the user's selection.

You can test the `K_WEBPOST` flag so that the screen entry procedure selects data only when a user requests the screen, not when the user submits the screen back to the server on a `POST`. For example:

```
proc scr_entry(scr_name, flags)
if !(flags & K_WEBPOST)
{
    DBMS QUERY SELECT title_id, name, pricecat FROM titles
}
return
```

In the *Panther Gallery*, HTTP Request Method illustrates a screen using `K_WEBPOST`. The *Panther Gallery* is accessible from the Web application server:

`http://server-name/cgi-bin/jwsamp/main`

## Screen Entry Processing and Option Menus

An option menu widget can have its Drop Down Source property set to External Screen so that it gets its data from a database. When a Panther screen has an option menu thus set, its screen entry event is processed twice—when the screen initially opens, and again after it returns from populating the option menu. When the Panther screen opens on a `GET` and then reopens on a `POST`, the additional screen entry event that is triggered by the option menu occurs on both the `GET` and `POST` events.

You can control screen entry processing so that it occurs only on the desired event:

- Set the application variable `EXPHIDE_OPTION` to `OFF_EXPHIDE` so that screen entry events only occur when the screen is explicitly opened.
- Test the `K_EXPOSE` flag. This flag is set to true only if the screen is exposed after another screen is closed. This is what occurs with the option menu initialization.

The following JPL procedure tests the `K_EXPOSE` and `K_WEBPOST` so that screen entry processing occurs only on `GET` and `POST` events; this processing is avoided when screen entry occurs after option menu initialization:

```
proc scr_entry (name, flags)
if ( !(flags & K_WEBPOST) && !(flags & K_EXPOSE) )
{
    // Processing occurs only on a GET event.
}
if ( (flags & K_WEBPOST) && !(flags & K_EXPOSE) )
{
    // Processing occurs only on a POST event.
}
```

## Web Entry Context Flags

When a screen is submitted, values are written to the following variables for use in `web_enter` procedures:

`@web_action`

This variable tells why the screen was submitted. The values are:

`PV_SUBMIT`

General submit (using JavaScript submit or by pressing Enter in a text field).

`PV_BUTTON_PRESS`

Push button was pressed. `PV_INSERT`, `PV_APPEND`, `PV_DELETE`,

`PV_SCROLL_UP`, `PV_SCROLL_DOWN`, `PV_SCROLL_TOP`, `PV_SCROLL_BOTTOM`

The corresponding grid push button was pressed.

`@web_action_widget`

For `PV_BUTTON_PRESS`, this will contain the object ID of the button that was pressed.

For the grid push buttons, this will contain the object ID of the grid being operated on.

Otherwise, this will contain `PR_NULL_OBJID`.

`@web_action_occurrence`

For `PV_BUTTON_PRESS`, this will contain the occurrence number of the button pressed.

For `PV_INSERT`, `PV_APPEND` and `PV_DELETE`, this will contain the occurrence being operated upon.

For `PV_SCROLL_UP`, `PV_SCROLL_DOWN`, `PV_SCROLL_TOP` and `PV_SCROLL_BOTTOM`, this will contain the occurrence number placed in the first onscreen element of the grid being scrolled.



# 6 Preserving Application State

Unlike other platforms, a Panther application that is running on the Web does not completely control the sequence of its own screens. The inherent Web behavior is for an application to transmit a screen to the browser, break the connection and terminate. The application restarts when the screen is submitted back to the HTTP server, retaining no memory of the previous transmission. Also, the user can disrupt the application's screen sequence by using browser controls, such as the Back push button, to view screens outside the control of the Panther application. The user can also shut down the browser program and never submit the screen back to the server.

However, a typical database application requires continuity across multiple exchanges of data. For example, an application accepts a user name on the first HTML form. After receiving the user name, the application selects database information for that user name and returns it in another HTML form to the browser. When the browser submits the second form with updated information, the application might need the user name from the first form to save the changes. Because the HTTP server is stateless, it has no information about the previous form.

The following sections discuss features and options that let you save the state of a Panther Web application.

---

# Caching Data

---

In order to pick up where it left off, Panther caches data for a screen before it generates its HTML and transmits it to the browser. A screen's data needs to be cached differently depending whether the `METHOD` attribute is set to `POST` or `GET`.

## Posting Screens Back to the Server

By default, Panther generates screens with the `METHOD` attribute set to `POST`. With this method, each screen in the application has a push button which submits the screen back to the web application server.

If the screen is posted back to the server before the cache expires, Panther finds the screen's data cache, restores it, and executes the next step in the application. An application can cache screen data either on the browser or on the server; the application specifies its caching method through the initialization option `BrowserData`, whose default is set to enable server caching.

The user of a Web application can interact with the screen in the following ways. Your data caching options determine the application's behavior in these situations:

The user fills in the screen and posts it without delay or side trips to other screens. Between receiving and posting the screen, the user redisplay another Panther screen through non-Panther controls such as the browser's Back and Forward push buttons.

The user posts the screen, then redisplay it through browser controls and posts it again. (Some browsers detect this action and prompt the user for confirmation of the repost.)The user posts the screen but its data cache has expired. The user quits the browser program and never posts the screen.

If browser caching is enabled, cached data is encoded in the HTML document that is generated for the Panther screen and sent to the browser. When the screen is posted, the browser returns the encoded data cache along with user-entered data. Browser caching must be used if the https server and the http server are unable to share a common cache file directory.

Browser caching offers these benefits:

- The user can always post the screen. The post never expires and never fails.
- No server disk space is devoted to cache files.

However, browser caching can (depending on your application) greatly increase the size of the HTML document and the amount of time it takes to download the screen.

If server caching is enabled, cached data is maintained in the cache file directory when a screen is transmitted to the browser. The cache file directory is specified by the initialization option `CacheDirectory`. When a screen is posted back to the server, the data is retrieved from the cache file and mapped into the Panther screen along with user entries.

If the initialization option `RetainCacheFiles` is set to 0 (no), Panther automatically deletes cache files after the corresponding screen is posted back to the server.

If the initialization option `RetainCacheFiles` is set to 1 (yes), Panther retains the cache files until its lifetime exceeds the time limit specified by the `ExpireTime` initialization option. By default, `ExpireTime` is set to 120 (two hours). If a screen's cache file is retained, the user can redisplay the screen with the browser's Back push button and repost it.

All server cache files can be retained indefinitely if `RetainCacheFiles` is set to 1 (yes) and `ExpireTime` is set to a negative number.

You can also use the `monitor` utility to delete cache files manually.

If the application lets users go to other Web resources through action list boxes or calls to `sm_web_invoke_url`, it must retain cache files so the user can return to the Panther Web application.

If the cache file is unavailable when the screen is posted, Panther transmits the `smrepost.scr` screen, which reports the error that the cache file is unavailable. This behavior can be changed with global variable `@web_posted_screen`, which contains the name of the most recently posted screen.

One use for this variable is in `smrepost.scr`. As part of the screen entry procedure for this screen, you can test for the value of `@web_posted_screen` and specify the application's behavior based on its value.

## Getting Screens from the Server

Instead of submitting screens back to the server with a `POST`, you can use a `GET` if you retain the current state information on the server and program the screen to store the cache filename and ask for it on the `GET` as part of the URL.

This allows you to use hyperlinks to retrieve screens, to have state information available in a series of HTML templates, and for all screens in a frameset to share the same state information.

In order to use this method, the web initialization file must have `EnableWebid` set to 1. Just before the HTML is generated for the screen, obtain the name of the next cache file to be generated using the `webid` application property and store it in a variable. The following JPL command stores the name in a variable called `my_cache`.

```
my_cache=@app()->webid
```

To retrieve the next screen using a `GET`, use `@webid` in the URL:

```
http://myserver.com/cgi-bin/myapp/next.scr?@webid=my_cache
```

To use this feature, `BrowserData` must be set to 0, which allows cache files to be generated. It is also recommended that `RetainCacheFiles` be set to 1, especially if using multiple screens in a frameset.

Since there are security issues using `@webid` in the URL which would allow non-application users to retrieve the state information, you can use the `previous_form` application property to store the name of the last screen that was accessed and then check it on screen entry. The following screen entry procedure calls a security error screen if the previous screen value is not correct:

```
if (@app()->previous_form != "first.scr")
{
    call sm_jform("security_error.scr")
}
```

## Cached Data

Panther automatically caches the following data about an application's state:

- Scrolling widgets—If the user posts a screen by pressing one of the VCR scroll buttons, the Web application server can get the next page of rows because it



remembers the scroll state of the widgets—that is, it remembers which occurrences were visible when the HTML was generated.

- Transaction manager before-image data and mode—The transaction manager can generate SQL statements properly by accessing the data's before image. Also, Panther must preserve the transaction mode: if a screen in transaction manager update mode is sent to a browser, Panther must recognize that the screen is in update mode when it is posted. Otherwise, the transaction manager's save command (`sm_tm_command("SAVE")`) cannot know what action to perform.

If no cached data is available for a screen, Panther posts an error message. Regardless of whether data is cached, Panther always updates the `@web_posted_screen` variable after each screen posting to contain the name of the posted screen. A screen's entry procedure can use this variable to find out which screen preceded it and thereby determine the application's behavior.

Several objects that are cached can be used to save the application's state—hidden widgets, send bundles, and context globals. These are discussed in the following sections.

When Panther builds the cache file before HTML generation, it saves the values of all hidden widgets on the active screen. When Panther reopens the screen on a subsequent `POST`, it updates the screen's hidden widgets with their cache values. In fact, all widgets are updated at once: visible widgets are updated with their browser-supplied values; hidden widgets are updated with their cache values. This occurs after normal screen entry processing and before the `web_enter` event executes.

When Panther builds the cache file before HTML generation, it saves the values of bundles created by the send command. A bundle is accessible if no receive command has been issued on it, or the receive command was issued with the keep option. When Panther opens the cache on a subsequent `POST`, it restores all bundles before it opens the screen. This occurs before the screen opens so that bundle data is available to the screen's unnamed JPL procedure or any later event.

For example, an application's first screen might put the contents of an address array into the `addr` bundle:

```
send bundle "addr" data address
```

When a subsequent request needs the address values, it executes this receive command and puts the bundle into the `whereami` field:

```
receive bundle "addr" data whereami
```

User-specific information can be saved in JPL context global variables. Each context global is private to a single user of a Web application server; to create one, call `sm_web_save_global` on a JPL global variable previously created with the global command. When Panther builds the cache file before HTML generation, it saves the values of all context globals. On opening the cache on a later `POST`, Panther recreates the context globals and initializes them to the cached values. This occurs before the screen is opened, so all context globals are available to the screen's unnamed JPL procedure or later events.

Panther saves context globals in subsequent cache files until the application calls `sm_web_unsave_global` or `sm_web_unsave_all_globals`, which remove context globals one at a time or all at once.

For example, the HTML form that accepts the user name might execute the following during processing for the `POST` event:

```
proc enter (screen, status)
if ( status & K_WEBPOST)
{
// Create a JPL global
global current_user(31)

// Set it to the value of the widget called user.
current_user = user
// Make current_user a context global so it is
// saved between transmissions.
call sm_web_save_global("current_user")
}
```

After this procedure executes, requests can refer to the `current_user` global. For example, if a subsequent request selects database values and a later request saves changes to them, the save screen might include the user name in its status messages:

```
proc save_changes
call sm_tm_command("save")
if ( sm_tm_inquire(TM_STATUS) == 0 )
{
message = "Thank you, " ## current_user \
## ". Your changes have been saved."
}
else
{
message = "Sorry, " ## current_user \
## ". Your changes could not be saved."
}
```

```
msg emsg message  
return
```

For more about JPL variables, refer to “Variables” [on page 19-24](#) in *Application Development Guide*; also to the global command.

---

## Saving State Data in Cookies

---

Cookies are pieces of information from the browser side of a connection. After a cookie is set by an HTML document, it is stored on the browser and can be retrieved when that browser contacts the same HTTP server.

Cookies are best used for simple, persistent, client state information, such as a user ID, the date, or the number of times the client visits a specific URL. If the cookie specification includes an expiration date, this information is saved on the browser and is available in subsequent browser sessions by the same user.

---

## Unpreserved State Information

---

Some information on an application's state is not automatically preserved. If the application requires this information, it should save it with one of the methods described earlier.

### LDB

No LDB (local data block) data is stored in the cache. Any changes made to the LDB are known to all users of the Web application server.

## Window Stack

If other screens are on the window stack when HTML is generated, no information about these screens is saved in the cache.

For example, consider an application where screen A's submit button opens screen B and screen B contains the following screen entry function:

```
proc enter (screen, status)

// Add user name from screen a to title.
title = "Welcome " ## a!user_name
```

This entry procedure fails when screen B is submitted because screen A no longer exists on the window stack. If the value is not actually needed on post, the problem is corrected by testing for `K_WEBPOST`:

```
proc enter (screen, status)

if !( status & K_WEBPOST )
{
    // Add user name from screen a to title.
    title = "Welcome " ## a!user_name
}
```

If the value from screen A is actually needed during the post of screen B, the value should be saved in a context global or as send data.

## Property Changes

The cache does not save any property changes in by the application. For example, a screen contains a grid with 200 onscreen rows and a push button that executes this procedure:

```
proc get_data

call sm_tm_command("SELECT")
if (grid->num_occurrences < grid->onscreen_rows)
{
    grid->onscreen_rows = grid->num_occurrences
}
return
```

The screen also has another push button that calls this procedure:

```
proc save_changes
call sm_tm_command("SAVE")
```

When the screen opens to execute the `get_data` button, it executes a select and sets the number of grid rows to the number of rows found by transaction manager. For example, assume that 50 rows are found. When the user later presses the `save_changes` button, the screen opens and the save command executes. However, when the HTML is generated the second time, the grid shows 200 rows, not 50. To maintain the grid size for a subsequent post, apply the property change each time HTML is generated: save the desired grid size between transmissions with a context global or send bundle; or move the property change to an event that is processed for all events:

```
proc web_enter
if (grid->num_occurrences < grid->onscreen_rows)
{
    grid->onscreen_rows = grid->num_occurrences
}
return
```



# 7 JPL Globals in Web Applications

Web applications can set JPL global variables to store data at three levels: application, individual users, and requests.

A sample screen in the Panther Gallery entitled *JPL Globals* demonstrates an application using different types of global variables; it is accessible from the Web application server:

`http://server-name/cgi-bin/jwsamp/main`

---

## Application Globals

---

An application global is shared by all users of a `jserver`. To create an application global, issue the JPL global command before the `web_startup` event completes. You can declare application globals in the following areas:

- `SMINITJPL` file
- `web_startup` procedure

Because changes in an application global are visible to all users, it is unusual to change an application global's value after `web_startup`. Instead, application globals are best used to supply values that remain constant for the duration of the application. For example, `PV_YES` and `PV_NO` are application globals whose values should remain unchanged.

---

## Context Globals

---

A context global variable is private to a single user of a jserver. Its value is preserved between transmissions of the browser and HTTP server. A context global is created by creating a JPL global variable with the `global` command, and then calling `sm_web_save_global` on that variable. For example:

```
proc make_jpl_global
global my_global
call sm_web_save_global("my_global")
return
```

Panther automatically maintains a context global and its value in the application's cache file. Because each set of context globals is specific to a given user, you can use them to save user-specific information such as ID, preferences, or start time.

A context global is maintained until `sm_web_unsave_global` is called for that global, or all context globals are removed by `sm_web_unsave_all_globals`.

If the `global` command executes a second time, it overwrites the global's previous value. If you execute the `global` command in the unnamed JPL procedure or during screen entry, also test whether the screen is being opened for a `GET` event, because the screen is then reopened on a `POST` event. You can test this by using the `K_WEBPOST` flag or the CGI variable `@cgi_request_method`.

For example, the following screen entry procedure creates a JPL global variable for the current user on a `GET` event:

```
proc enter( screen, status )

if !(status & K_WEBPOST)
{
```



```
        //Create a JPL global
global current_user(31)
        //Make current_user a context global so it
        //saved between transmissions

    call sm_web_save_global( "current_user" )
}
```

Subsequent requests can refer to the `current_user` global. For example, if a later request saves changes to previously selected values, the save screen can use the user name in its status message:

```
proc save_changes

call sm_tm_command( "SAVE" )
if (sm_tm_inquire(TM_STATUS)==0)
{
    message="Thank You, "##current_user\
        ##" Your changes have been saved."
}
else
{
    message="Sorry, "##current_user\
        ##" Your changes could not be saved."
}
msg emsg message
return
```

---

## Transient Global Variables

---

A transient global variable is one that is created after the `web_startup` event is complete and is not added to the cache with `sm_web_save_global`. A transient global exists for a single request and is destroyed during exit processing after HTML generation. A transient global is useful when a `POST` event opens multiple screens that need to share a data value among themselves but not with later requests.



# 8 Customizing HTML Generation

Application screens are stored in Panther's binary format. When an HTTP server requests a screen, Panther opens that screen, performs the processing specified for it, generates HTML or downloads Java for the screen, and returns the data to the HTTP server. This chapter describes how the HTML for the screen can be customized for your application.

A standard HTML document is composed of two main sections:

- The `HEAD` element containing information about the document itself.
- The `BODY` element containing the document content.

Each of these main elements can be composed of elements, each with its own use and format. All elements are identified by a tag or pair of tags. Each tag is enclosed by angle brackets—for example, `<HR>`. All elements begin with a start tag; and most elements also have an end tag, which appears at the element's end. To differentiate the two tags, the end tag is prepended with a slash—for example, `</BODY>`.

When Panther generates HTML for a screen, it creates an HTML `FORM` element to contain the screen's content. `FORM` elements, according to HTML standards, are located within a `BODY` element.

Within the `FORM` element, Panther creates a series of tables containing `INPUT` elements corresponding to the widgets. The tables maintain widget positions in the HTML document. Otherwise, the structure of the screen is lost because HTML has no other method to define positioning information.

Although Panther automatically translates each object in a Panther screen into its appropriate HTML element, you can customize the HTML generation. This chapter describes how to perform these tasks:

- Add HTML markup for screens and widgets in the Custom HTML properties.
- Use a pre-existing HTML document to provide the HTML structure.

In addition, some HTML elements have no Panther equivalent. This chapter describes how to create those elements in a Web application.

- Create common HTML elements, such as hyperlinks and headings.
- Use graphics and image maps in your Web application.
- Use Panther screens within HTML frames.
- Use cookies to retrieve browser data.
- Embed Java applets in the HTML document.

---

## Setting Custom HTML Properties

---

The properties under Custom HTML category let you add elements or attributes to the HTML that Panther generates for your screen. Each of these properties allows free-form text entry; their contents are not validated by Panther.

### Screen Custom HTML Properties

A Panther screen has the following Custom HTML properties:

#### Head Markup

The Head Markup property specifies HTML to insert within the <HEAD> section of the HTML document. Generally, this section includes information about the document itself, and except for the document title, this information is not displayed in the Web browser.

### Body Attributes

The Body Attributes property allows you to add to, or modify, the attributes within the `BODY` element that Panther generates for a screen. Within the `BODY` element are the elements and tags containing the content of the HTML document. You can specify either standalone attributes or *name=value* pairs.

There are attributes of the `BODY` element which allow you to specify the background color for the display window (`BGCOLOR`), the color for text in the document (`TEXT`), the color for unvisited hyperlinks (`LINK`), and the color for visited hyperlinks (`VLINK`). To specify one of these attributes, include the attribute name and its value. For example, the following entry displays blue text:

```
TEXT=#00ff00
```

For reports, this property defines a background. The following entry uses the image `plaster.gif` as a background:

```
BACKGROUND=plaster.gif
```

For screens, Panther automatically generates the `BACKGROUND` attribute using the entry in the Wallpaper Pixmap property.

### Form Attributes

The Form Attributes property lets you add to or modify the attributes within the `FORM` element that Panther generates for a screen. The `FORM` element contains the content of your Panther screen. You can specify either standalone attributes or *name=value* pairs.

Panther automatically generates the `METHOD` and `ACTION` attributes for this element. The `METHOD` attribute specifies which CGI method, `GET` or `POST`, is used to send information to the server. The `ACTION` attribute specifies the URL to receive the form content.

### Stylesheet Source

Specify the style sheet to use for Dynamic HTML generation, if available. For more information, refer to [page 8-19](#), “Using Style Sheets.”

## Widget Custom HTML Properties

A Panther widget can have the following Custom HTML properties:

#### Attributes

Additions or changes to the HTML attributes within the INPUT element that Panther generates for a widget. You can specify either standalone attributes or *name=value* pairs. For example, the following entry for a dynamic label which displays an image specifies the thickness of the border in pixels:

```
BORDER=2
```

#### Link Attributes

Additions or changes to the HTML attributes that Panther generates for a hyperlink. You can specify either standalone attributes or *name=value* pairs. This property is only displayed when there is a value in the Default Link property.

#### Prefix Markup

HTML to insert immediately before Panther generates its HTML for the widget. You can use this property to create HTML headings. For example, to create a level 2 heading, set a label's Prefix Markup property to the start tag `<H2>` and its Suffix Markup property to the end tag `</H2>`.

#### Suffix Markup

HTML to insert immediately after Panther generates its HTML for the widget—typically, end tags for elements defined in the Prefix Markup property.

---

## Using HTML Templates

---

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. The document provides the basic HTML structure instead of using Panther to generate that structure.

In order to use an HTML template:

- Set the screen's HTML Template property to the path and name of the HTML document.

- Insert HTML template tags into the HTML document as needed.

You can use HTML templates to perform these tasks:

- Add database values to a pre-existing HTML document.
- Convert a pre-existing static HTML document to a form.
- Submit a form using the `GET` method to a CGI program other than Panther.

## HTML Template Tags

HTML template tags are specified by enclosing the tag in a set of double curly braces. For example, the following tag places the value from the `title_id` widget in the HTML element `id`:

```
<INPUT TYPE=text NAME=id VALUE="{{title_id}}" SIZE=10>
```

The following tags are available for use in a Web application:

```
{{emit:object}}
```

Generates the HTML that Panther would normally output for the specified object. If *object* is a box or grid, all of its contents are generated. If *object* is a field, only its first element is generated. To generate particular elements, use `{{emit:object[[n]]}}`.

```
{{eval:statement}}
```

Process a simple JPL statement. *statement* in this context can contain assignments.

```
{{form:info}}
```

Interpolates the hidden data which is needed to submit the form. Place this value in the template after the `<FORM>` tag (or `{{form:tag}}`). In order to cache the application data, this tag must be used in the HTML template.

```
{{form:messages}}
```

Outputs any messages in the generated HTML. If there are any messages, they will be emitted within `<div class="sm_message_text">..</div>` tags. New in Panther 5.40.

```
{{form:output}}
```

Outputs the entire form in the HTML format Panther would normally use. The HTML is contained within a set of `<TABLE>...</TABLE>` tags.

`{{form:script}}`

Generates the JavaScript procedures based on the edits and validations of the widgets used on the form. Place this tag directly before the closing form tag `</FORM>`, following any widgets using the JavaScript validation functions—such as the functions for the Yes/No or Digits Only properties.

**Note:** The values for the `onLoad` and `onUnload` screen events, which must be part of the `<BODY>` tag, must be specified in the HTML template. These JavaScript event specifications are not generated with this tag.

`{{form:tag}}`

Generates the start `<FORM>` tag with `ACTION` and JavaScript attributes. You must specify the corresponding end `</FORM>` tag in the HTML template.

`{{if:condition}}`  
`{{else:}}`  
`{{elseif:condition}}`  
`{{while:condition}}`  
`{{break:}}`  
`{{continue:}}`  
`{{next:}}`  
`{{end:}}`

Perform conditional processing based on a JPL boolean expression. Examples of conditions are `i < 7` and `i != 0`.

`{{break:}}` exits a while block.

`{{continue:}}` and `{{next:}}` go to the next iteration of a while block.

`{{end:}}` terminates an if or a while block.

`{{include:filename}}`

Include the specified file. The search path is the same as for the HTML template itself.

`{{include:value:expression}}`

Evaluate *expression* and use the value as the name of the file to include. This would be used, for example, if a widget contains the name of the file. The file search path is the same as for the HTML template itself.

`{{raw:variable}}`

Generates the value corresponding to the specified variable. This is similar to `{{value:variable}}` except that characters like less than (`<`) that are part of HTML syntax are not converted to escape sequences. New in Panther 4.29.



```
{{value:variable}}
```

Generates the value corresponding to the specified variable. This is the same as `{{variable}}`. If *variable* has multiple occurrences, only the current occurrence is output unless the occurrence is specified. The current occurrence is the first one except for scrolling fields that have been scrolled.

```
{{ws:off}}
```

```
{{ws:on}}
```

Controls whether whitespace characters (spaces, tabs, newlines, etc.) from the template are to be included in the generated HTML. `{{ws:on}}` is the default.

## HTML Template Document

The HTML template must contain a set of HTML tags, a set of BODY tags, and a set of FORM tags. The start FORM tag can be specified using `{{form:tag}}` or `<FORM>`; the end tag must be specified as `</FORM>`.

The following HTML template, which illustrates the order of the tags, outputs the entire Panther screen in the generated HTML:

```
<HTML>
<BODY>
<H2>Film Titles</H2>
<HR>
{{form:tag}}
{{form:info}}
{{form:output}}
{{form:script}}
</FORM>
</BODY>
</HTML>
```

## Conditional Processing

Conditional processing can be specified using the following tags:

```
{{while:condition}}
{{if:condition}}
{{else:}}
{{elseif:condition}}
{{end:}}
{{eval:statement}}
```

Each block of processing must begin and end in the same file.

In order to prevent infinite loops on the jserver, two application properties are provided, `html_max_loop` and `html_max_nest`:

- `html_max_loop` is a limit on the number of `while` loop iterations that will be permitted before the template processor aborts; the default setting is 1000.
- `html_max_nest` is a limit on the number of nesting levels. Each `if`, `while` and `include` counts as one level. The default setting is 20.

Setting `html_max_loop` or `html_max_nest` to zero removes the limit.

As an example, if field `myhtml` contains HTML you want to put in the generated HTML, you might include the following in your template file:

```
{{eval:@app()->html_max_loop = myhtml->num_occurrences}}
{{eval:vars i = 1}}
{{while:i <= myhtml->num_occurrences}}
  {{raw:myhtml[i]}}
  {{eval:i = i + 1}}
{{end:}}
```

## Passing Database Values

To illustrate passing database values using an HTML template, Figure 8-1 shows a Panther screen as it appears in the editor:

The screenshot shows a web browser window titled "titleact". The form contains the following fields and values:

- Title\_id: 82
- Name: Room With A View, A
- Director: James Ivory
- Rating\_code: PG
- Release\_date: 1985
- Film\_minutes: 110
- Pricecat: G
- Pricecat\_dscr: General
- Genre\_code: DRAM

Below the form is a section titled "Actor Information" containing a table:

Actor_id	First_name	Last_name	Role
84	Maggie	Smith	Charlotte Bartlett
202	Helen Bonham	Carter	Lucy Honeychurch
229	Denholm	Elliott	Mr. Emerson

At the bottom of the form are three buttons: "Select Titles...", "More Titles...", and "More Actors..."

**Figure 8-1** The widgets contain values fetched from the database.

This screen's HTML Template property is set to `vidname.htm`. This HTML document contains the following tags:

```
<HTML>
<HEAD>
</HEAD>
<BODY onLoad="window.status =
    ('Displays film title information')">
  {{form:tag}}
  {{form:info}}
  <HR>
  <H2>Film Titles for {{value:username}}</H2>
  {{emit:title_id[[1]]}}
  {{emit:name[[1]]}}
  <BR>
  {{emit:title_id[[2]]}}
  {{emit:name[[2]]}}
  <BR>
  {{emit:title_id[[3]]}}
  {{emit:name[[3]]}}
  {{form:script}}
</FORM>
<A HREF="http://myserver/cgi-bin/proweb/vidlist.scr"
  Start a New Search</A>
</BODY>
</HTML>
```

For each of the elements inside a double set of curly braces, the value is inserted at runtime. Consequently, the browser displays the HTML document as follows:

---

**Title Information**

32	Room With A View, A	DRAM	G
----	---------------------	------	---

[Go Back to Titles List](#)

**Figure 8-2** The HTML document in the Web browser contains the database values.

## Submitting a Form

The form can be submitted back to your Panther Web application or, if another GET method is specified in the HTML template, to a non-Panther Web application.

To submit the form back to the Panther Web application, the HTML template must contain the `{{form:info}}` tag. This maintains the cache data for a Panther screen utilizing an HTML template and updates the template dynamically to associate the cache file with it.

To submit the form to a non-Panther Web application, you must specify the `<FORM>` tag with the appropriate GET method.

The HTML document receives two values from the Panther application:

- The value of `search_str`, used to initialize the HTML document's search field:

```
<INPUT NAME=q SIZE=55 MAXLENGTH=200  
VALUE="{{search_str}}">
```

- The value of `@cgi_http_referer`. This value is specified as the `HREF` destination in an anchor on the HTML document; it lets the user return to the Panther Web application. Of course, this can be replaced with a hard-coded URL:

```
<A HREF="{{@cgi_http_referer}}">Film Info</A>
```

---

# Using Hyperlinks

---

Hypertext links enable users to move from one HTML document to another. You can set hyperlinks in a Panther Web application in several ways, as shown in following sections.

To view different types of hyperlinks in Panther, refer to *Hyperlinks* in the Panther Gallery accessible from the Web application server.

## Creating Hyperlinks

### Setting the Default Link Property

Designate a dynamic label or graph widget to act as a hyperlink by setting its Default Link property to the desired URL, either in the editor or at runtime:

- In the editor, set the label's Default Link property (under Web Options) to the desired hyperlink's URL. For example:

```
http://prolifics.com
```

- Set the label's `default_link` property at runtime. For example:

```
home_lnk->link="http\://prolifics.com"
```

**Note:** To avoid JPL colon expansion, prefix any colon in the URL string with a backslash.

To specify attributes for the hyperlink, use the Link Attributes property.

### Setting the Item Link Property

For arrays, you can specify a different URL location for each occurrence of the array using the Item Link property.

The following JPL procedure specifies the Item Link property for each occurrence in an array by building on the value in the Default Link property and using the occurrence number to make each value unique:

```
proc makelink()  
{  
  vars i  
  for i = 1 while i <= arraydoc->max_occurrences  
  {  
    arraydoc[i]->item_link = \  
      arraydoc->default_link ## doc[i] ##".html"  
  }  
}
```

### Calling `sm_web_invoke_url`

Call `sm_web_invoke_url` to invoke a hyperlink without awaiting user action:

```
call sm_web_invoke_url("http://prolifics.com")
```

`sm_web_invoke_url` immediately stops Panther processing and generates no HTML for the screen. Instead, a request is sent to the browser to go to the specified URL.

### Placing an Action List Box Inside a Grid

If a list box is in a grid widget and its Listbox Type property is set to Action, the values appear as hyperlinks. The screen is submitted back to the server when the user selects one of these hyperlinks.

If Listbox Type is set to Select Any, the values are converted to columns of radio buttons or check boxes:

- Radio buttons if the group's Number of Selections property is set to 0 or 1 or 1
- Check boxes if the group's Number of Selections property is set to Any.

## Using Hyperlinks in Reports

Since widgets in reports can have link and image properties, you can make the web-deployed report interactive by using links to:

- Move within a report.

- Display another web-based document.
- Invoke another report.

By invoking another report, you can obtain a “drill down” effect, generating a detail report for any item in the original report. For example, if a main report generates a list of customers, each customer's last name can be a link to a detail report. This “drill down” speeds access to report data by displaying only the detail data that the user wishes to see.

To provide this functionality, a widget in the main report (in this example, `last_name`) has its `Link` property set to the URL that invokes the detail report. By setting this property at runtime through a JPL procedure called from the main report's `Detail` node, the appropriate customer ID can be inserted for the detail report's invocation.

For more information on invoking reports using a URL, refer to “How to Invoke a Report From a URL” [on page 9-9](#) in *Reports*.

---

## Setting Target Windows

---

When you invoke a document from a hyperlink, it typically appears in the same browser window as the previous document. Several screen and widget properties are available that let you assign names to browser windows—or targets—and specify which one displays a document. If the window does not exist, the browser creates it and loads the document in it.

Targets can also be used with framesets that let the user view multiple documents in the same browser window. The frameset definition specifies the number of frames inside a browser window and the target name for each frame.

You can specify target windows for individual hyperlinks, all hyperlinks on a screen, or for a specific screen:

## Setting the Window for a Specific Hyperlink

Select the dynamic label that acts as a hyperlink and set its Target property to the name of the browser window. For example, if the Target property is set to `window2`, the hyperlink is displayed in the browser window `window2`. If this window does not exist, the browser program creates it.

**Note:** The Target property is accessible only if the Default Link property is filled.

## Setting the Default Window for All Screen Hyperlinks

Set the screen's Target Default property to a browser window name. This inserts the `<BASE TARGET>` tag in the HEAD section of the generated HTML. All hyperlinks on the screen whose Target property is empty use this window.

For example, if the screen `test.scr` has Target Default set to `window2`, then each of the hyperlinks on that screen whose Target property is empty display in the same browser window `window2`.

## Setting the Window for a Screen

The screen's Display Window property determines which window displays this screen, no matter where it is invoked. For example, if the screen `test.scr` has Display Window set to `content`, the screen is always displayed in the browser window `content`.

**Note:** If the screen is invoked by a hyperlink that has its target window set, either through the Target Default or Target property, the hyperlink specification overrules the screen's own Display Window property.

---

# Specifying the Browser's Title Bar

---

Using a descriptive name in the browser's title bar provides users with an easy reference for the current document:



1. Set focus to the screen. The Properties window will display the screen properties.
2. Under the Identity category, select the Title subproperty.
3. Enter the text that you want to appear in the browser's title bar.

---

## Using Graphics

---

Most Web browsers can display graphics files that are in GIF or JPEG formats. Panther supports both formats in a number of widget types. Graphics can be included for illustrative purposes only; or they can be used as image maps that enable navigation to other Web resources.

Web browsers vary in their support for graphic formats. You can test whether a specific format is available with the CGI variable `@cgi_http_accept`.

The graphics files can be located in relation to the application directory or in the directory named in the ImageDir setting in the application's initialization file.

To include graphics in a Web application:

1. Create a static label, dynamic label, push button, radio button, or check box widget on your screen.
2. (Optional) Under Identity in the Label property, assign the widget a name.
3. Under Format/Display in the Active Pixmap property, enter the name of the graphics file.

The default border setting for the graphic is `BORDER=0` so that a border is not displayed. To give a graphic a border, enter the border's pixel size in the Attributes property. In this example, the Attributes property sets a border to 2 pixels:

```
BORDER=2
```

## Setting Graphics Size

If Keep Image Size is set to Yes (the default), Panther uses the GUI height and width of the graphic when it positions widgets in the generated HTML. If set to No, it positions the graphic on a single line in the HTML. You can usually leave this property set to Yes.

If Panther cannot determine the graphic's size at runtime, it supplies a default size of 32 x 32 pixels to the browser. The size of the graphic can also be set in the Attributes property.

## Graph Widgets

Graphs and charts created with the graph widget are available in Web applications. The graph can also be a hyperlink so the Link and Link Attribute properties are available for this widget type.

## Image Maps

An image map is a navigational tool that can take users from one location on the Web to another, or to another location within the same site. The coordinates of a graphic are used to divide the graphic into separate sections. Each section is assigned a hyperlink; users can click on it in order to go to a Web resource.

A Web application can include server-side image map files. An image map file contains the coordinates for each section and its hyperlink. Image map file formats vary according to the program that is used to process the image map. The HTTP server decides which program processes the image map and where the image map file must be located.

To include a server-side image map:

1. Create a dynamic label widget.
2. Under Format/Display in the widget's Active Pixmap property, enter the name of the graphics file to display in the browser.
3. In the widget's Image Map property (under Web Options), enter the URL of the server-side image map file that contains the hyperlinks and coordinates.

In the Panther Gallery, *Graphics* illustrates different types of graphic formats and an image map. The Panther Gallery is accessible from the Web application server:

```
http://server-name/cgi-bin/jwsamp/main
```

## Creating Image Maps in JPL

You can create a client-side image map in JPL using the `sm_web_invoke_url` function. Two JPL globals, `@web_image_click_x` and `@web_image_click_y`, contain the X and Y coordinates of the user's mouse click.

To include a server-side image map:

1. Create a push button.
2. Under Format/Display in the widget's Active Pixmap property, enter the name of the graphics file to display in the browser.
3. Under Validation in the JPL validation property, enter the name of the JPL procedure. In this example, it is `goto_dept`.
4. With the screen having focus, under Focus in the JPL Procedures property, enter a JPL procedure which finds and processes the coordinates. The following sample procedure goes to a different URL based on the coordinates.

```
proc goto_dept
if (@web_image_click_x > 1 && < 125) && \
    (@web_image_click_y > 1 && < 110)

    call sm_web_invoke_url("http://myhost/top.html")
    .
    .
    .
return
```

## Loading Graphics at Runtime

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.

For local intranets, only specify the sub-directory name; for the Internet, specify the HTTP protocol, followed by the domain and the sub-directory name. For example, to retrieve the graphic `my_logo.gif`, the following setting for a local intranet:

```
ImageDir=my_app
```

results in the following HTML, prepending the name of the current machine:

```
<IMG SRC="http://currentMachine/my_app/my_logo.gif">
```

However, if the HTTP protocol is specified:

```
ImageDir=http://prolifics.com/my_app
```

results in the following HTML:

```
<IMG SRC="http://prolifics.com/my_app/my_logo.gif">
```

---

## Using the FRAME Extension

---

Frames allow a browser window to be divided up into several independent subwindows. Each subwindow can display a different Web resource. Each region, or frame, has several features:

- An individual URL so it can load information independently of other frames on the page.
- A `NAME` attribute so other URLs can target it.
- Dynamic resizing if the user changes the window's size. Resizing can also be disabled, ensuring a constant frame size.

Frames are generated with extensions `FRAMESET` and `FRAME`. In the HTML document, `FRAMESET` replaces the `BODY` tag. `FRAMESET` describes the frames to appear in the browser window.

You can include frames in your Panther application by specifying the HTML document or the HTML template that contains the `FRAMESET` element.

The Panther Gallery illustrates this in a sample screen entitled *Frames* which is accessible from the Web application server:

`http://server-name/cgi-bin/jwsamp/main`

---

## Using Style Sheets

---

You can define the look of your Panther web application using style sheets. The style definitions can be included in the HTML document or be specified using a URL location.

For each screen, change the Stylesheet Source property to indicate that style sheets will be used.

Stylesheet Source (`stylesheet_source`)

Specify whether style sheets for the web application screen are included in the screen's HTML (Inline) or specified in a URL (Link).

Stylesheet Type (`stylesheet_type`)

Specify the type of style sheet to be used for the web application screen: CSS (cascading style sheet) or JavaScript.

Stylesheet Data (`stylesheet_data`)

For inline style sheets, enter the style specification.

Stylesheet Link (`stylesheet_link`)

Specify the URL location of the style sheet. The style sheet file must be located in the HTTP server's primary document directory. A common name for this directory is `htdocs`.

For example, the following cascading style sheet specification would change all H1 headings to be in a sans-serif font, Arial if it is available, and to be red in color.

```
H1 { font-family: "Arial", sans-serif; color: red }
```

To specify styles for widgets, use the HTML Attributes property to enter the setting, as in:

```
STYLE = "..."
```

W3C has issued a recommendation on the use of style sheets. For more information, see their website at [www.w3.org](http://www.w3.org).

---

## Creating Headings

---

Headings can be used to give order to an HTML document. Browsers apply distinctive typefaces or styles to distinguish heading levels H1 through H6.

### To create a heading:

1. Create a dynamic or static label.
2. In the Properties window under Web Options, expand the Custom HTML properties. Select the Prefix Markup subproperty.
3. Enter the start tag for the heading level that you want—for example, `<H1>` for heading level 1.
4. Select the Suffix Markup subproperty to close the markup tag.
5. Enter the end tag for the heading level that you created—for example, `</H1>` to end a level 1 heading.

---

## Drawing Horizontal Rules

---

A horizontal rule visually divides the contents of an HTML document. You can use it to group together similar controls or to make the page easier to read.

## To create a horizontal rule:

1. Create a line widget.
2. To specify line thickness, expand the Web Options category, and select the Attributes subproperty.
3. Enter the size attribute for the line. For example, entering `SIZE=4` specifies a line thickness of 4 pixels which results in the HTML output of `<HR SIZE=4>`.

---

# Using Cookies

---

Cookies are pieces of information from the browser side of a connection. After a cookie is set by an HTML document, it is stored on the browser and can be retrieved when the browser contacts the same HTTP server.

Cookies are best used for simple, persistent, client state information, such as a user ID, the date, or the number of times the client visits a specific URL. If the cookie specification includes an expiration date, this information is saved on the browser and is available in subsequent browser sessions by the same user.

The cookie must initially be set by an HTML document—in a Panther Web application, by calling `sm_web_set_cookie`. For example, the following screen entry function retrieves two cookie values, `user` and `visit_num`. `visit_num` is then incremented by 1 and `sm_web_set_cookie` resets the corresponding cookie to its new value:

```
proc entry
user=sm_web_get_cookie("user")
visit_num=sm_web_get_cookie("visit_num")
visit_num=visit_num+1
call sm_web_set_cookie("visit_num=:visit_num;\
    expires=Monday, 03-Jan-2000 00::00::00 GMT; \
    domain=.prolifics.com; path=/samples")
```

## Retrieving Cookie Values

Cookies are retrieved when a browser requests a URL from an HTTP server. The browser compares the value of the URL with the domain and path of any cookie values stored on the browser. If any of them match, a line containing the name=value pairs of all matching cookies is included in the HTTP request.

Cookie values can be retrieved in your Web application with `sm_web_get_cookie`, as illustrated in the previous example. They are also available in the CGI variable `@cgi_http_cookie`.

The Panther Gallery sets and retrieves cookie values in a documentation sample entitled `Client-side Cookies` which is accessible from the Web application server:

```
http://server-name/cgi-bin/jwsamp/main
```

---

## Embedding Java Applets

---

You can embed Java applets into your Panther Web application to create attractive and interesting Web pages with animation, scrolling text, and banners. Java applets are built with Java, an object-oriented programming language. A Java applet is not a standalone program; it must be embedded within an HTML document and is triggered by a JavaScript event. Java applets can run only in a Java-compatible browser.

If you include Java applets in an application, take into account these limitations:

- Java provides no method to transmit user input from the Java applet to the HTML document. The Java applet can only be used to display information.
- You can not dynamically change a Java applet while a user is working on it. For example, if a Java applet is a pie chart, you can not change the chart to reflect new values while the user is entering data into the form.

To incorporate a Java applet into a Web application, you need this information:

- The applet's name (Java is case sensitive).



- The applet's .class file location. This file can be stored in a Panther library.
- The arguments that the applet accepts.

## To embed a Java applet into your Panther screen:

1. Create a dynamic label on your screen.
2. In the label's Label property, enter the text to display when a browser is not Java-enabled.
3. In the label's Prefix Markup property (under Custom HTML), select the Prefix Markup property and enter the APPLET tag and its arguments.

For example, this entry specifies to call the TickerTape applet:

```
<APPLET CODE="TickerTape" WIDTH=300 HEIGHT=40 ALIGN=top>  
<PARAM NAME=fontsize VALUE="20"> <PARAM NAME=message  
VALUE="This is my sample Java applet!">
```

In this example, Panther looks for a file named TickerTape.class.

4. Select the label's Suffix Markup subproperty and enter the applet's end tag.

For example, given the previous setting for Prefix Markup, the following entry closes the TickerTape applet:

```
</APPLET>
```

In the Panther Gallery, the TickerTape applet is modified through Panther properties. The sample screen *JavaApplets* is on the Web application server:

```
http://server-name/cgi-bin/jwsamp/main
```

---

## Refreshing Screens in a Web Browser

---

If your screen contains information which needs to be updated frequently, you can either have the user refresh the screen in the Web browser by pressing the Reload button or have the screen automatically refresh itself using the META tag. For this task, the META tag format is:

```
<META HTTP-EQUIV="refresh" CONTENT="seconds; URL=URL_location">
```

Another use for this tag would be for an online slide show with the URL for the next screen in the META tag. This technique is known as client pull since reading the HTML document into the Web browser activates the process.

### To specify a META tag:

Under Web Options, enter the META tag in the Head Markup property. For example, the following entry refreshes a screen each minute:

```
<META HTTP-EQUIV="refresh" CONTENT="60; URL=myapp.scr">
```

---

## Using ActiveX Controls

---

For ActiveX controls to be available in your Web applications:

- The user must have a browser that supports ActiveX controls.
- The browser security settings must allow ActiveX controls to be downloaded.
- The Codebase property must specify the location on your Web application server of the downloadable file containing the ActiveX control.

In Panther Web applications, the control's properties can be set using the JPL property syntax or in JavaScript or VBScript functions. Methods can be called using JavaScript or VBScript. Event control must be specified using VBScript in Microsoft's Internet Explorer 4. With Internet Explorer 3, JavaScript can also be used for event handling.

## Using ActiveX Controls in Web Browsers

In the Web environment, only Microsoft's Internet Explorer has native support for ActiveX controls, although most browsers have plug-in support.

In Internet Explorer, if a user accesses a Web document containing an ActiveX control not already registered on the user's system, Internet Explorer performs the following steps:

- Internet Explorer checks to see if the control has been digitally signed.
  - If digitally signed and security is set at high or medium, the certificate appears in a dialog box asking the user if they want to download the control.
  - If not digitally signed and security is set at high, a dialog box tells the user that the control may be unsafe. The Web document displays a placeholder instead of the control.
  - If not digitally signed and security is set at medium, a dialog box tells the user that the control may be unsafe, giving the user the option of downloading the control. If the user chooses not to download the control, a placeholder, instead of the control, is displayed in the Web document.
  - If security is set to low, the control is installed and registered on the user's system and loaded and displayed in the Web document.

Once a control is registered on a user's system, the dialog boxes at high and medium security levels no longer appear. Even if originally not signed, it is considered safe.

- Before your Web document initializes the control, the browser checks the control's properties to verify that it is marked safe for both initializing and scripting.
  - On a high security setting, if the control fails these checks, the control will display and run but without initialization and scripting. Without

initialization, default settings are used for the control's properties. Without scripting, all scripts fail and display a dialog box stating that fact.

- On a medium security setting, a dialog box is displayed for each failed check. The user can choose to allow initialization or scripting. As with the high setting, without initialization, default settings are used for the control's properties. Without scripting, all scripts fail and display a dialog box stating that fact.
- On a low security setting, the ActiveX control is initialized and all scripts are allowed to run.

For Mozilla Firefox, one plug-in for ActiveX controls is IE Tab V2 at <https://addons.mozilla.org/en-US/firefox/addon/ie-tab-2-ff-36>.

## Signing Your ActiveX Controls

When an ActiveX control is embedded in a Web application, it is suggested that you digitally sign the control and set which licensing scheme (development or runtime) you want implemented.

In order to digitally sign controls, you will have to obtain a Software Publisher's certificate from a Certificate Authority, such as Verisign. It takes about a week to obtain this certificate. For information about signing controls and the underlying Authenticode technology, refer to Microsoft's website, [www.microsoft.com](http://www.microsoft.com). The VeriSign site, [www.verisign.com](http://www.verisign.com), contains information about their certification program.

Digitally signing the control places your company's guarantee for the control. Therefore, you should check that the control only modifies the user's system as described in the ActiveX specifications.

## Submitting Data to the Web Application Server

In Web applications, any data submitted to the Web application server must be in an input field. Since ActiveX controls have no input fields, data must be transferred from the ActiveX control to Panther input variables using JavaScript or VBScript before the screen is submitted back to the Web application server.

This is typically done by copying the relevant information to hidden fields in the HTML. Such a hidden field is created by setting both the Identity→Hidden property and the Web Options→Export To HTML property to Yes. This generates an HTML `<INPUT>` element with the `HIDDEN` attribute.

## Submitting Data using JavaScript

The following JavaScript `onSubmit` function, called when the screen is submitted back to the Web application server, transfers data from the `Value` property of the `Pr1Spinner` Controls back to the Panther variables:

```
function onSubmit()
{
    document.main.i_1_len1.value =
        document.main.Pr1Spinner1.value;
    document.main.i_1_len2.value =
        document.main.Pr1Spinner2.value;
    document.main.i_1_len3.value =
        document.main.Pr1Spinner3.value;
}
```

## Submitting Data using VBScript

Here is the same function in VBScript. It is defined as the `onClick` function associated with a push button.

```
Sub spo_1_button_onClick
    document.main.i_1_len1.value =
        document.main.i_1_Pr1Spinner1.value
    document.main.i_1_len2.value =
        document.main.i_1_Pr1Spinner2.value
    document.main.i_1_len3.value =
        document.main.i_1_Pr1Spinner3.value
End Sub
```

## Generating HTML for ActiveX Controls

In order for the Web application to download the control, you must set the `Codebase` property (under `Web Options`) which lists either the URL for the ActiveX control file from the document root directory or the location relative to the application directory.

For example, the following entry in the `Codebase` property:

PrlSpinner.ocx

results in the following HTML, prepending the name of the current machine:

```
CODEBASE="http://currentMachine/my_app/PrlSpinner.ocx"
```

However, if the HTTP protocol is specified as part of the Codebase property:

```
http://prolifics.com/PrlSpinner.ocx
```

the following HTML is generated, which downloads the file from the Web application server's documentation root directory:

```
CODEBASE="http://prolifics.com/PrlSpinner.ocx"
```

When an application screen contains an ActiveX control, an OBJECT tag is generated for the control container with PARAMETER attributes for each of the ActiveX properties and a CODEBASE attribute containing the URL for the ActiveX control. The following example is the OBJECT tag for the PrlSpinner Control.

```
<OBJECT ID=i_1_spinner WIDTH=83 HEIGHT=30
  CLASSID="CLSID:DA2599CE-F939-11D0-A19E-00A02481A2E9"
  CODEBASE="http://prolifics.com/src/PrlSpinner.ocx">
  <PARAM NAME="BlankIfZero" VALUE="0">
  <PARAM NAME="Maximum" VALUE="100">
  <PARAM NAME="Minimum" VALUE="0">
  <PARAM NAME="Value" VALUE="25">
</OBJECT>
```

When the OBJECT tag is included in the HTML document, Internet Explorer performs the following steps:

- Searches the registry for the CLSID. If not found, it performs the following steps:
  - Parses the OBJECT tag and searches for the CODEBASE attribute. If the CODEBASE attribute is absent or is preceded by a URL-to-object index server in the CodeBaseSearchPath, this index is used to retrieve the file.
  - Locates the file identified by the CODEBASE attribute.
  - Copies the files to the user's computer.
  - Registers the objects and/or files that require registration.
- When the control is registered, calls the Component Object Model (COM) function CoCreateInstance to create an instance of the specified object.

A `.cab` file is one software distribution format for ActiveX controls. Part of the `.cab` file is the binary and `.inf` file needed to run on the associated platform. An `.inf` file contains information used by Microsoft Windows to load and register an ActiveX control.

ActiveX controls can run on Intel x86 computers, on the Apple Macintosh, as well as on any of several RISC machines. You need to create, test, and bundle your control's binaries for all deployed platforms.

---

## Embedding Sound

---

A Panther web application can incorporate multimedia elements, including sound. Browsers such as Mozilla Firefox and Internet Explorer provide inline support for audio. The browser can play audio files provided two conditions are true:

- The required auxiliary plug-in programs are available.
- The PC or workstation on which the browser is running has a sound card installed.

Sounds are stored in different formats, known as MIME types (Multipurpose Internet Mail Extensions). The following table lists some standard MIME types:

Mime Type	Filename Extension	Description
audio/basic	<code>.au</code> , <code>.snd</code>	Sun Microsystems 8 bit Audio
audio/x-aiff	<code>.aif</code> , <code>.aiff</code> , <code>.aifc</code>	Macintosh audio
audio/x-wav	<code>.wav</code>	Microsoft audio
audio/x-pn-realaudio-plugin	<code>.rpm</code>	Real Audio

**Note:** To add MIME types such as Quick Time Movies, check your browser's documentation.

## **To embed a sound file in your application:**

1. Create a dynamic label widget on your screen.
2. In the Properties window under Web Options, select the Default Link property.
3. Enter the URL of the sound file. For example:

`http://myserver.com/cgi-bin/proweb/sound1.au`

To view a sound file in a Panther application, refer to the part of the Panther Gallery entitled *Sound Files on the Web* application server:

`http://server-name/cgi-bin/jwsamp/main`



# 9 Using JavaScript and VBScript

JavaScript and VBScript are interpreted, scripting languages that let you embed simple programs into Web pages. JavaScript/VBScript programs are embedded directly within the HTML document. When the browser loads the HTML document, the programs are also loaded.

With minimal programming, you can use JavaScript/VBScript to create dynamic Web pages that perform validation, and display popup prompts and dialogs to users.

**Warning:** Some browser programs do not support JavaScript or VBScript, while those browser programs that offer support also allow users to disable it. If support is unavailable, the Web application can be affected as follows:

- Data validation that Panther performs with JavaScript does not work.
- Action list box items located in grids are not displayed as hyperlinks.
- Custom JavaScript/VBScript that is embedded in an application does not execute.
- Embedded Java applets do not execute.

---

# Browser Events

---

The events that can occur when a user is viewing an HTML document provide places for triggering a JavaScript or VBScript function. For example, the `onSubmit` event is triggered when a user submits an HTML form back to the server.

Screens and widgets provide Browser Event properties as hooks for attaching JavaScript or VBScript functions. Each event property corresponds to an event type. For example, you can specify a function for a screen's `On Submit` property; whenever the form is submitted back to the server, the `onSubmit` event occurs and triggers execution of the attached function. Similarly, you can specify a function for a text widget's `On Change` property; the function executes whenever an `onChange` event occurs to the widget.

A Browser Event property can specify one of the functions that are defined in the screen's JavaScript or VBScript property; or it can contain a JavaScript or VBScript statement.

Some events can cause automatic generation of JavaScript for certain property settings. For example, if a widget's `Convert Case` property is set to `Upper`, Panther generates the appropriate JavaScript in order to implement the desired conversion when an `onChange` event occurs—for example, the user clicks out of the field. Panther supports all JavaScript and VBScript events, associating them with screen and widget properties under the Browser Events category.

## JavaScript Event Properties

### Screens

Screens have these browser event properties:

#### On Load

The HTML document is loaded in the browser.

**On Submit**

The user clicks a push button that submits the document back to the server. This event is triggered before the form is actually submitted.

**On Unload**

The user exits or unloads the HTML document from the browser.

## Widgets

Browser event properties vary according to a widget's type. This section lists the complete set of widget events:

**On Blur**

Focus is removed from a specific widget within a form.

**On Change**

The user switches focus away from the field and the field is modified. This event is triggered only after the user completes entering or modifying information.

**On Click**

The user clicks the mouse button on a hyperlink, a push button, a check box, or any other selection widget.

**Note:** For push buttons, do not specify both a pixmap and an On Click event. In HTML specifications, the image takes precedence, making the On Click event unavailable.

**On Focus**

The user clicks the mouse or presses TAB to activate, or bring focus to, a specific widget within the form.

**On MouseOver**

The mouse pointer moves over an area, such as a hyperlink. For example, this event can be used to display a hyperlink's URL in the browser status line when the mouse moves over it.

**On MouseOut**

The mouse pointer leaves an area (in client-side image maps) or a link. For example, if the mouse moves from one area into another in a client-side image map, the onMouseOut event occurs for the first area, followed by the onMouse Over for the second area.

If the JavaScript program sets text on the status line, the program must return true.

On Select

Text within a text widget is selected.

## Setting Event Properties

For the events listed under Browser Options, specify any JavaScript or VBScript function that is defined for the screen or one of its widgets in its JavaScript/ VBScript property or include a JavaScript/VBScript statement. Simply select the desired event property and enter either the name of the function and its parameters, or the text of the statement.

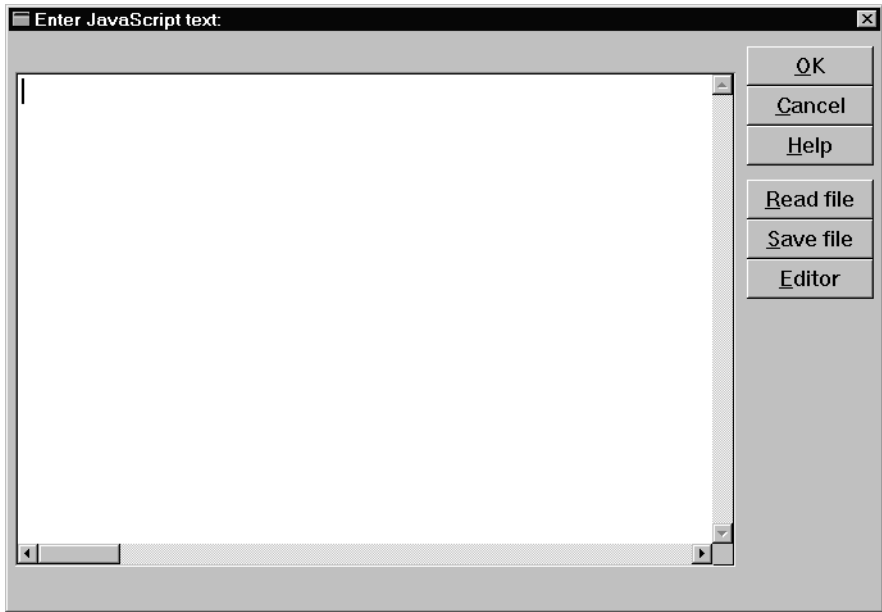
## Writing JavaScript and VBScript Functions

You can include your own JavaScript or VBScript function in the HTML document by entering the function in the JavaScript/VBScript property for the screen or a widget. After a function is entered into the property, you can specify that function for any browser event property for the screen and its widgets.

**Notes:** The functions that you write for a screen and individual widgets is equally accessible to all widget and screen event properties.

## How to Define a JavaScript or VBScript Function

1. Select the JavaScript/VBScript property for the screen or one of its widgets. The text dialog opens where you can write functions:



2. Enter or edit data with these actions:
  - Type the function directly into the dialog box window.
  - Choose Editor to invoke your local editor and enter the desired data.
  - Choose Read File to import an external file.
3. Choose OK to accept your changes.

**Note:** Panther does not check syntax in JavaScript statements and functions.

For example, the JavaScript property can contain the `num_links` function, which displays the number of hyperlinks (`<A HREF=>`) in the HTML document:

```
function num_links()  
  
msgtext="Number of links in document: " +  
    document.links.length;  
return window.confirm(msgtext)
```

## Accessing Widget Values

If a widget's Export to HTML property is set to Yes, an HTML tag is generated for the widget, even if the widget is hidden at runtime. This ensures that a JavaScript or VBScript function can reference the widget and access or manipulate its value in a Web browser.

## Accessing Widgets in JavaScript and VBScript

The naming conversion from Panther widget names to HTML field names prepends text in order to uniquely identify each widget's occurrence and purpose. The `html_name` property provides read-only access to the converted name. You can include that value in JavaScript or VBScript functions using the `{{variable->html_name}}` syntax.

If the `html_name` property cannot be used, the following table shows Panther's HTML naming conventions for named and unnamed widget types. If a widget is unnamed, Panther uses its field number in the HTML tag name. (For information about how widgets are numbered, refer to “Field Numbers” [on page 14-4](#) in *Application Development Guide*).

Panther widget type	HTML document name	Example
Named widgets:		
Text	<code>i_occurrence-num_widget-name</code>	<code>i_1_title_id</code>
Multitext	<code>mn_widget-name</code>	<code>mn_descrip</code>
List box	<code>sln_widget-name</code>	<code>sln_lname</code>
Option menu	<code>son_widget-name</code>	<code>son_type</code>
Push button	<code>spn_widget-name</code>	<code>spn_select</code>
Unnamed widgets:		
Text	<code>o_occurrence-num_field-num</code>	<code>o_1_2</code>
Multitext	<code>mf_field-num</code>	<code>mf_1</code>

Panther widget type	HTML document name	Example
List box	slf_field-num	slf_4
Option menu	sof_field-num	sof_3
Push button	spf_field-num	spf_2

The following section of a JavaScript function assigns time value to three widgets. This is how the function appears in the editor using the `html_name` property:

```
function showtime
{
// Assigning the time values
    document.forms[0].{{face_1->html_name}}.value=timeValue
    document.forms[0].{{face_2->html_name}}.value=londonValue
    document.forms[0].{{face_3->html_name}}.value=tokyoValue
}
```

In the generated HTML, the HTML name is substituted for the value inside the braces, and the function is output as follows:

```
function showtime
{
// Assigning the time values
    document.forms[0].i_1_face_1.value=timeValue
    document.forms[0].i_1_face_2.value=londonValue
    document.forms[0].i_1_face_3.value=tokyoValue
}
```

## Automatic JavaScript Generation

Panther automatically generates JavaScript to implement the Status Line Text property for widgets and screens, and to create hyperlinks for action list boxes in a grid widget. JavaScript is also automatically generated when certain input properties are set as shown in the following table, in order to validate or convert data:

<b>Input property</b>	<b>Value</b>
Keystroke Filter	Digits Only Yes/No Alphabetic Numeric Alphanumeric Edit Mask
Convert Case	Upper/Lower
Required	Yes
Must Fill	Yes
Minimum Value	Any non-null value
Maximum Value	Any non-null value

When a widget has one of these properties set, the entry in that widget is validated whenever an `onChange` or `onBlur` event occurs—for example, the user mouse clicks out of the field. Invalid data causes an error message to display. If data is to be converted, the conversion automatically takes place.

A widget that has JavaScript Event property settings might have two JavaScript functions triggered for the same event. In this case, the automatic JavaScript executes first, followed by the function or statement that the property specifies. For example, a widget might have its `Convert Case` property set to `Upper` and its `On Change` property set to a JavaScript function. In this case, the `onChange` event triggers two JavaScript functions for the same widget—first, the function that Panther automatically generates for `Convert Case`, then the `On Change` function.



# 10 Accessing Databases

Part of a Panther application is its database interface. Due to the stateless nature of the Web, the database processing in a Web application might not mirror the processing in other two-tier, client/server models. This chapter briefly describes the following topics that are necessary in building a database application:

- Connecting to the database.
- Initializing the Panther client.
- Using database cursors.
- Handling database transactions.
- Fetching multiple rows.

---

## Connecting to the Database

---

Database connections are handled differently in two-tier and three-tier applications. In two-tier client/server applications, the Web application server handles database connections as part of its jserver startup. In three-tier applications, the Panther application server maintains the database connections. This section describes the procedures required by two-tier applications.

In two-tier Panther Web applications, connections to the database are usually declared in the `web_startup` procedure. Generally, the `web_startup` procedure is part of a JPL module specified in the application variable `SMINITJPL`. When the Web application server starts a `jserver` process, it initializes the JPL modules in that procedure before calling the `web_startup` event.

By declaring the database connection for the `jserver` (and not for each client request), you can reduce the overhead incurred by connecting to the database server for each URL. Database connections are made using the command `DBMS DECLARE CONNECTION`. For example, the following `web_startup` procedure makes a connection to JDB's `videobiz` database:

```
proc web_startup
DBMS ENGINE jdb
DBMS DECLARE jdbconn1 CONNECTION FOR DATABASE 'videobiz'
return
```

Database connections should be closed in the `web_shutdown` procedure. Before the Web application server closes a `jserver` process, it calls the `web_shutdown` event. Database connections are closed with the `DBMS CLOSE CONNECTION` command, as in this example:

```
proc web_shutdown
DBMS CLOSE CONNECTION jdbconn1
return
```

Each database engine has its own syntax for database connections. For the command syntax, refer to “Database Drivers.”

---

# Initializing the Panther Client

---

In three-tier applications, the Web application server's startup procedure `web_startup` must initialize the server as a Panther client. The `web_startup` procedure is usually in a JPL module that is specified by the application variable `SMINITJPL`. When the Web application server starts a `jserver` process, it initializes the JPL modules in that procedure before calling the `web_startup` event. For example:

```
proc web_startup
    client_init
return
```

This connection to the request broker is closed in the `web_shutdown` procedure. Before the Web application server closes a jserver process, it calls the `web_shutdown` event.

```
proc web_shutdown
    client_exit
return
```

---

## Using Database Cursors

---

If you use the transaction manager, it automatically opens and closes the cursors it needs to perform database processing. However, if you are writing your own SQL statements, you also need to manage the database cursors.

For example, if you open a database cursor to execute a SQL statement or a stored procedure, you also need to close that database cursor. The commands to open and close the cursor should occur during a single URL request; otherwise, depending on your database engine, a shared lock can be in effect until the jserver shuts down or the cursor closes.

---

## Database Transactions

---

If you are writing your own SQL statements, you should also complete the data base transaction during a single request. The `DBMS BEGIN` and `DBMS COMMIT` (or `DBMS ROLLBACK`) should both occur when the screen is posted back to the server.

By testing for `K_WEBPOST`—the event flag that is set for `POST` events—you can ensure that the database transaction only starts when the screen is submitted back to the server. For more information on `K_WEBPOST`, refer to [page 5-7](#), “Screen Entry Context Flag.”

Some type of optimistic locking for the database should be in place so that the data remains concurrent. Pessimistic locking schemes are not recommended for Web applications. The transaction manager has a version field that can be used to perform optimistic locking. You can also use engine-specific locking techniques, such as timestamp columns.

---

## Fetching Multiple Rows

---

To fetch multiple rows, use Panther scrolling events instead of `DBMS_CONTINUE` commands. With Panther scrolling, you set the number of occurrences in the screen to exceed the number of rows in the select set.

For example, the screen has a grid with 10 onscreen rows and a maximum number of 500 occurrences. You execute a `SQL SELECT` statement that returns 600 rows. The first 10 rows of the select set are displayed in the browser. The first 500 rows of the select set are cached on the Web application server. However, the remaining 100 occurrences institute a shared lock on the database until the rows are flushed, the cursor is closed, or the jservlet is shut down.

# 11 HTTP Variables

As HTTP servers and Web clients exchange data, the HTTP protocol sends a series of header fields containing information about the data being transferred over the Web. These header fields can, in turn, be passed to any program on your HTTP server. Because the information in these variables can be useful in your Panther Web application, Panther converts some of these header fields to global variables that can be accessed through JPL procedures or C functions.

An HTTP header field has a corresponding Panther HTTP variable. The Panther variable name begins with `@cgi_` followed by the HTTP field name in lower case. The Panther HTTP variables are read-only and are automatically reset on each `GET` or `POST` of a Panther screen. Because Panther updates these variables automatically, copy their values elsewhere if you need them for a subsequent `POST`.

Common uses of the HTTP variables include:

- Building a URL using `@cgi_server_name` and `@cgi_script_name`.
- Calling a screen from an HTML document using `@cgi_http_referer`.
- Testing for MIME types and image formats in `@cgi_http_accept`.
- Testing for the browser version using `@cgi_http_user_agent`.

For example, this JPL procedure builds a URL for a link with `@cgi_server_name` and `@cgi_script_name`.

```
proc build_link
// This procedure sets dynamic label's
// default_link property

home->default_link = "http://" ## \
    @cgi_server_name ## \
    @cgi_script_name ## "/home.scr"
return 0
```

A Panther Gallery sample entitled *HTTP Variables* is accessible from the Web application server at:

`http://server-name/cgi-bin/jwsamp/main`

---

## Definitions

---

@cgi\_auth\_type

The authentication method required to authenticate a user who desires access to a protected script.

@cgi\_content\_length

The length of the data message if POST is used to submit data back to the server.

@cgi\_content\_type

The MIME Content-Type of the data if POST is used to submit data back to the server. For Panther, this variable is set to:

`application/x-www-form-urlencoded`

The basic MIME types are listed in the following table:

MIME type	Description
application	Binary data that can be executed or used by another application
audio	Sound data
image	Image data
message	Encapsulated mail message
multipart	Multiple parts possibly consisting of many data types
text	Textual data

---

MIME type	Description
video	Video data

---

**@cgi\_gateway\_interface**

The version of the CGI specification to which this server complies in the following format:

*CGI/version-number*

A sample value would be:

CGI/1.1

**@cgi\_http\_accept**

A comma-separated list of MIME Content-Types that are acceptable to the client. They are listed in the following format:

*type/subtype*

You can use the contents of this variable to determine which image formats a browser can accept. For example, the following string indicates that the browser can display PNG, JPEG and GIF images:

image/png, image/jpeg, image/gif

**@cgi\_http\_cookie**

A list of all the returned cookie values, separated by semi-colons. The cookie values are returned when the browser requests a document from the same HTTP server that set the cookie.

**@cgi\_http\_referer**

The URL of the document where the request originated. This can be a partial URL, in which case it is interpreted relative to the URL of the document being requested.

For Panther applications, this information is only useful on GET events. If the user enters a Panther application from elsewhere, for example from another website, this variable will specify that location, which can then be logged if desired.

**@cgi\_http\_user\_agent**

Information about the browser software making the request. This variable identifies the browser type. Even though there is no standard format, this information usually appears in the following format:

### *software/version comments*

You can determine the format for a particular browser only by experimentation. For example, the value sent from 32 bit Microsoft Internet Explorer 10.0 running on a 64 bit Windows 7 system might be:

```
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
```

### @cgi\_path\_info

Any extra path information found in the URL. Generally, this is the relative path to a resource.

Panther uses this information to determine which Panther screen to open. For the example, this variable would be:

```
/vbiz/vidlist.scr
```

### @cgi\_path\_translated

The absolute path on the local system for a resource. This is done by pre-pending the server's document root directory, or `DOCRROOT`, to the path specified in `@cgi_path_info`.

If the server's `DOCRROOT` directory is `/usr/local/htdocs`, the variable would have the following value for the screen listed in the example:

```
/usr/local/htdocs/vbiz/vidlist.scr
```

### @cgi\_query\_string

A URL-encoded search string. This string is separated from the URL by a question mark. For the example, this variable would be:

```
start=top
```

For more information on URL encoding, refer to [page 4-3](#), “Encoding Parameters in the URL.”

### @cgi\_remote\_addr

The numeric IP address of the remote computer making the request. This is not necessarily the address of the client, but could be the address of the host machine where the browser is running.

### @cgi\_remote\_host

The Internet domain name of the host machine making the request. This host machine is where the browser is running. If the domain name is unavailable, this field is left blank.



@cgi\_remote\_ident

The remote user name retrieved by the server using the idntd identification daemon.

@cgi\_remote\_user

The authenticated name of the user.

@cgi\_request\_method

The method associated with the request. For HTTP servers accessing Panther, this will be either GET or POST. The GET method is used when the user enters a specific URL or activates a hyperlink. The POST method is used when the user submits a form. Using this variable, you can choose which portions of your code to execute for each method.

@cgi\_script\_name

The path and name of the CGI script being accessed, as it is referenced in a URL. Note that this is only the path that appears in the URL; it is not the actual, complete path of the CGI program. For the example, this value is:  
*/cgi-bin/webdev*

@cgi\_server\_name

The Internet domain name of the HTTP server. If the domain name is not available, the numerical IP address is used. This is useful if the HTTP server is acting as home for multiple domains, and they each call the same Panther executable. In the example, the domain name is *vbiz.com*.

@cgi\_server\_port

The port number receiving the browser request.

This is 123 in the example. Port numbers are useful if there are multiple servers running on the same machine, each calling the same Panther executable.

@cgi\_server\_protocol

The name and version of the information protocol for the incoming request using the following format:

*protocol/version-number*

A sample value for this variable would be:

*HTTP/1.0*

@cgi\_server\_software

The name and version of the HTTP server software invoking the external program. The format is:

*software-name/version*

The following examples are typical values:

Apache/1.3.14 (Unix) (Red-Hat/Linux) PHP/4.0.3pl1

Microsoft-IIS/7.5

# 12 Web Initialization Options

A Panther web application requires its own initialization file whose name is derived from the name of the requester executable, using the format *application-name.ini*.

The default initialization file is named *proweb.ini*. On Windows, it is located in the `\WINDOWS` directory or where Windows is installed. On UNIX, it is located in `~proweb/ini`. Before you start development of a Panther web application, run the Web Setup Manager utility in your browser. This tool creates the initialization file from settings you specify. For more information on the Web Setup Manager, refer to Appendix B, “Web Setup Manager.”

The following sections describe the different types of environment variables that can be found in your application's initialization file. There are variables that determine the behavior of the web application, the Panther environment used by the jserver program, and the database environment.

---

## Setup Variables

---

The web initialization file contains an `[Environment]` section where you set Panther setup variables such as `SMBASE` and `SMFLIBS`. This section also sets the directory path to use for Panther utilities and programs.

**Note:** Panther variables are case sensitive.

## Required Settings

Two variables must be set for two-tier and three-tier applications: [SMBASE](#) and [SMFLIBS](#). [SMBASE](#) specifies the location of the web application server installation on the HTTP server.

Some typical settings for [SMBASE](#) are:

- UNIX: `SMBASE=/usr/panther`
- Windows: `SMBASE=C:\Program Files\Prolifics\Panther`

## Required for JetNet/Oracle Tuxedo Applications

One or more of the following variables must be set in order for the web application server to connect to the middleware as a client:

- [SMRBCONFIG](#) is required for a web application server that connects to the middleware as a non-workstation client (one that resides on an master or non-master machine). This variable is set to the location of the JetNet configuration file; this setting must match the value specified for the machine's local configuration file (refer to “Local JetNet Configuration File” on page 3-14 in *JetNet/Oracle Tuxedo Guide*).

UNIX: `SMRBHOST=/home/myapps/broker.bin`

Windows: `SMRBHOST=Panther-appserver-location\broker.bin`

- [SMRBHOST](#) and [SMRBPOR](#) are required for a web application server that connects to the middleware as a workstation client (refer to “Workstation Clients” on page 2-8 in *JetNet/Oracle Tuxedo Guide*). [SMRBHOST](#) provides the middleware with the network addresses of the machines to which the client can connect; [SMRBPOR](#) provides the port number associated with each machine that the client can use to establish its connection.
- [SMTPJIF](#) specifies the JIF file to open for the web application.

## Optional Settings

The following variables only need to be set under certain conditions, as noted in their descriptions:

- `LD_LIBRARY_PATH` must be set to enable three-tier processing for a web application server that resides on a UNIX machine. `LD_LIBRARY_PATH` (or its equivalent) must be set to the location of shared Motif libraries. For example:

```
LD_LIBRARY_PATH=$SMBASE/lib
```

On HPUNIX, use `SHLIB_PATH` in place of `LD_LIBRARY_PATH`; on AIX, use `LIBPATH`.

- `PATH` sets the application's `PATH` variable and overrides any existing `PATH` settings. Set `PATH` to the Panther Web `util` subdirectory if the `jservlet` executable is not in it, and to the required database directories if two-tier processing is implemented.

You must provide explicit path names; environment variables are not expanded:

```
UNIX: PATH=/home/proweb/util:$PATH
```

```
Windows: PATH=C:\proweb\util
```

- `SMINITJPL` specifies a JPL file to run on web application startup and shutdown. This file must be in the web application server's `client.lib` library.
- `JAVA_HOME` specifies the location of the Java Virtual Machine program.
- `SMJAVAFACTORY` specifies the name of the Java class factory, if different from the default, `com.prolifics.jni.ClassTagFactory`.
- `CLASSPATH` specifies the location of the Java class libraries. The Panther class libraries are at `$SMBASE/config/pro5.jar`. You must also specify the location of the Java class libraries that you add to your Panther application.

```
UNIX: CLASSPATH=$SMBASE/config/pro5.jar:$CLASSPATH
```

```
Windows: CLASSPATH=$SMBASE\config\pro5.jar;%CLASSPATH%
```

## Database Information

For two-tier applications, the [Environment] section can contain variables needed for your database configuration. For example, the following entry sets these variables for Oracle on UNIX:

```
ORACLE_HOME=/u/home/oracle
ORACLE_SID=oracle
```

Under Windows, installing any Panther database driver except JDB automatically adds a third section, [Database], to the initialization file. This section lists the DLLs that Panther should load at startup.

---

## Behavior Variables

---

The initialization file's [Prolifics Web] section contains variables that determine the behavior of the web application server. The `AppDirectory`, `Dispatcher`, `LMLicenseFile`, and `Server` variables are required; all others can be omitted. To view a sample, refer to [page 12-9](#), “Sample Initialization File.”

**Note:** Web option names are not case sensitive.

## Required Settings

The following variables must be set:

### `AppDirectory`

The path name of the application's working directory. This directory includes `common.lib` and `client.lib`, which contains the application's screens, reports, JPL modules, graphics, and other files needed by the application.

The path name must be complete. For example:

UNIX: `AppDirectory=/home/webapps/vidorder`

Windows: `AppDirectory=C:\webapps\vidorder`

#### Dispatcher

The location of the dispatcher program. If the path has no leading /, the location is relative to the location specified in AppDirectory.

UNIX: Dispatcher=/home/proweb/util/dispatcher

Windows: Dispatcher=C:\proweb\util\dispatch.exe

#### LMLicenseFile

The name and location of the license file. This entry is required.

LMLicenseFile is equivalent to LM\_LICENSE\_FILE.

UNIX: LMLicenseFile=/home/prolifics/licenses/license.dat

Windows: LMLicenseFile=C:\proweb\licenses\license.dat

#### NumServers

The number of jserver processes, or concurrent users, that the dispatcher can run simultaneously. In order for the application to run, it must be set greater than 0 (its default value).

NumServers=5

#### Server

The location of the jserver program. If the path has no leading /, the location is relative to AppDirectory.

UNIX: Server=/home/proweb/util/jserver

Windows: Server=C:\proweb\util\jserver.exe

## Optional Settings

The following variables are optional; if omitted, Panther supplies default values as specified.

#### BrowserData

Determines whether the application uses server or browser caching. Server caching puts the cached data in a temporary file on the HTTP server. Browser caching includes the cached data in the HTML data that Panther sends to the browser.

A value of null or 0 enables server caching; a value of 1 enables browser caching. The default specifies server caching.

BrowserData=0

For more information about caching data, refer to Chapter 6, “Preserving Application State.”

`CacheDirectory`

The path name on the HTTP server where a jserver creates and finds server cache files. The default is the subdirectory `procache` in the system's `tmp` directory. Panther creates this directory if it does not exist. If directory creation fails, it enables browser side caching.

The cache files for each application are located in a separate subdirectory. The subdirectory name matches the application name.

UNIX: `CacheDirectory=/home/proweb/procache`

Windows: `CacheDirectory=C:\proweb\procache`

For more information about caching data, refer to Chapter 6, “Preserving Application State.”

`ClientLog`

The path name of the file to which the requester logs events.

UNIX: `ClientLog=/home/proweb/logs/client.log`

Windows: `ClientLog=C:\proweb\logs\client.log`

**Note:** All the logging variables—`ServerLog`, `ClientLog`, and `ErrorFile`—can point to the same file if desired. The entries are listed in chronological order.

`EnableWebid`

Enables the caching of state information on a `GET` using `webid` in the URL by setting this variable to 1.

`EnableWebid=1`

`ErrorFile`

The path name of the file to which the dispatcher appends server error messages. You can also write application errors to this file using [sm\\_web\\_log\\_error](#).

UNIX: `ErrorFile=/home/proweb/logs/error.log`

Windows: `ErrorFile=C:\proweb\logs\error.log`



**Note:** All the logging variables—`ServerLog`, `ClientLog`, and `ErrorFile`—can point to the same file if desired. The entries are listed in chronological order.

`ExpireTime`

The amount of time in minutes that the application waits for the `POST` of a screen before removing the screen's server cache file. The default value is 120 (two hours). Setting this variable to a negative number inhibits expiration of any cache.

```
ExpireTime=120
```

`HTTPHOST`

If using `SSL`, the name of the host computer running the `HTTP` server. If the default port number is not used, it must also be specified.

```
HTTPHOST=server-name
```

`HTTPSHOST`

If using `SSL`, the name of the host computer running the secure server. This option can be set only if `HTTPHOST` is also set.

```
HTTPSHOST=server-name
```

`IdleServerTimeout`

The number of seconds that a `jservlet` will wait for an incoming request before exiting. If unspecified, the process continues indefinitely.

Sybase users should set this variable to avoid timeouts in the database server.

```
IdleServerTimeout=
```

`ImageDir`

The directory containing the graphics files for the web application. If specified, the `HTTP` protocol is used to fetch the graphics files instead of the web application server. This directory must be a sub-directory of the `HTTP` server's document root directory. If unspecified, the Panther search path is used to locate graphics files.

For intranets, enter the sub-directory; the machine name is prepended in the `HTML`:

```
ImageDir=my_app
```

For the Internet, enter the protocol, domain name, and document root sub-directory:

```
ImageDir=http://prolifics.com/my_app
```

Panther provides internal graphics for displays of scrolling grids, web reports and wizard transaction pages. If you are using any of these graphics in your application and you choose to use the `ImageDir` setting, you must copy these internal graphics to that HTTP server directory as well. These graphics have been provided separately in the graphics subdirectory of the Panther web installation.

`ListenQueueLength`

Determines the length of the listen queue. This is the value passed to the listen function, which is a standard socket call. It approximately represents the number of web requests that can be waiting for an available server at any given moment.

`PadOptionMenus`

For option menus, generates trailing and HTML spaces (`&nbsp;`) in option menus if set to `Yes`. To only pad the first occurrence, set this option to `First`.

```
PadOptionMenus=Yes
PadOptionMenus=First
```

`RetainCacheFiles`

Determines whether the application retains the server cache file for the screen after the screen is submitted back to the HTTP server. In order to allow use of the Back and Forward buttons in the browser program, the server cache files need to be retained. The `ExpireTime` option determines how long to retain them before they are deleted.

A value of null or 0 deletes cache files when a screen is submitted on a POST event; a value of 1 retains the cache file. The default specifies to delete the cache file.

```
RetainCacheFiles=0
```

`ServerLog`

The path name of the file to which the dispatcher and jserver logs events.

```
UNIX: ServerLog=/home/proweb/logs/server.log
```

```
Windows: ServerLog=C:\proweb\logs\server.log
```

**Note:** All the logging variables—`ServerLog`, `ClientLog`, and `ErrorFile`—can point to the same file if desired. The entries are listed in chronological order.

ServerTimeOut

The time in seconds after which the requester or dispatcher aborts its wait response for a jserver. This traps infinite loops that may otherwise be hard to interrupt. If this variable is not set, then it waits indefinitely.

ServerTimeOut=60

---

## Sample Initialization File

---

A sample configuration file on UNIX might contain the following [Prolifics Web] and [Environment] sections:

```
[Prolifics Web]
```

```
AppDirectory=/home/webapps/vidstore
```

```
ServerLog=logs/server.log
```

```
ClientLog=logs/client.log
```

```
ErrorFile=logs/error.log
```

```
ExpireTime=120
```

```
CacheDirectory=/tmp/procache
```

```
ServerTimeOut=60
```

```
MaxServers=10
```

```
Server=/usr/panther/util/jserver
```

```
Dispatcher=/usr/panther/util/dispatcher
```

```
LMLicenseFile=/usr/panther/licenses/license.dat
```

```
[Environment]
```

```
SMBASE=/usr/panther
```

```
SMFLIBS=client.lib
```

```
SMINTJPL=webapp.jpl
```

```
PATH=/usr/panther/util
```

```
LD_LIBRARY_PATH=/home/motif/usr/lib
```

```
SMRBHOST=aspen,willow
```

```
SMRBPOR=300,400
```



# 13 Deploying Web Applications

After you build an application, follow the instructions in this section to configure an application to run on the Web.

---

## How to Configure a Panther Web Application

---

1. Create an application directory on the web application server.
2. Copy `client.lib` to the application directory. This library should contain the files that belong to your application, including: Panther screens and reports, image files, JPL modules, JavaScript files, and Java applet files.
3. For JetNet and Oracle Tuxedo applications, copy `common.lib` to the application directory.

**Note:** The web setup manager can guide you through creating the requester and initialization files needed by your application.

4. Create the application's requester executable:

- For CGI: In the CGI programs directory, copy `proweb[.exe]` to `application-name[.exe]`.
- For ISAPI: In the ISAPI programs directory, copy `proweb.isa` to `application-name[.isa]`.
- For NSAPI: In the NSAPI programs directory, copy `proweb.nsa` to `application-name[.nsa]`.

**Note:** For NSAPI applications, NSAPI must be configured to run Panther web applications. For more information, refer to Appendix C, “Setting Up an NSAPI Web Server.”

5. Alternatively, the Panther web application can run as a Java servlet. For more information, refer to Appendix D, “Using Java Servlets.”
6. Create the application initialization file by copying the `proweb.ini` file. Name the copy `application-name.ini`. (For example, if the application name is `vidorder`, the initialization file would be called `vidorder.ini`.)

The `proweb.ini` file is located in:

- UNIX: the `ini` subdirectory under the `proweb` user home directory.
  - Windows: the `Windows` directory.
7. Edit the application's initialization file for the following settings. (For complete information on initialization file settings, refer to Chapter 12, “Web Initialization Options.”)

- Set the `AppDirectory` variable.

This variable specifies the name of your application directory (the directory that contains all of your application files). Be sure the period character (`.`) does not appear in the application directory path name.

- Set other variables that are specific to the application—for example, `ServerLog`, `ClientLog`, and `ErrorFile`.
- Set `NumServers` to the number of concurrent users. The value must be updated from its default value, which is 0.
- For better graphics performance, set `ImageDir` to the location of the graphics files in the `docroot` directory of the HTTP server.
- Verify the location of the license file.
- Set `SMBASE` to the Panther distribution on the Web application server.

- Set `SMFLIBS` variable to the Panther libraries.
- Set `SMINITJPL` to specify the JPL file to load on startup of the Web application in the variable. The JPL file should be in `client.lib`.

Typically, this JPL file contains the `web_startup` and `web_shutdown` procedures which set global variables for the application, open and close your database connections in two-tier applications, and open and close middleware connections in JetNet and Oracle Tuxedo applications. By including the JPL module in `SMINITJPL`, it is automatically initialized with the Web application server.

- For JetNet and Oracle Tuxedo applications, set the variables required to connect to the Panther application server: `SMRBHOST`, `SMRBPORT`, and/or `SMRBCONFIG` and to access the JIF file `SMTPJIF`.
  - For JetNet applications, specify the location of shared libraries through `LD_LIBRARY_PATH`, `LIBPATH`, or `SHLIB_PATH`.
  - For two-tier applications, set database environment information in the initialization file's [Environment] section. Under Windows, you also need to set the [Database] section. from `proweb.ini`.
8. For UNIX web applications, after the application is configured, start the application by using the `monitor` command. At the command line, type:  

```
monitor -start application-name[.exe]
```
  9. For Windows web applications, after the application is configured, install the application as a service using the `monitor` command. Refer to the `monitor` command for more information.





# A Web Application Utility

This chapter describes the command-line utility that can help you develop and manage a Panther Web application. The utility description is organized into the following components, as applicable:

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

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

```
monitor -h
```

## monitor

### *Administers Web applications*

```
monitor -option [appName...]
```

option

Specifies the task to perform with one of these constants:

-clean *appName* ...

Deletes expired cache files. This option is also automatically performed by the dispatcher while the application is running.

-configure *appName* ...

Directs the dispatcher to reread its configuration file and reset its behavior and the behavior of its jservers, if necessary.

-findsvc *serviceName*

For Windows, finds matching entries for the specified service name.

-install *appName* [-display *displayName*]

[-description *service-description*]

[-{automatic|manual|disabled}]

[-user {*domain\user*|\.*user*}]

[-password *password*] [[-depend *serviceName*] ...]

For Windows, installs the application as a service using the display name (if specified), service description (if specified), the user and password, and starting any services specified as dependent first. The automatic/manual/disabled option corresponds to the settings in the Services section of the Control Panel. The Control Panel can also be used to add installation options when they are not specified with this command.

-log *appName* ...

Starts the event log request.

-list

Lists the applications that are currently running.

-listsvc

For Windows, lists all service names and service display names.

-remove *appName*

For Windows, removes the application as a service.

- 
- `-restart appName`  
Runs the three following monitor options: `-clean`, `-stop`, `-start`. On Windows, if the Web application has been registered as a service, this option is obsolete.
- `-start appName ...`  
Starts the named application if it is not already running. On Windows, if the Web application has been registered as a service, this option is obsolete. In this case, use the Services section of the Control Panel or type `net start appName` to start the application.
- `-status appName ...`  
Produces the same output as the `-log` option but sends it to standard output (or to another convenient platform-specific place) for immediate viewing.
- `-stop appName ...`  
Stops the application. This action causes a normal shutdown of dispatchers and jservers. On Windows, if the Web application has been registered as a service, this option is obsolete. In this case, use the Services section of the Control Panel or type `net stop appName` to stop the application.

*appName*

The name of the application's configuration file without its `.ini` extension. This argument is identical to the name of the application's requester. You can specify multiple applications with the same command. If you specify an application that is not running, the utility returns an error message.

---

**Description** `monitor` is a command-line utility—`monitor` on UNIX, `monitor.exe` on Windows—which lets you administer a running web application. Each invocation can perform a single task on one or more applications. You can use `monitor` to start and stop a web application server; you can also perform these maintenance tasks:

- Delete expired cache files.
- Start logging events, either to the event log file or to standard output.
- List the applications that are currently running.
- Modify the behavior of a dispatcher and its jservers by instructing them to reread its configuration file.

**Installing an Application as a Windows Service**

For Windows Web servers, you can install the Web application as a service using the `-install` option. You can also specify the display name and description for the Services window and the dependent services that must be started before this service. For dependent services, the service name, not the display name, must be specified. If more than one dependent service needs to be specified, the `-depend` keyword must be specified before each service. If the name of the associated service contains a space, that service should be surrounded in double quotes.

Use the `monitor -listsvc` or `monitor -findsvc` options to find the service name.

Once the application is installed as a service, you must start the application with the `net start` command, in the Services dialog box in the Control Panel, or by having the machine automatically start the application when it reboots. The `monitor -start` command must not be used.

For example, if the application is named `storeinven` and needs database services named `DBProcess 1` and `DBCheck`, the following command line installs the application as a service:

```
monitor -install storeinven -depend "DBProcess 1" -depend DBCheck
```

In addition, if the name of this application in the Services dialog box is Video Store Inventory, the command line changes to the following:

```
monitor -install storeinven -display "Video Store Inventory" -depend "DBProcess 1" -depend DBCheck
```

You must stop the application with the `net stop` command or in the Services dialog box.

**Order of Services**

It is also recommended that the services needed by the Panther web application start in a specific order:

- Database engine
- Oracle Tuxedo IPC Helper (for JetNet and Oracle Tuxedo applications)
- HTTP server
- Panther Web application

**Full Syntax**

The full syntax of the command is:

```
monitor -start application-name [application-name ...]  
monitor -stop application-name [application-name ...]  
monitor -configure application-name [application-name ...]  
monitor -log application-name [application-name ...]  
monitor -clean application-name [application-name ...]
```

```
monitor -status [application-name ...]
monitor -list
monitor -install application-name
    [-display display-name]
    [-description service-description]
    [-{automatic|manual|disabled}]
    [-user {domain\user|\.user}]
    [-password password]
    [[-depend service-name]...]
monitor -remove application-name
monitor -restart application-name
monitor -listsvc
monitor -findsvc name
```

The `-install`; `-remove`; `-listsvc` and `-findsvc` options are only supported in Windows.



# B Web Setup Manager

The Web setup manager utility creates or updates the configuration files needed by your Web application. After you install your Panther Web application server, you can run this utility from any Web browser having access to the HTTP server's programs.

Running this utility creates or updates:

- The initialization file containing Web application settings.
- The requester program for your Web application.

The Panther Web application server supports three types of requester programs:

- CGI (Common Gateway Interface)—The default name is `proweb` or, for Windows, `proweb.exe`.
- ISAPI (Microsoft's Internet Information Server API)—The default name is `proweb.isa`.
- NSAPI (Netscape's Web Server API)—The default name is `proweb.nsa`.

For a description of the requester process, refer to [page 2-2](#), “Requester Program.”

For more information about settings in your web initialization file, refer to Chapter 12, “Web Initialization Options.”

---

# Using the Web Setup Manager

---

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

## Creating a Web Application

In order to use this utility, you must have completed the installation of the Panther web application server and know the following information:

- The location of your Panther web installation.
- The location of your HTTP server's program directory (usually called cgi-bin or scripts).

**Note:** For the O'Reilly HTTP server, use the directory specified for Standard CGI in the O'Reilly Administration dialog. Typically, this is cgi-bin or scripts, not cgi-win.

For information on setting up your web application server, refer to Chapter 2, “Web Application Setup.”

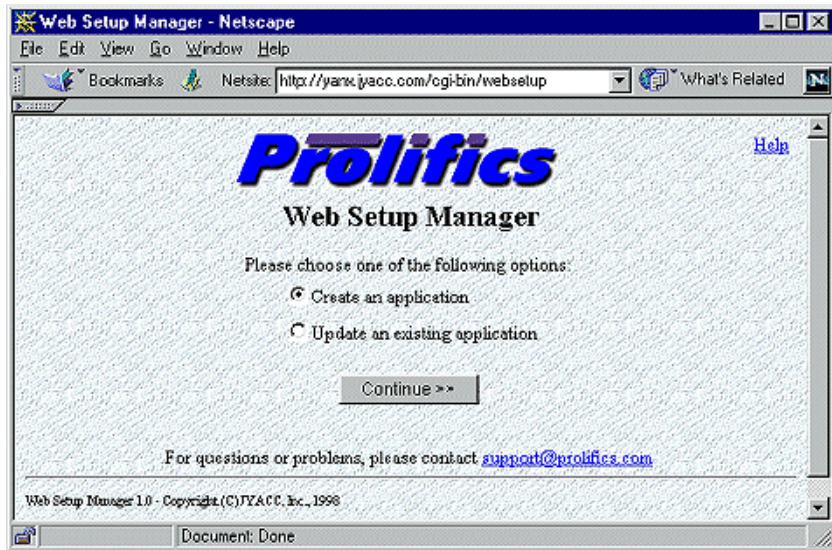
## To create a new Panther Web application:

1. Start your web browser (for example, MS Internet Explorer, Mozilla Firefox).
2. In the URL field, type the location of the Panther Web Setup Manager:

`http://hostMachineName/webProgramDirectory/websetup`

(For example, `http://myhost.com/cgi-bin/websetup`).



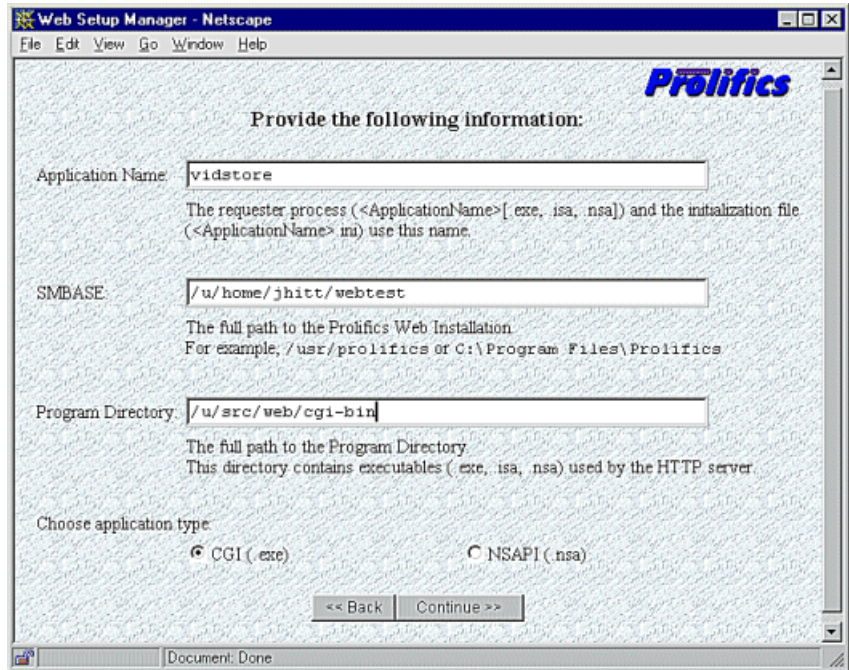


By default, Create an Application is already selected

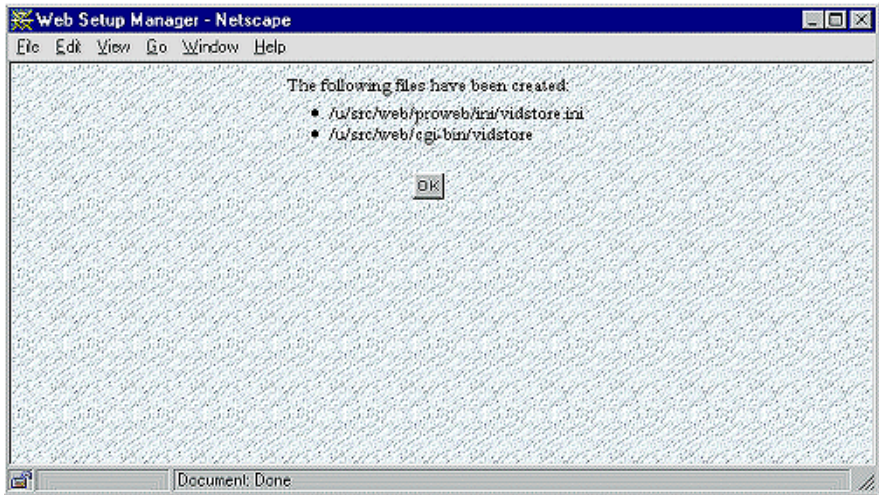
3. Choose Continue.

Enter  
Program  
Locations

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

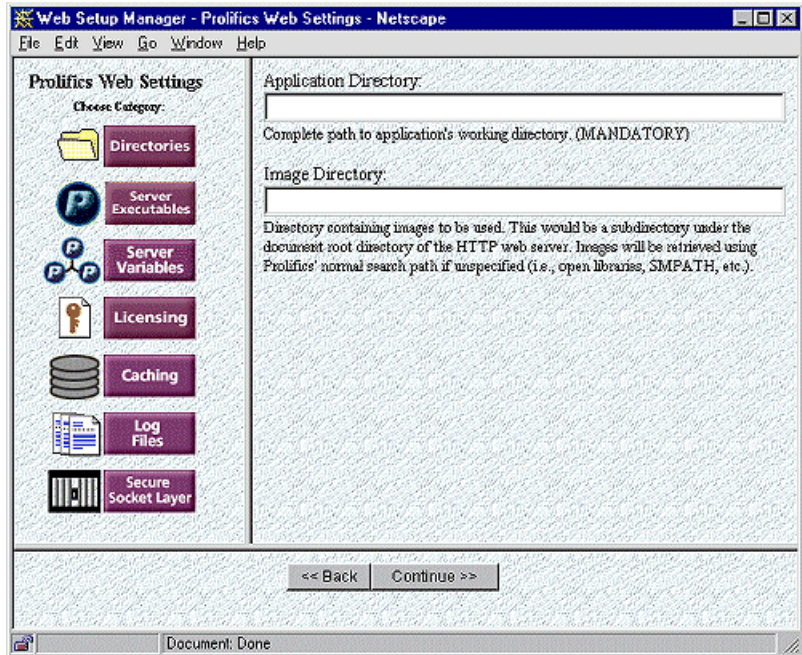


4. For Application Name, enter the unique name you are using to identify your Web application.
5. For **SMBASE**, enter the full path pointing to the Panther Web installation.
6. For Program Directory, enter the path to your HTTP server's program directory.  
**Note:** This should be a directory path, not a URL.
7. Select the web application type: CGI, NSAPI, ISAPI.  
**Note:** All HTTP servers support CGI. Check with your HTTP server administrator to find out if NSAPI or ISAPI support is available.
8. Choose Continue.

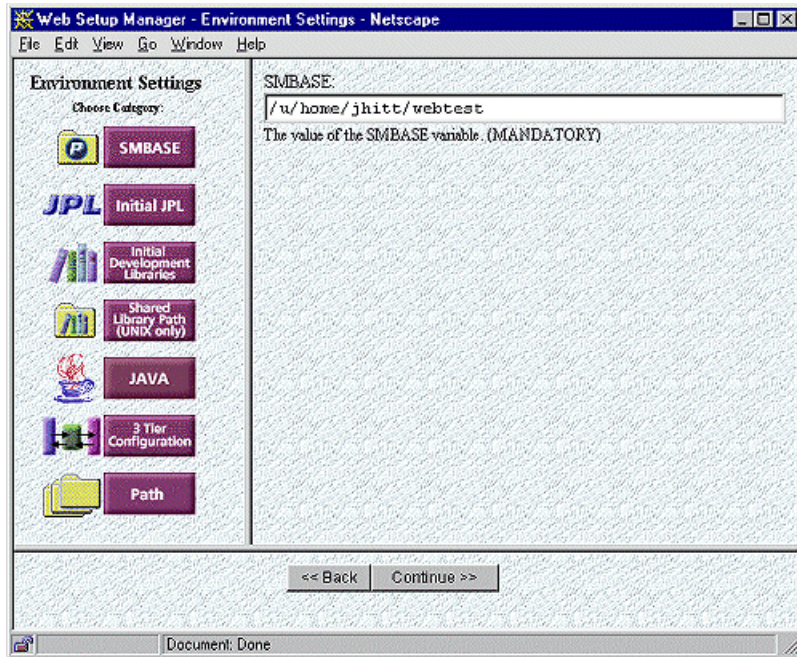


The program creates both a `.ini` file and an executable specific to your application—in the sample Windows screen, `vidstore.ini` and the executable `vidstore.exe`). The next screen will tell you that both files have been created.

9. Choose OK.



Enter Each Panther web application can have its own settings which are stored in the Panther initialization file. In the left-hand frame, icons represent the different categories of Web Settings. In the right-hand frame, there are fields to enter your settings with an accompanying explanation of each setting.



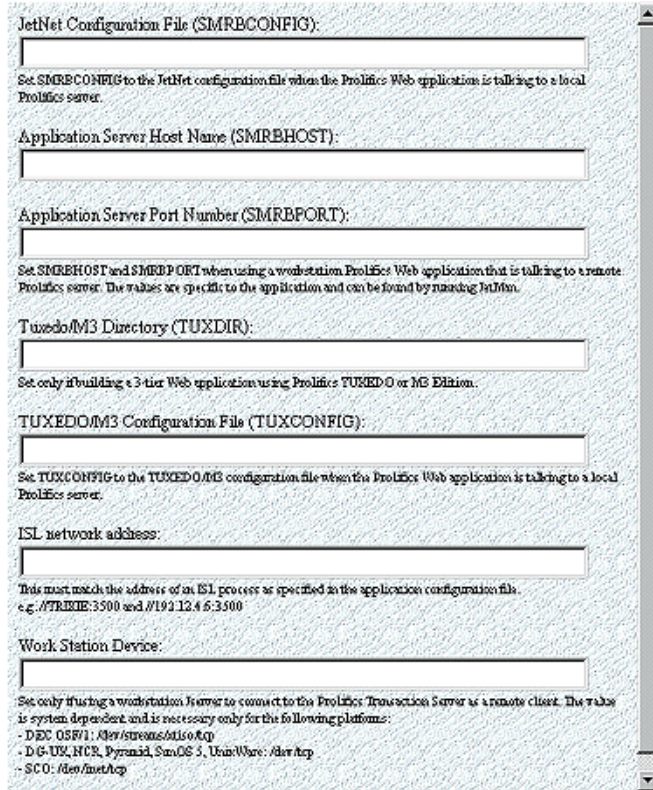
10. Settings which you should set or double-check include:

- Under Directories, in Application Directory, enter the path where the application libraries will be located.
- Under Server Executables, you can change the path to the jserver and dispatcher executables if they are not in the util directory of the Panther web application installation.
- Under Server Variables, set Number of Servers to the number of concurrent users. Since this setting greatly affects performance, you need to set it for each web application.
- Under Licensing, check the path of your license.dat file.
- Under Caching, a series of variables set the data caching options. If you do not know which options will be used, leave the defaults when first creating your web application and update them later.
- Under Secure Socket Layer, enter the host IDs needed for the HTTPS protocol.

11. Choose Continue.

Specify Environment Settings

Your Panther web application also needs its own environment settings.



12. Settings which you should set or double-check include:

- Under **SMBASE**, the location of the Panther web application server installation (entered in the first screen) is automatically displayed.
- Under Initial JPL, enter the name of the JPL file that will be run when the web application server starts. This JPL file should be in an application library that is available to the web application server.
- Under Initial Development Libraries, in **SMFLIBS**, set or check the settings for client libraries, for example `client.lib|vidstore.lib`.
- (UNIX only) Under Shared Library Path, the web application server's `lib` directory should be added to the shared libraries.

## Configure Three-Tier Applications

(For JetNet and Oracle Tuxedo applications) Depending on whether you are running the Panther web application server on a separate machine from your Panther application server, your settings under 3 Tier Configuration will be different.

13. Choose 3 Tier Configuration from the left-hand menu.

*(If running Panther application server and Panther web application server on the same machine):*

In JetNet Configuration File ([SMRBCONFIG](#)), type the directory location of your `broker.bin` file. (It is generally located in the application directory.) Leave the lines for [SMRBHOST](#) and [SMRBPORT](#) blank.

*(If running the servers on different machines):*

Leave the line for JetNet Configuration File ([SMRBCONFIG](#)) blank. Enter the information for [SMRBHOST](#) and for [SMRBPORT](#).

Under Work Station Device, check the platform list and enter the location of the tcp directory if needed.

**Web Setup Manager - Additional Settings - Netscape**

File Edit View Go Window Help

**Prolifics**

**Additional Environment Variables**

Specify any environment variables that may be needed in the running of your application. e.g. database environment variables (ORACLE\_HOME, SYBASE, INFORMIXDIR, etc)

Setting	Value

<< Back    Continue >>

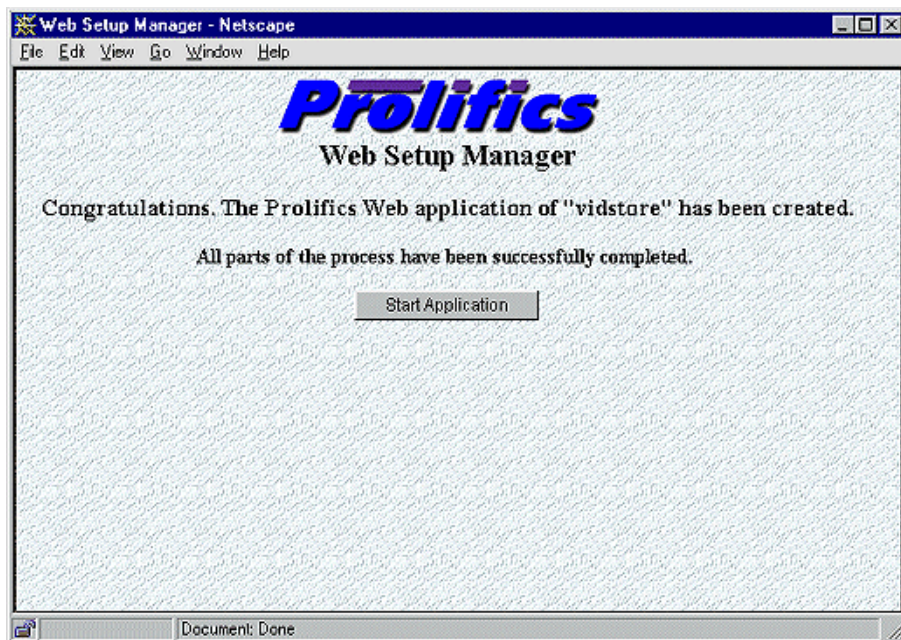
Document: Done

14. Choose Continue.

Specify  
Additional  
Settings

Under Additional Environment Variables, you can enter settings for databases or other optional web settings.

15. For database settings, use an Oracle database as an example. Enter the environment variable name under the Setting column (for example, ORACLE\_HOME) and enter its value in the Value column (for example, /usr/oracle).
16. For JetNet and Oracle Tuxedo applications, specify access to the JIF file on this screen. Under Setting, enter the environment variable `SMTPJIF`; under Value, enter the name of the JIF file located in an application library (for example, `jif.bin`).
17. After completing any necessary information, choose Continue. A screen informs you that your Panther Web application server file has been created.





# C Setting Up an NSAPI Web Server

Panther web applications support NSAPI for HTTP processes.

---

## Configuring Your NSAPI-Compliant Server

---

In the following instructions, the HTTP server is installed at `/usr/netscape/suitespot`, and the `cgi-bin` directory is `/usr/netscape/suitespot/cgi-bin/`.

In order to install Panther NSAPI support:

1. Copy `prlldr.nsa` to `/usr/netscape/suitespot/cgi-bin/`.
2. Modify the file `/usr/netscape/suitespot/http*/config/obj.conf` as follows:
3. At the top of the file, add:

```
Init fn="load-modules" \  
    shlib="/usr/netscape/suitespot/cgi-bin/prlldr.nsa" \  
    funcs="prolifics,prolifics-init" \  
Init fn="prolifics-init"
```

4. After the line `<Object name=default>`, add the following preceding any line similar to `NameTrans fn="pfx2dir" ... from="/":`

```
NameTrans fn="pfx2dir" \  
    from="/prolifics"\  
    dir="/usr/netscape/suitespot/cgi-bin" \  
    name="prolifics"
```

5. After the line `</Object>`, add:

```
<Object name=prolifics\  
    ObjectType fn=force-type type=magnus-internal/prolifics  
    Service fn=prolifics  
</Object>
```

6. The file `/usr/netscape/suitespot/http*/config/mime.types` must also be modified. After the line `type=magnus-internal/cgi exts=cgi,exe,bat`, add:

```
type=magnus-internal/prolifics exts=nsa
```

7. After the edits are complete, restart your Netscape server.

---

## Accessing the Panther Web Application

---

To access the Panther web application using NSAPI:

- Copy `proweb.nsa` to the program directory (for example, `cgi-bin`).
- For each Panther web application, copy `proweb.nsa` to `AppName.nsa`.
- Start the application either as a service or with the `monitor` command. The syntax for the `monitor` command is:

```
monitor -start AppName
```

The simplest service syntax is:

```
net start AppName
```

- Enter the application's URL:

`http://HostName/prolifics/AppName.nsa/ScreenName`

**Note:** For applications previously deployed using CGI, be sure to update the URLs to access the Panther directory and the application's NSAPI executable.

---

## A Sample Obj.conf File

---

The following obj.conf file illustrates the edits detailed in the previous section.

```
Init fn="load-modules" \  
    shlib="/usr/netscape/suitespot/cgi-bin/prlldr.nsa" \  
    funcs="prolifics,prolifics-init"  
Init fn="prolifics-init"  
  
Init fn=flex-init  
access="/usr/netscape/suitespot/https-myserver/logs/access"  
format.access="%Ses->client.ip% - %Req->vars.auth-user%  
[%SYSDATE%] \" %Req->reqpb.clf-request%\" %Req->srvhdrs.clf-status%  
%Req->srvhdrs.content-length%"  
Init fn=load-types mime-types=mime.types  
  
<Object name=default>  
# the following directive must precede any line similar to  
# NameTrans fn="pfx2dir" ... from="/"\  
NameTrans fn="pfx2dir" \  
    from="/prolifics" \  
    dir="/usr/netscape/suitespot/cgi-bin" \  
    name="prolifics"  
  
NameTrans fn=pfx2dir from=/ns-icons  
dir="/usr/netscape/suitespot/ns-icons"  
NameTrans fn=pfx2dir from=/mc-icons  
dir="/usr/netscape/suitespot/ns-icons"  
NameTrans fn="pfx2dir" from="/help"  
dir="/usr/netscape/suitespot/manual/https/ug"  
  
NameTrans fn=document-root root="/usr/netscape/suitespot/docs"  
PathCheck fn=unix-uri-clean  
PathCheck fn="check-acl" acl="default"  
PathCheck fn=find-pathinfo  
PathCheck fn=find-index index-names="index.html,home.html"
```

```
ObjectType fn=type-by-extension
ObjectType fn=force-type type=text/plain
Service method=(GET|HEAD) type=magnus-internal/imagemap
fn=imagemap
Service method=(GET|HEAD) type=magnus-internal/directory
fn=index-common
Service method=(GET|HEAD) type=~magnus-internal/* fn=send-file
AddLog fn=flex-log name="access"
</Object>

<Object name=prolifics>
  ObjectType fn=force-type type=magnus-internal/prolifics
  Service fn=prolifics
</Object>

<Object name=cgi>
  ObjectType fn=force-type type=magnus-internal/cgi
  Service fn=send-cgi
/Objct>
```

# D Using Java Servlets

A Panther web application can run as a Java servlet. For this feature, you need a servlet engine using Java Virtual Machine Version 1.1.5 or later with native thread support enabled.

Java servlets extend the functionality of a Java-enabled HTTP servers. They are server-side components which interact with servlet engines running on HTTP servers through requests and responses. For example, a client program running on a web browser sends a request to an HTTP server. This request is processed by the servlet engine that runs with the HTTP server. The HTTP server returns a response to the servlet which in turn sends a response in HTTP format to the client.

Java servlets are an alternative to CGI programs and to vendor-specific APIs, such as NSAPI or ISAPI. By being both platform-independent and threaded, Java servlets have advantages over other protocols.

---

## Installing Java Servlet Support

---

The files for Java servlet support are located at `WebInstallDir/servlet`, as in `/usr/panther/servlet` on UNIX or `C:\Prolifics\Panther\Servlet` on Windows.

The `servlet` directory of your Panther web installation contains the following files:

- `proweb.jar`—servlet class files
- `proweb.java`—sample source code

- `filtered.java`—sample source code
- `prowebjni.dll`—servlet native code (Windows only)
- `libprowebjni.so`—servlet native code (UNIX only)
- `*.html`—HTML documentation of Java classes and usage instructions

### **To install Java servlet support:**

1. Add the location of `proweb.jar` to the servlet engine's `CLASSPATH`. You must specify the full path.
2. Add the directory containing the file `prowebjni.dll` (`libprowebjni.so` on UNIX) to the servlet engine's `PATH`. Alternatively, on Windows you could copy `prowebjni.dll` to the Windows directory.
3. Restart the servlet engine.

---

# **Accessing the Panther Web Application**

---

### **To access the Panther web application using Java servlets:**

1. Start the Panther web application. Access the application using the following URL:

`http://HostName/proweb/WebAppName/ScreenName`

The following URL accesses the `dstord.scr` screen in the `vidstore` web application:

`http://myhost.com/proweb/vidstore/dstord.scr`

---

# Panther's Java Servlet Classes

---

Panther's Java servlet implementation extends the `javax.servlet` and the `javax.servlet.http` packages in the Java Servlet API.

The Java classes in `proweb.jar` include:

- `ProlificsHttpServletRequest`
- `FilterHttpServletRequest`
- `FilterHttpServletResponse`
- `FilterServletInputStream`
- `FilterServletOutputStream`

## Methods

For more information on the following methods, refer to the Java Servlet API documentation at <http://java.sun.com> and to the HTML version of the Java class files.

### Panther HTTP Servlet

The following methods are in `ProlificsHttpServletRequest`:

#### `doGet`

Passes the `GET` request to a Panther application.

```
public void doGet(HttpServletRequest req,
                 HttpServletResponse res)
    throws ServletException, IOException;
public void doGet(HttpServletRequest req,
                 HttpServletResponse res, String appname,
                 Boolean chunked)
    throws ServletException, IOException;
```

#### `doPost`

Passes the `POST` request to a Panther application.

```
public void doPost(HttpServletRequest req,
                  HttpServletResponse res)
    throws ServletException, IOException;
```

```
public void doPost(HttpServletRequest req,
                   HttpServletResponse res, String appname,
                   Boolean chunked)
    throws ServletException, IOException;
```

getServletInfo

Describes the servlet.

```
public String getServletInfo();init
```

If specified, it overrides the `init` method in the `GenericServlet` class which the servlet engine calls when the servlet is loaded.

```
public void init(ServletConfig config)
```

throws `ServletException`;

## Servlet Requests

The following methods in the `HttpServletRequest` interface are used by `ProlificsHttpServletRequest`:

getAuthType

Gets the authentication scheme of this request. Same as the CGI variable `AUTH_TYPE`.

```
public String getAuthType();
```

getContentLength

Returns the size of the request entity data, or -1 if not known. Same as the CGI variable `CONTENT_LENGTH`.

```
public int getContentLength();
```

getContentType()

Returns the Internet Media Type of the request entity data, or null if not known. Same as the CGI variable `CONTENT_TYPE`.

```
public String getContentType();
```

getCookies

Gets the array of cookies found in this request.

```
public Cookie[] getCookies();
```

getHeader

Gets the value of the requested header field of this request.

```
public String getHeader(String name);
```

getHeaderNames

Gets the header names for this request.

```
public Enumeration getHeaderNames();
```



`getInputStream`

Returns an input stream for reading binary data in the request body.

```
public ServletInputStream getInputStream()  
    throws IOException;
```

`getMethod`

Gets the HTTP method (for example, GET, POST, PUT) with which this request was made. Same as the CGI variable `REQUEST_METHOD`.

```
public String getMethod();
```

`getPathInfo`

Gets any optional extra path information following the servlet path of this request's URI, but immediately preceding its query string. Same as the CGI variable `PATH_INFO`.

```
public String getPathInfo();
```

`getPathTranslated`

Gets any optional extra path information following the servlet path of this request's URI, but immediately preceding its query string, and translates it to a real path. Similar to the CGI variable `PATH_TRANSLATED`.

```
public String getPathTranslated();
```

`getProtocol`

Returns the protocol and version of the request as a string in the form *protocol/major version.minor version*. Same as the CGI variable `SERVER_PROTOCOL`.

```
public String getProtocol();
```

`getQueryString`

Gets any query string that is part of the HTTP request URI. Same as the CGI variable `QUERY_STRING`.

```
public String getQueryString();
```

`getRemoteAddr`

Returns the IP address of the agent that sent the request. Same as the CGI variable `REMOTE_ADDR`.

```
public String getRemoteAddr();
```

`getRemoteHost`

Returns the fully qualified host name of the agent that sent the request. Same as the CGI variable `REMOTE_HOST`.

```
public String getRemoteHost();
```

`getRemoteUser`

Gets the name of the user making this request. The user name is set with HTTP authentication. Whether the user name will continue to be sent with each subsequent communication is browser-dependent. Same as the CGI variable `REMOTE_USER`.

```
public String getRemoteUser();
```

`getRequestURI`

Gets, from the first line of the HTTP request, the part of this request's URI that is to the left of any query string.

```
public String getRequestURI();
```

`getServerName`

Returns the host name of the server that received the request. Same as the CGI variable `SERVER_NAME`.

```
public String getServerName();
```

`getServerPort`

Returns the port number on which this request was received. Same as the CGI variable `SERVER_PORT`.

```
public int getServerPort();
```

`getServletPath`

Gets the part of this request's URI that refers to the servlet being invoked. Analogous to the CGI variable `SCRIPT_NAME`.

```
public String getServletPath();
```

There are additional methods included in `FilterHttpServletRequest` that are not called by `ProlificsHttpServlet`. See the HTML documentation for more information.

**Servlet Responses**

The following methods in the `HttpServletResponse` interface are used by `ProlificsHttpServlet`:

`addCookie`

Adds the specified cookie to the response. It can be called multiple times to set more than one cookie.

```
public void addCookie(Cookie cookie);
```

`getOutputStream`

Returns an output stream for writing binary response data.

```
public ServletOutputStream getOutputStream()
    throws IOException;

setContentLength
    Sets the content length for this response.

public void setContentLength(int len);

setContentType
    Sets the content type for this response.

    public void setContentType(String type);

setHeader
    Adds a field to the response header with the given name and value.

    public void setHeader(String name, String value);

setStatus
    Sets the status code, or the status code and message, for this response.

    public void setStatus(int sc);
    public void setStatus(int sc, String sm);
```

There are additional methods included in `FilterHttpServletResponse` that are not called by `ProlificsHttpServlet`. See the [HTML documentation](#) for more information.

## Filter Servlet Input Stream

The following methods, if specified, override the methods in `InputStream`:

```
available
    public int available() throws IOException;

close
    public void close() throws IOException;

mark
    public synchronized void mark(int readlimit);

markSupported
    public boolean markSupported();

read
    public int read() throws IOException;
    public int read(byte buf[]) throws IOException;
    public int read(byte buf[], int off, int len)
        throws IOException;

readLine
    public int readLine(byte buf[], int off, int len)
        throws IOException;
```

reset	<code>public synchronized void reset() throws IOException;</code>
<b>Filter Servlet Output Stream</b>	The following methods, if specified, override the methods in <code>OutputStream</code> or <code>ServletOutputStream</code> . See the <b>HTML</b> version of the class file for more information.
close	<code>public void close() throws IOException;</code>
flush	<code>public void flush() throws IOException;</code>
print	<code>public void print(boolean bval) throws IOException;</code> <code>public void print(char cval) throws IOException;</code> <code>public void print(double dval) throws IOException;</code> <code>public void print(float fval) throws IOException;</code> <code>public void print(int ival) throws IOException;</code> <code>public void print(long lval) throws IOException;</code> <code>public void print(String sval) throws IOException;</code>
println	<code>public void println() throws IOException;</code> <code>public void println(boolean bval) throws IOException;</code> <code>public void println(char cval) throws IOException;</code> <code>public void println(double dval) throws IOException;</code> <code>public void println(float fval) throws IOException;</code> <code>public void println(int ival) throws IOException;</code> <code>public void println(long lval) throws IOException;</code>
write	<code>public void write(byte buf[]) throws IOException;</code> <code>public void write(int val) throws IOException;</code> <code>public void write(byte buf[], int off, int len) throws IOException;</code>

# E Sample Web Applications

The Panther Web Gallery contains sample applications that demonstrate a variety of HTML and Panther features. You can view the applications in the Panther Web Gallery or in the editor.

To visit the Panther Web Gallery, use the following URL:

```
http://server-name/cgi-directory/jwsamp[.exe]/main
```

You can view these applications in the editor to see how they were created. Locate the sample application files in the appropriate subdirectory under your Panther samples subdirectory (in the Web application server home directory). For example, the Java applet application is located in ... web/samples/java, where ... web is the web application server home directory. Full directory information for each application is provided in the descriptions that follow.

---

## General Applications

---

The following applications incorporate a variety of HTML and Panther features that you might find in a typical application:

Tracker (package tracking)

Shows how easily the world can be linked together on the World Wide Web. This application lets you track a package that is shipped via a popular carrier such as Federal Express or UPS.

For example, suppose your software company has deployed this package to track its software shipments. To locate a package: first, select the customer to whom the package is being shipped; a database query returns the shipping information for the selected customer. Then click on the waybill number for the package that you're interested in, and the application calls the carrier's tracking program and passes the waybill number as a parameter.

Directory: ... `web/samples/shipping`

Search the world (search engine)

Pass values between Panther screens and HTML forms. You can use this application to search the Internet for information by entering a search string. To search the Internet, enter your search string and click the appropriate push button; the application passes your search string to a second HTML form which calls several popular search engines. The search string from the main form is passed to the second form.

Directory: ... `web/samples/search`

Email of the famous (mailer)

Incorporate your existing database of URL addresses into an application through `sm_web_invoke_url`. You can use this application to query your URL database and create a dynamic hyperlink to a selected address.

Directory: ... `web/samples/dblinks`

Fly by Web (airline reservations)

Web applications can use three-tier processing in order to off-load business logic onto the Panther application server. This sample shows how your browser and Web application server act as a client to the Panther application server.

Directory: ... `web/samples/3tier`

---

# Feature-Specific Examples

---

The following examples show how to use specific HTML or Panther features:

## Action and selection in grids

Use list boxes to invoke an action or allow a selection. There are two types of list boxes in a Panther Web application: Action and Selection. An Action List Box has an action or event associated with each item in the list. The action is triggered when the item is selected. A Selection List Box allows the user to select any number of items from the list.

This example demonstrates both action and selection list boxes. Both types of list boxes have been put into grids. The master grid uses the properties of an action list box. When you select an item (a customer's name) in the master grid, the detail grid is populated with the appropriate data (the film videos that the customer has rented). The detail grid has the properties of a selection list box. In this grid, you can select all of the film videos that are to be returned. (Press the save button to update the database after your selection.)

Directory: ... web/samples/listbox

## 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.

Directory: ... web/samples/activex

## Automatically generated JavaScript

Dynamically generate JavaScript to control field edits and validations in order to make your Web application more intelligent. This example demonstrates some of the dynamically generated JavaScript functions that are used to valid field input.

Directory: ... web/samples/jamjavas

#### Business graphics

Generate a graphical representation of the data from your database or any other source. You can display graphs and charts based on your data in the following formats: pie chart, bar/line graph, XY plot, High/Low chart. In this example, a graph widget is used to generate both a pie chart and a bar chart.

Directory: ... `web/samples/busgraph`

#### CGI variables

Access all available CGI variables in JPL. These values can be used to determine which server or browser type is being used, whether a GET or a POST has occurred, and other important information.

You can view the values of this application's CGI (Common Gateway Interface) variables. (Press the Submit button to view the CGI variables after a POST has occurred.)

Directory: ... `web/samples/cgi`

#### Client cookies

Store values in the browser using cookies.

The first time you access this sample, the user name and number of visits are set as cookies in the browser. Each subsequent visit to this sample increments the number of visits and displays the user name. You can reset the user name which also resets the number of visits to 0.

Directory: ... `web/samples/cookie`

#### Custom JavaScript

Embed custom JavaScript into a Panther application. JavaScript allows you to do field or screen processing on the client without going back to the server. For instance, you can do calculations, give warnings, manipulate data, use timer functions, or do field validations.

In this example, you can view asynchronously changing clocks that compute the time in New York, London, and Tokyo.

Directory: ... `web/samples/javascr`

#### Data transfer (send and receive)

Use the JPL send and receive commands to easily pass data back and forth between screens. The movie search criteria which you provide is passed to a second screen, where the data is used for a query by example search on the database.



Directory: ... web/samples/sendrec

#### Expandable grids

Display tabular data in expandable grids, which dynamically expand to the number of rows selected from the database.

The grid in this sample expands according to the number of rows returned from the database. You can select all video titles, titles beginning with a certain letter, or the titles in a specific classification. If you want detailed data about the video, click on the `title_id`, and the information is displayed at the bottom of the screen.

Directory: ... web/samples/expgrid

#### Frames

Divide an HTML frameset into two frames: an index and contents display window. The WEB application in the index frame controls the HTML documents that are displayed in the contents window.

You can open a new Web page or HTML file in the contents (right-side) window frame by selecting the topic name in the index (left-side) window.

Directory: ... web/samples/frames

#### Graphics

Use graphics to enhance your WEB applications. The application screen includes a transparent image, an animated image, an image as hyperlink, and an image map of the United States that allows you to select a region and jump to a new site to view information about the selected region.

An image map is a single graphic with hot spots that link to other locations. This example uses an NCSA standard image map. It assumes that the HTTP server's image map CGI program is available in the CGI directory. (If your server does not support this configuration, then this image map can not function properly.) To enable the image map, copy

...web/samples/graphics/usa.map to your DOCROOT directory.

Directory: ... web/samples/graphics

#### HTTP request method (K\_WEBPOST)

Use the two methods by which a Web browser can transfer data to an HTTP server, GET and POST. GET is used when the browser is requesting data or submitting a limited amount of data.

POST is used when the browser is submitting large amounts of data. The Panther flag `K_WEBPOST` enables the Application Server program to determine if the incoming request is a POST.

In the sample, the flag `K_WEBPOST` is queried to prevent a second connection to the database after the initial connection has been established.

You can use this application to insert, update, or delete data from a database.

Directory: ... `web/samples/kflag`

### Hyperlinks

Perform a variety of functions on your Web page using hyperlinks. They bring up documents, target these documents for a new instance of the browser, send mail to people, pass along input values, and perform other activities. This example demonstrates several different types of hyperlinks.

Directory: ... `web/samples/hyperlnk`

### Java applets

Build more robust applications using Java applets, and incorporate those Java applets into your Web applications. This example contains an embedded Java applet (the ticker-tape banner) that uses Panther's property API to dynamically change the banner text being sent to the Java code.

Directory: ... `web/samples/java`

### JPL global variables

Show Panther's power and flexibility by showing how to maintain global information through JPL globals and the Panther function `sm_web_save_global`. You can set your own user and application preferences for a scheduler. (Note: this sample application is not connected to a database.)

Directory: ... `web/samples/globals`

### Master/Detail in frames

Use frames to make your master/detail applications user-friendly.

In this sample, the list of movie titles appears in one frame. Each movie title is a hyperlink so that when a movie title is selected, specific information about that title is displayed in an additional frame.

Directory: ... `web/samples/mdframes`

### Reports

Generate a report based upon user input. You can enter search criteria (a film name) and generate a movie cast report based upon the entered film name. The generated report includes page navigation buttons. You can also generate a movie cast report by selecting the film title from a list of the top ten films. Panther automatically formats the report based upon the amount of data that is being displayed.

Directory: ... web/samples/webwrite

### Scrolling grids

Page through the entries in your database easily with scrolling grids. In this example, you can insert or delete entries into the sample database of video customers or scroll through the list of customers. (Use the vertical arrow buttons at the left or the grid to scroll through the entries; the push buttons at the bottom to insert and delete information.)

Directory: ... web/samples/scroll

### Sound files

Embed sound, video, or other popular MIME file types easily into your Web application. It uses graphic hyperlinks whose URLs are audio files.

If your PC has a sound card installed, you can listen to the sound files by clicking on the appropriate graphics.

Directory: ... web/samples/sound

### 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.

Directory: ... web/samples/template

### Web and GUI applications

Run a single application in any environment—two-tier, three-tier, and Web.

The application property

`in_web` is used to determine whether the application is running in a Web or non-Web environment, and the detail information is populated accordingly.

You can select a film from a grid and view detail information about the film.

Directory: ... web/samples/webflag

# Index

## Symbols

- @cgi global variables [11-1](#)
- @web\_action global variables [5-9](#)
- @web\_image\_click\_x/y [8-17](#)
- @web\_posted\_screen [6-3](#)
- {{}}.HTML template tags [8-5](#)

## A

- Active Pixmap property
  - in web applications [3-8](#)
- ActiveX controls
  - setting the Codebase property [8-27](#)
  - using in Web applications [8-24](#)
- Aliasing
  - Web application fonts [3-17](#)
- Application directory
  - for web applications [2-3](#), [12-4](#)
- Application state
  - saving [6-1](#)
- Attributes property
  - for widgets [8-4](#)
- Auto Expand property [3-11](#)

## B

- Body Attributes property
  - for web screens [8-3](#)
- BODY element [8-1](#)

- setting attributes [8-3](#)

## Browser

- base font [1-4](#)
  - cache settings [2-12](#)
  - specifying events
    - using events [9-2](#)
  - target window [8-13](#)
  - title bar [8-14](#)
  - viewing capabilities [1-4](#)
- Browser caching [6-2](#)
  - BrowserData option [12-5](#)

## C

- Cache files
  - deleting with monitor utility [A-2](#)
- Cached data
  - CacheDirectory option [12-6](#)
  - in HTML file [6-2](#)
  - on HTTP server [6-3](#)
  - retaining [6-3](#), [12-8](#)
- Calculation property
  - in web applications [3-8](#)
- Cascading stylesheets
  - setting stylesheet type [8-19](#)
- CGI (Common Gateway Interface)
  - directory [2-2](#)
  - setting the type of requester executable [2-6](#)
  - setting type of requester executable [2-2](#), [B-1](#)
- Check box widget

- Web application usage 3-10
- Configuration
  - setting web application directory 2-3
  - web initialization file 12-1
- Context global variables
  - saving Web application state 6-6
- Context Web global variables 7-2
- Control flow
  - in HTML templates 8-6
- Cookies
  - in web applications 8-21
  - retrieving values 8-22, 11-3
  - saving Web application state 6-7
- D**
- Data
  - caching
    - in web applications 6-2
- Database
  - connecting to
    - Web application 10-1, 10-2
  - fetching multiple rows in Web application 10-4
  - optimistic locking
    - Web application 10-4
  - transaction processing
    - Web application 10-3
- Database connections
  - setting in Web application 10-1, 10-2
- Dispatcher executable 1-8
  - rereading configuration file A-2
  - setting location of 12-5
- Display Window property 8-14
- Double Click property
  - in web applications 3-8
- E**
- Edit Mask
  - in web applications 3-8
- Error messages
  - for Web applications 2-5
  - logging for Web application server 12-6
- Events
  - web applications 5-1
- Export to HTML property 9-6
- F**
- Firewall 2-12
- Font
  - browser base font 1-4
  - Web application 3-16
    - aliasing 3-17
    - name 3-16
    - point size 3-17
- Form Attributes property 8-3
- FORM element
  - attributes set by Prolifics properties 8-2
- Frames
  - in web application 8-18
- FRAMESET element 8-18
- G**
- GET method
  - submitting a form to another program 8-10
  - using URL to send data 4-2
- Global JPL variable
  - Web application usage 7-1
    - application globals 7-1
    - context globals 7-2
    - setting for user 7-2
    - transient globals 7-3
- Graphics file
  - setting web server location 8-17
  - supported Web browser formats 8-15
  - Web application usage 8-15
- Grid widgets
  - Web application 3-10
    - deleting data 3-11

- expanding 3-11
- inserting data 3-11
- scrolling in browser 3-13
- selecting row 3-12

## H

- HEAD element 8-1
  - adding data for 8-2
- Head Markup property 8-2
- Headings
  - creating in HTML document 8-20
- Hidden property
  - in web applications 3-7
- Hidden widgets
  - generating HTML tag 9-6
  - saving Web application state 6-5
- Horizontal Anchor property 3-14
  - in web applications 3-14
- Horizontal lines
  - creating in HTML document 8-20
- HR element 8-20
- HTML
  - BODY element 8-1
  - components 8-1
  - FORM attributes 8-2
  - FORM element 8-1
  - format 1-3
  - FRAMESET element 8-18
  - generating from screen 8-1
  - generating tags for widgets 9-6
  - HEAD element 8-1
  - headings 8-20
  - horizontal lines 8-20
  - HR element 8-20
  - hyperlinks 8-11
  - modifying with Panther properties 8-2
  - tags 1-3
  - title bar 8-14
- HTML Template property 8-4
- HTML templates

- for Web applications 8-4

HTTP

- variables 11-1

HTTP server 1-3, 1-5

HTTPS protocol 4-6

Hyperlinks

- creating 8-11
- image map 8-16
- in list box widget 8-12
- target window 8-14
- using in reports 8-12

## I

- Image map
  - including server-side map 8-16
- Initialization file
  - for Web 12-1, B-2
- Input Protection property
  - in web applications 3-8
- Ins/Del Buttons property 3-12
- Internet
  - about 1-1
- ISAPI
  - setting type of requester executable B-1

## J

- Java
  - invoking Java applets 8-22
  - using Java servlets D-1
- Java applets
  - invoking 8-22
- Java servlets
  - configuring D-1
- JavaScript
  - accessing HTML names 9-6
  - events 9-2
  - generating from editor properties 9-7
  - including function in web application 9-4
  - referencing screen widgets 9-6

JavaScript property [9-4](#)

JPL variable

    globals in Web application [6-6](#)

    HTTP variables

        defined [11-1](#)

jsviewer executable [1-7](#)

    setting location of [12-5](#)

    setting number of [12-5](#)

## K

K\_WEBPOST

    set on screen posting [5-7](#)

Keep Image Size property [8-16](#)

## L

Length property

    in web applications [3-7](#)

License file

    specifying [12-5](#)

Link Attributes property

    for hyperlinks [8-4](#)

LMLicenseFile [12-5](#)

Log file

    setting for web applications [A-2](#)

## M

Message

    from web applications [A-2](#)

    logging Web client events [12-6](#)

    logging Web server events [12-8](#)

monitor utility

    administering web applications [A-2](#)

    cause dispatcher to reread configuration file  
        [A-2](#)

    deleting expired cache files [A-2](#)

    find Windows service names [A-2](#)

    installing Windows service [A-4](#)

    restarting a web application [A-3](#)

    starting a web application [A-3](#)

    stopping a web application [A-3](#)

## N

NSAPI

    setting type of requester executable [B-1](#)

    setting up web applications [C-1](#)

## O

Optimistic locking

    Web application [10-4](#)

Option menu widget

    populating in Web application [5-8](#)

## P

Password Field property

    in web applications [3-9](#)

Positioning regions

    in web applications [3-15](#)

POST method

    sending data to HTTP server [4-2](#)

Posting screens

    getting name of screen [6-3](#)

Prefix Markup property [8-4](#), [8-20](#)

Push button widget

    Web application usage [3-9](#)

## R

Radio button widget

    Web application usage [3-10](#)

        selecting a grid row [3-12](#)

Reports

    using hyperlinks

        in web applications [8-12](#)

Requester executable [1-7](#)

    location of [2-2](#)

    specifying type [B-1](#)



- Restarting
  - Web application [A-3](#)
- Rows
  - retrieving multiple rows
    - Web application [10-4](#)
- Runtime properties
  - for web applications [3-18](#)
- S**
- Sample applications
  - Panther Gallery
    - web [E-1](#)
- Screen
  - HTML generated from [8-1](#)
  - HTML properties [8-2](#)
  - specifying in a URL [4-2](#)
  - submitting to HTTP server [4-1](#)
  - submitting to secure server [4-6](#)
  - target window [8-14](#)
- Screen properties
  - in web applications [3-1](#)
- Secure POST property [4-6](#)
- Secure server
  - submitting screens to [4-6](#)
- Selection group
  - Web application usage [3-10](#)
- Send data
  - saving Web application state [6-5](#)
- Server caching
  - BrowserData option [12-5](#)
  - web applications [6-3](#)
- Size to Contents property
  - in web applications [3-7](#)
- Sound
  - associating file with widget [8-30](#)
  - including in Web application [8-29](#)
- SSL (Secure Sockets Layer) [4-6](#)
- Starting
  - Web application [A-3](#)
- Status line
  - displaying in web applications [9-3, 9-7](#)
- Status Line Text property [9-7](#)
  - in web applications [3-8](#)
- Stopping
  - Web application [A-3](#)
- Stripe Current Row property [3-12](#)
- Stylesheet Data property [8-19](#)
- Stylesheet Link property [8-19](#)
- Stylesheet Source property [8-3, 8-19](#)
- Stylesheet Type property [8-19](#)
- Stylesheets
  - setting for Web applications [8-19](#)
- Submitting screens
  - getting name of screen [6-3](#)
- Suffix Markup property [8-4, 8-20](#)
- T**
- Target Default property [8-14](#)
- Target property [8-14](#)
- Target window
  - for a screen [8-14](#)
  - for hyperlinks [8-14](#)
- Template
  - HTML [8-4](#)
    - conditional processing [8-7](#)
    - passing database values [8-8](#)
    - submitting a form [8-10](#)
    - template tags [8-5](#)
- Title property [8-14](#)
- Transactions
  - in web applications [10-3](#)
- Transient Web global variables [7-3](#)
- U**
- URL
  - defining parts of [1-2, 4-2](#)
  - encoding parameters [4-3](#)
- Utilities
  - monitor [A-2](#)

## V

- VBScript
  - accessing HTML names 9-6
  - events 9-2
- VBScript property 9-4
- Vertical Anchor property 3-14
  - in web applications 3-14

## W

- Web application server
  - administering A-2
  - components 1-7
  - dispatcher 1-8, 12-5
  - jserver 1-7, 12-5
  - licensing 12-5
  - requester 1-7, 2-2
  - setting number 12-5
- Web applications
  - about 1-5
  - defining stylesheets 8-19
  - initialization file B-2
    - options 12-1
    - sample 12-9
  - installing as Windows service A-2
  - listing Windows services A-2
  - loading graphics 8-17
  - removing as Windows service A-2
  - runtime properties 3-18
  - samples E-1
  - setting browser events 9-2
  - setting up
    - Java servlets D-1
    - NSAPI C-1
    - using the Web Setup Manager B-1
  - starting with monitor utility A-3
  - stopping A-3
  - using ActiveX controls 8-24
    - setting the Codebase property 8-27
  - using HTML templates 8-4

- writing error logs 2-5
- Web events
  - application shutdown 5-3
  - application startup 5-1
  - context flags on web entry 5-9
  - hook procedures 5-1
  - in the browser 9-2
  - pre-HTML generation 5-2
  - screen posting 5-2
  - sequence 5-5
- Web Options properties
  - screen 3-3
  - widgets 3-5
- Widgets
  - accessing HTML names 9-6
  - exporting to HTML 9-6
  - HTML attribute properties 8-3
  - overlapping
    - in web applications 3-14
  - positioning properties 3-14
- Windows
  - servers
    - configuring 2-8
    - installing as a service A-4
- Windows service
  - finding service names A-2
  - installing Web application as A-2
  - removing Web application A-2
- Word Wrap property
  - in web applications 3-9