Open Tools from Sybase, Inc.

PowerBuilder

Building Internet Applications with PowerBuilder

Version 6

PowerBuilder®

AB0879

October 1997

Contents

About This Book

Subject

This book provides instructions for using the Internet Tools supplied with PowerBuilder to create various types of Web applications.

Audience

This book is for anyone who will be using the Internet Tools with PowerBuilder to build Web applications. It assumes that:

- You have a solid understanding of how to build applications with PowerBuilder. If not, read the *Feature Guide* and *Application Techniques* before you read this book.
- You are familiar with the techniques for building distributed applications with PowerBuilder. If not, read the part on distributed application techniques in *Application Techniques* before you read this book.

PART 1

Getting Started

This part introduces the Internet Tools supplied with PowerBuilder and describes certain setup tasks required to use the Internet Tools.

CHAPTER 1

Introduction to the Internet Tools

About this chapter

This chapter introduces the Internet Tools supplied in PowerBuilder and explains how its components can help you build various types of Web applications. It also describes setup tasks you may need to perform after installing the Internet Tools.

Contents

About the Internet Tools

What the Internet Tools are

The Internet Tools are a set of software components for building PowerBuilder applications for the World Wide Web. The Internet Tools are part of PowerBuilder. Building on PowerBuilder's database access technology and scripting language, the Internet Tools bring the strength and style of PowerBuilder to the Internet.

You can use one or more of the Internet Tools components to build your application, depending on your development strategy and operating system platform. (Different PowerBuilder platforms support different Internet Tools.)

FOR INFO For guidelines to consider when determining your development strategy, see "Choosing which Internet Tools to use" on page 8.

Components and supported platforms

This table describes each Internet Tools component and lists the PowerBuilder platforms on which it is supported:

Component	Platforms	What it does
Web.PB	Windows NT Windows 95 Solaris HP-UX AIX Power Macintosh	Brings the distributed computing capabilities of PowerBuilder to the World Wide Web: ♦ Enables Web browsers to invoke the services of distributed objects ♦ Enables HTML documents to access centralized business logic stored on application servers ♦ Enables Web pages to display live database information
Web.PB class library	Windows NT Windows 95	Helps you develop the PowerBuilder server application Includes custom class user objects that provide functions for generating HTML and for managing the state of a browser connection
Web.PB Wizard	Windows NT Windows 95 Solaris HP-UX AIX Power Macintosh	Helps you create the HTML elements required to invoke the services of distributed objects
Web.PB examples	Windows NT Windows 95	Provides code examples that help you learn how to use Web.PB

Component	Platforms	What it does
PowerBuilder window plug-in (standard)	Windows NT Windows 95 Power Macintosh	Displays a PowerBuilder window in an HTML page Runs a PowerBuilder application in a browser on the client workstation. It works with any browser that supports Netscape plug-ins, including Netscape Navigator and Microsoft Internet Explorer
PowerBuilder window plug-in (secure)	Windows NT Windows 95 Power Macintosh	Provides a secure version of the PowerBuilder window plug-in Ensures that PowerBuilder applications downloaded over the Internet will not damage a client system or access information on a client workstation
DataWindow plug-in	Windows NT Windows 95 Power Macintosh	Displays a Powersoft report in an HTML page Works with any browser that supports Netscape plug-ins, including Netscape Navigator and Microsoft Internet Explorer
PowerBuilder window ActiveX	Windows NT Windows 95	Lets you provide a graphical interface inside HTML pages when using a Web browser that supports ActiveX: ♦ Provides all capabilities of the window and DataWindow plug-ins, plus the ability to use JavaScript or VBScript to access a subset of a PowerBuilder child window's events and functions ♦ Includes methods you can call to invoke functions and events in the child window contained in the window ActiveX control
WebSite Web server software	Windows NT Windows 95	Provides a complete 32-bit, multithreaded environment for creating, managing, and administering a Web server on the Windows platform FOR INFO For more information about WebSite, see the HTML files provided with WebSite

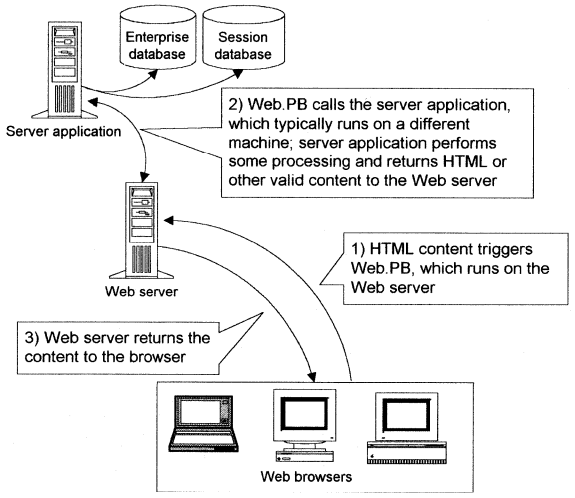
How the Internet Tools and Internet architecture interact

An Internet application begins with two-way communication between a client Web browser and a Web server. The client requests an HTML document and the server delivers it. Elements (or tags) in the document may refer to other resources, such as images, sound files, plug-ins, ActiveX controls, or programs. When the client encounters these elements, it makes additional requests to the Web server. Some program resources, such as plug-ins and ActiveX controls, may require additional software for the client to use the resources.

How Web.PB fits in An element in an HTML document can specify that the Web server run a program instead of downloading a file. That program, which runs on the Web server, generates data that it returns to the client, or calls another program running on the same or another computer. This is how Web.PB works.

When an HTML document runs Web.PB, the Web.PB program calls a PowerBuilder server application. The PowerBuilder server application listens for requests from clients (in this case Web.PB), processes those requests, and returns information.

The Web server receives the information and sends it to the Web browser client.



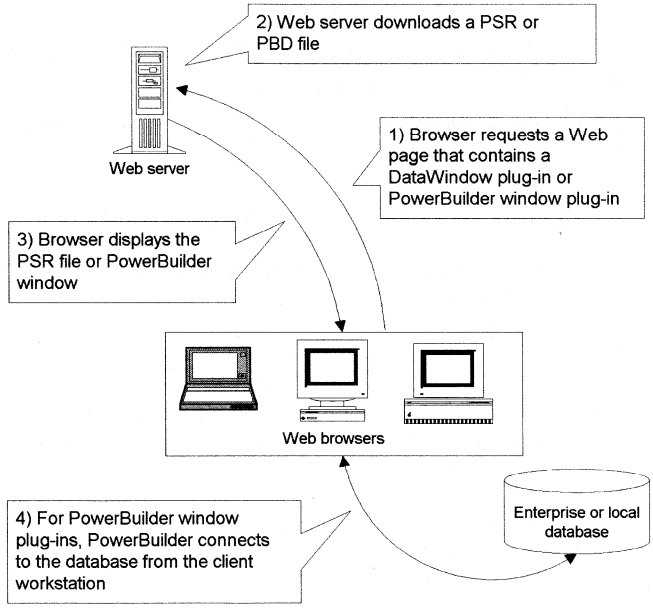
How the plug-ins fit in The Internet Tools include two plug-ins for Web browsers:

- **DataWindow plug-in** Displays Powersoft reports

- **PowerBuilder window plug-in** Runs a PowerBuilder application on the client

Files that you have stored on the Web server provide the data for the plug-ins:

- **For the DataWindow plug-in** A Powersoft report (PSR) file contains the report that will be displayed (the PSR file contains data and formatting information for the report)
- **For the PowerBuilder window plug-in** A PowerBuilder dynamic library (PBD) file contains the objects for the PowerBuilder application, including the child window that starts the application



Choosing which Internet Tools to use

Before you begin using the Internet Tools to build a Web application, you need to design the application. This section gives some general guidelines you can use to choose which components best suit your application design.

Things to consider

As you design your Web application, you need to answers these questions:

- What browsers will your clients be using?
- How much software can you require on the client workstation?
- Where do you want the processing to take place: on an application server or on the client?
- Does the application require database access or can reports be prepared ahead of time and delivered on request?
- Should database access be available from the application server or from the client workstation?

Your strategy

Your answers to these questions will determine your strategy.

How the Internet Tools can help

The Internet Tools include components that are useful in building various kinds of Web applications. Which components you use depends on which application development strategy you adopt.

The following table outlines the tradeoffs you make by using each of the major Internet Tools components. By carefully evaluating the tradeoffs before you begin development, you can ensure that the strategy you adopt best suits your needs.

Component	Characteristics
Web.PB	**PowerBuilder application on server** When you use Web.PB to deploy a Web application, most of the processing takes place in the PowerBuilder server application. The server application executes business logic to retrieve information requested by the Web browser. If necessary, the server application connects to the database to access data **Thin client** The browser client requires no extra software **Works with all browsers** The application works with any commercially available browser **Server database access** Database access is centralized on the application server **Displays HTML** The application server formats its results as an HTML page or other valid content type, which Web.PB sends to the client
PowerBuilder window plug-in	**PowerBuilder application on client** An application in the PowerBuilder window plug-in can execute most PowerBuilder functionality as long as the application begins with a child window **Fat client** The browser client requires the PowerBuilder runtime DLLs or shared libraries and the PowerBuilder window plug-in DLL or shared library **Client database access** Database access is initiated on the client workstation using the client's connections **Runs PowerBuilder windows** The application can run any PowerBuilder windows, display information, accept data input, and update databases

Component	Characteristics
PowerBuilder window ActiveX	**PowerBuilder application on client** An application in the PowerBuilder window ActiveX can execute most PowerBuilder functionality as long as the application begins with a child window. Additionally, the browser can interact with the child window through either JavaScript or VBScript **Fat client** The browser client requires the PowerBuilder runtime DLLs or shared libraries and the window ActiveX **Client database access** Database access is initiated on the client workstation using the client's connections **Runs PowerBuilder windows** The application can run any PowerBuilder windows, display information, accept data input, and update databases
DataWindow plug-in	**PowerBuilder reports on client** The DataWindow plug-in displays reports that have been generated previously and stored on the Web server **Some client software** Browser client requires the DataWindow plug-in DLL or shared library **No database access** Because the PSR files have been generated previously, database access is not necessary **Displays read-only data** The user can view, print, and save the reports

Setting up for the Internet Tools

You can install one or more of the Internet Tools by running the PowerBuilder installation program.

FOR INFO For information about using the PowerBuilder installation program, see the *PowerBuilder Installation Guide*.

This section describes setup tasks that you may need to perform *after* installing the Internet Tools.

Setting up SQL Anywhere and the Powersoft Demo Database

On Windows, you need SQL Anywhere and the Powersoft Demo Database installed to do the Web.PB tutorial, and you also need SQL Anywhere installed to use the Web.PB Class Library for session and transaction management.

FOR INFO For information on whether your version of PowerBuilder includes SQL Anywhere and how to install the Powersoft Demo Database on your platform, see the *PowerBuilder Installation Guide*.

Setting up WebSite

On Windows, you can choose to install WebSite Web Server Version 1.1 from O'Reilly & Associates when you run the PowerBuilder Setup program.

Why install WebSite

Installing WebSite is optional, but recommended so that learning to use the Internet Tools and testing your applications will be easier. If you want to do the Web.PB tutorial, you *must* install WebSite.

FOR INFO For information about installing WebSite as part of the PowerBuilder Setup program, see the *PowerBuilder Installation Guide*.

WebSite documentation

The WebSite directory installed on your machine contains its own HTML documentation in the \WSDOCS subdirectory. You can read this documentation using a Web browser.

Here are some good HTML files to start with to help you learn more about using the WebSite server:

File in WebSite directory	Description
WSDOCS\INDEX.HTML	Provides an introduction to WebSite and links to useful documentation resources
WSDOCS\32DEMO\INDEX.HTML	Tests and demonstrates the capabilities of the WebSite server and your browser

Verifying the WebSite installation

After you install WebSite, you should verify that the server is installed correctly as described in the WebSite documentation. The verification has four parts that demonstrate if you can:

- Start the server properly
- View a WebSite document from the local computer
- View a WebSite document from a remote computer
- Run the server self-test and demonstration

FOR INFO For information, see the HTML files described in "WebSite documentation" on page 11.

Setting up for the Web browser and Web server

Web browser for plug-ins

To use the PowerBuilder window plug-in or the DataWindow plug-in, you need a Web browser for your platform that supports Netscape plug-ins, such as:

- Netscape Navigator Version 3.0 or higher
- Microsoft Internet Explorer Version 3.0 or higher

Web server for Web.PB

Web.PB can work with a variety of program interfaces, depending on the operating system running on the Web server where Web.PB is installed.

To use Web.PB, you need a Web server that supports the program interface in use on your platform. On Windows, you need a Web server that supports standard CGI, ISAPI, or NSAPI. On UNIX, your Web server must support standard CGI. On Macintosh, your Web server must support WSAPI.

FOR INFO For more information about Web.PB, see Chapter 2, "Web.PB Terms and Concepts" and Chapter 4, "Using Web.PB".

Setting up the plug-ins

Internet Tools plug-ins

The PowerBuilder window plug-in and the DataWindow plug-in are installed in the Internet Tools directory. On some platforms, the PowerBuilder installation program may also install a copy of the plug-ins in your Web browser's plug-ins directory if you have Netscape Navigator or Microsoft Internet Explorer installed.

If you have no browser installed or the installation program did not find the browser, you need to install the Netscape or Internet Explorer Web browser and then copy or move the plug-ins to your browser's plug-ins directory.

PowerBuilder deployment DLLs

On Windows, the PowerBuilder window plug-in must know the location of the PowerBuilder deployment DLLs. If the Setup program finds your browser, it will make the appropriate registry modifications. But if the Setup program can't find your browser, you'll have to make the modifications. You can do this in one of two ways:

- Add the directory for the PowerBuilder deployment DLLs to the application path key for the browser's executable in the Windows registry.
- Add the directory for the PowerBuilder deployment DLLs to the system path.

Copying the PB1.JPG file

The Internet Tools examples and the Web pages you create using the Web.PB Wizard reference PB1.JPG, which is a picture file.

The PB1.JPG file is installed in the Internet Tools directory. A copy of PB1.JPG must also be in the directory in which you are saving HTML files. For WebSite, \HTDOCS is the default HTML documents directory.

If PB1.JPG is not in the directory in which you are saving HTML files, you must copy it there. This ensures that the Internet Tools examples and the Web pages you create as part of the Web.PB tutorial will be able to access the PB1.JPG file.

PART 2

Web.PB

This part describes how to use the Web.PB component of the Internet Tools.

CHAPTER 2

Web.PB Terms and Concepts

About this chapter

This chapter introduces terms and concepts you need to know to begin using Web.PB.

Contents

Overview of Web.PB

What Web.PB does for you

Web.PB brings the distributed computing capabilities of PowerBuilder to the World Wide Web. By making it possible for Web browsers to invoke the services of distributed objects, Web.PB enables HTML documents to take advantage of the power and sophistication of PowerBuilder's database access technology and scripting language.

How it works

With Web.PB you can invoke the services of a PowerBuilder object with either of two HTML elements:

- Form element (<FORM>)
- Anchor element (<A>)

With these elements you can create Web applications that allow users to invoke remote object methods by clicking on hypertext links or entering information in input forms.

Supported platforms

PowerBuilder provides Web.PB for the following operating system platforms:

- Windows 95
- Windows NT
- UNIX (Solaris, HP-UX, and AIX)
- Power Macintosh

Supported program interfaces

Web.PB can work with a variety of program interfaces depending on the operating system running on the machine where Web.PB is installed:

Web.PB program interface	Supported platforms
Common Gateway Interface (CGI)	Windows UNIX
Internet Server API (ISAPI)	Windows
Netscape Server API (NSAPI)	Windows
WebSTAR API (WSAPI)	Power Macintosh

To use Web.PB, you need a Web server that supports the program interface in use on your platform. On Windows, your Web server must support standard CGI, ISAPI, or NSAPI. On UNIX, your Web server must support standard CGI. On Macintosh, your Web server must support WSAPI.

FOR INFO For more information about configuring Web servers for use with Web.PB, see "Configuring your Web server for Web.PB" on page 108.

Supported communications drivers

Web.PB can work with certain communications drivers depending on the operating system running on the Web server where Web.PB is installed:

Driver	Provides communication through	Platforms
WinSock	Sockets facility for a TCP/IP network	Windows UNIX Power Macintosh
NamedPipes	Named Pipes facility	Windows

NamedPipes driver on Windows

On Windows, WinSock driver support is linked with Web.PB automatically. Support for NamedPipes connections is not.

If you plan to use the NamedPipes communications driver, you must have the PowerBuilder deployment DLLs installed on the Web server machine.

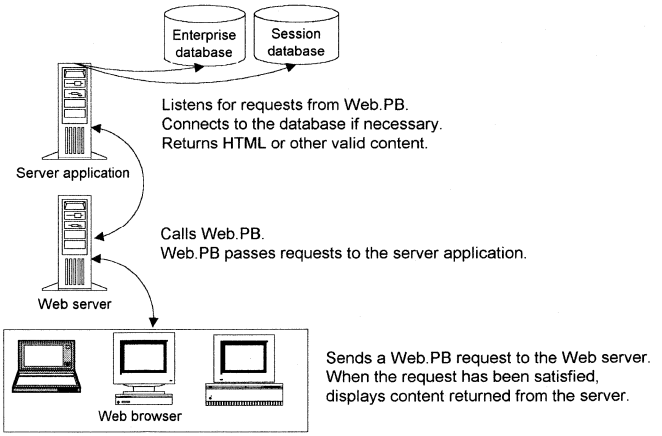
Architecture of a Web.PB application

A Web.PB application typically uses the services of three software components located on different machines:

- Web browser
- Web server
- Server application

When a Web browser makes a request that invokes a PowerBuilder object method, the Web server calls Web.PB to service the request. Web.PB passes the request to the server application, which then performs some processing and returns HTML or binary data to the browser.

If necessary, the server application connects to one or more databases to handle the request. To access corporate data, it may connect to one or more enterprise databases. On Windows, Web.PB can also connect to the session management database provided with the Web.PB class library to keep track of session and transaction information.



Web.PB program files

When a Web browser makes a request that invokes a PowerBuilder object method, the Web server actually calls one of *several* Web.PB program files to service the request. The HTML document that invokes the PowerBuilder method specifies which Web.PB program is called.

Which Web.PB program file you need to call depends on the interface you plan to use. Web.PB supplies different program interface files on different operating system platforms. The Web.PB program files *must* reside on the Web server machine.

Program interface	On Windows	On UNIX	On Power Macintosh
Standard CGI	PB6\IT\BIN\PBCGI60.EXE	pb6/pb6web/bin/pbcgi60.exe (invoked by pbcgi60.script)	—
ISAPI for the Microsoft Internet Information Server (IIS)	PB6\IT\BIN\PBISA60.DLL	—	—
NSAPI for the Netscape Commerce and Communications servers	PB6\IT\BIN\PBNS160.DLL	—	—
NSAPI for the Netscape FastTrack and Enterprise servers	PB6\IT\BIN\PBNS260.DLL	—	—
WSAPI for WebSTAR Web servers	—	—	Powersoft 6.0 Folder: Internet Tools:WebPB: WebPB

Additional requirements on UNIX

To use Web.PB on UNIX, you must have an additional component on the Web server machine where Web.PB is installed:

The pbcgi60.script file When you use Web.PB on UNIX, the HTML document that invokes the PowerBuilder method must specify pbcgi60.script instead of pbcgi60.exe (as described in "Designing the HTML pages" on page 150). The pbcgi60.script file runs the pbcgi60.exe binary image in the Web.PB /bin directory.

What you do To make this work, copy pbcgi60.script from the Web.PB /bin directory to the directory where your Web server expects to find CGI programs. Typically, this is the /cgi-bin subdirectory of your Web server's root directory (for example, /usr/ns-home/cgi-bin for Netscape Enterprise Server 2.0).

The pbcgi60.exe file should remain in the Web.PB /bin directory.

FOR INFO For instructions on configuring Web servers supported on UNIX, see "Configuring your Web server for Web.PB" on page 108.

Additional requirements on Macintosh

To use Web.PB on Power Macintosh, there are two additional requirements:

WebPB shared library The WebPB file installed in Powersoft 6.0 Folder:Internet Tools:WebPB is the WebPB shared library. This file must be copied to your Web server's plug-ins folder in order to use Web.PB on Power Macintosh.

If the PowerBuilder Installer finds a Web server on your machine, it will install a copy of WebPB to the server's plug-ins folder as part of the Web.PB installation. *In this case, no action is needed on your part.*

WebPB Preferences file The WebPB Preferences file installed in Powersoft 6.0 Folder:Internet Tools:WebPB is the Macintosh version of the Web.PB initialization file. This file must be copied to System Folder:Preferences.

The PowerBuilder Installer copies WebPB Preferences to System Folder:Preferences as part of the Web.PB installation. *No action is needed on your part.*

Distributed computing and Web.PB

Web.PB builds on the distributed computing architecture that PowerBuilder provides. Like a distributed PowerBuilder application, a Web.PB application has a client component and a server component. But in a Web.PB application, the client component is *not* a traditional PowerBuilder application. The Web.PB program file takes the place of the PowerBuilder client application, acting as a *client interface* to the PowerBuilder server application. And unlike the PowerBuilder client application, Web.PB does not use Proxy objects to address remote user objects.

Web.PB initialization file

To establish a connection to the distributed PowerBuilder server application, Web.PB uses connection properties that are maintained in the Web.PB initialization file. The name and location of the Web.PB initialization file depends on the operating system platform you are using:

Platform	Filename	Location
Windows	PBWEB.INI	Windows directory on Web server machine
UNIX	.pbweb.ini (hidden file)	Web.PB installation directory on Web server machine
Power Macintosh	WebPB Preferences	System Folder:Preferences

The parameters in the Web.PB initialization file correspond to properties of the PowerBuilder Connection object. In a conventional distributed application, the client sets its connection parameters by assigning values to properties of the Connection object. In a Web.PB application, you need to set the connection parameters in the Web.PB initialization file instead.

FOR INFO For instructions on setting connection parameters for Web.PB, see "Editing the Web.PB initialization file" on page 143.

CHAPTER 3

Web.PB Tutorial

About this chapter

This chapter is a tutorial in which you build some simple Web applications using Web.PB in PowerBuilder on the 32-bit Windows platform.

How long will it take?
About 2 hours.

Contents

About the tutorial

Two lessons

The Web.PB tutorial has two lessons:

- **Lesson 1: creating a Web.PB application**

  In this lesson you'll create a simple Web.PB application that calls some PowerBuilder functions that retrieve data and display the output in HTML pages.

- **Lesson 2: creating an interactive Web.PB application**

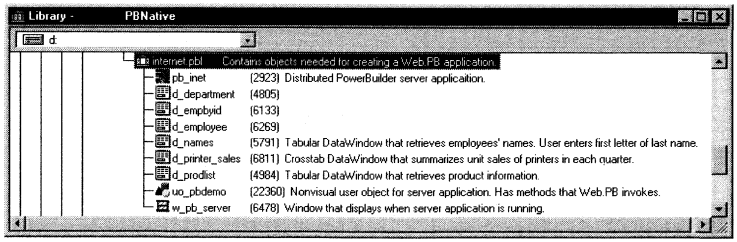
  In this lesson you'll use the Web.PB class library to build an interactive Web application. The Web.PB class library has several objects you can use to generate HTML syntax and maintain session and transaction information.

Objects you will use

The server application In the tutorial you'll use a PowerBuilder server application that is provided for you in the INTERNET.PBL file in the TUTORIAL subdirectory of the your Internet Tools (IT) installation directory. The server application has the following components:

Server application component	Description
pb_inet	The server's Application object
w_pb_server	The window that displays when the server is running. The Transport object is created in the Open event of this window
uo_pbdemo	A custom class user object that has several functions that you will use in the tutorial

The INTERNET.PBL file If you open the INTERNET.PBL in the Library painter, you can see the objects it contains. Each object has a comment associated with it to tell you what it is.



Software you need to have installed

To do the tutorial, you must have the following software installed on your computer:

Software	Details
PowerBuilder on Windows NT or Windows 95	This must include SQL Anywhere and the Powersoft Demo Database
Internet Tools	To do the tutorial, you must use the PowerBuilder Setup program to install Web.PB, which includes the Web.PB Wizard, and the Web.PB (Internet) class library FOR INFO For information about using the PowerBuilder Setup program, see the *PowerBuilder Installation Guide* FOR INFO For setup tasks you may need to complete *after* installing the Internet Tools, see "Setting up for the Internet Tools" on page 11
WebSite	To complete the tutorial, you must use WebSite as your Web server software. You can use the PowerBuilder Setup program to install WebSite FOR INFO For information about installing WebSite, see the *PowerBuilder Installation Guide*
Web browser	To view the HTML pages you create in the tutorial with the Web.PB Wizard, you need a Web browser that supports standard CGI The tutorial uses Netscape Navigator as the Web browser
TCP/IP	WebSite requires that TCP/IP be properly installed and running

How you will work

Location of components A Web.PB application usually uses the services of three software components that are typically located on three machines:

- Server application on an application server machine
- Web server software on a Web server machine
- Web browser software on a client machine

If necessary, the server application connects to one or more databases to handle requests.

In the tutorial you'll be working on only one machine. Your computer will act as the client machine, the Web server machine, the application server machine, and the database server machine.

Standard CGI The application you create in the tutorial uses standard Common Gateway Interface (CGI) technology to retrieve data. WebSite, the Web server software that ships with PowerBuilder on Windows NT and Windows 95, supports standard CGI.

Lesson 1: creating a Web.PB application

What you will do

In this lesson you will create a simple Web.PB application that calls some PowerBuilder functions that retrieve data and display the output in HTML pages.

Functions provided for the tutorial

The uo_pbdemo user object in the INTERNET.PBL file has three functions you'll be using to retrieve and display data. Each function references one DataWindow that you'll also find in the INTERNET.PBL file. In this lesson you will use these functions to retrieve and display the following:

- Product information in Tabular style
- The quarterly unit sales of printers in Crosstab style
- Employee information in Tabular style for employees with a last name beginning with a letter entered by the user

If you were working from scratch

If you were creating an application from scratch, you would need to create your own DataWindow objects and write scripts to perform the processing you want. For the tutorial, you do not need to this because the INTERNET.PBL file already has what you need. In the tutorial, you'll look at the DataWindow objects and the functions provided to familiarize yourself with the programming techniques used in the sample application.

Look at and modify WebSite server properties

Where you are

> Look at and modify WebSite server properties
Look at the PowerBuilder server application
Update the library search path
Look at the DataWindow objects
Look at the server application's nonvisual user object
Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
Run the Web.PB application

In this exercise you will run the WebSite Server Administration tool to look at WebSite server properties (specifically the Mapping properties) and then use the Mapping page to map a URL to a directory.

1 Run the *WebSite Server Administration* tool.

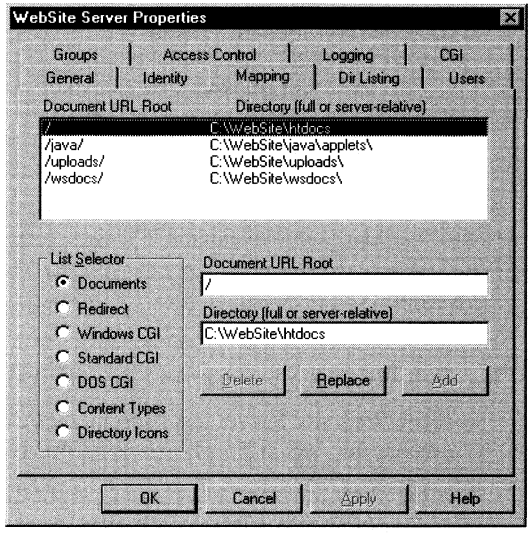
If the WebSite Server isn't running, click the Start button on the taskbar and select Programs>WebSite 1.1>Server Admin.

If the WebSite Server is running, right-click the WebSite icon on the taskbar and select WebSite Server Properties from the popup menu, or select Control>Properties from the WebSite menu bar.

The WebSite Server Properties property sheet displays.

2 Click the *Mapping* tab.

The Mapping property page displays. By default, the List Selector groupbox has the Documents radio button selected. This mapping list applies to HTML documents.



The Mapping page is a behind-the-scenes direction finder for the WebSite Server. One of the mapping types that the WebSite Server supports is URL Mapping, which includes document mapping. You use this page to map a logical URL path to a physical location (a directory) on your system.

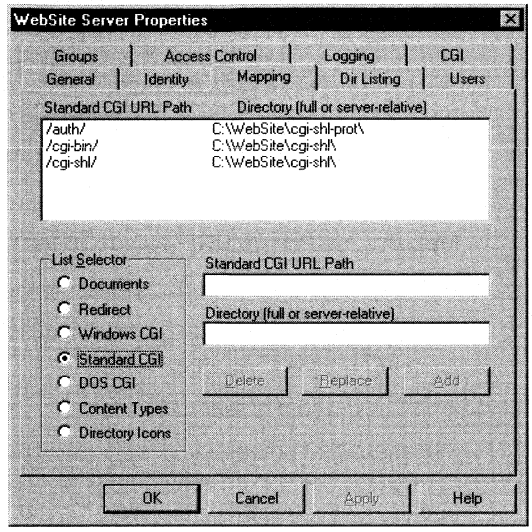
3 Notice the first default map specified by:

```
/     C:\WebSite\htdocs
```

This means that C:\WEBSITE\HTDOCS is the default document root of the server. The HTDOCS directory is where HTML documents go. Later in the tutorial you'll be putting HTML documents you create in that directory.

4 In the *List Selector* groupbox, click *Standard CGI*.

The mapping list for Standard CGI displays. The mappings are relative to the server root, which in this tutorial is C:\WEBSITE.



In WebSite, CGI mapping:

- Identifies a URL as a program (and a browser responds by executing a program rather than returning a document)
- Specifies the type of CGI program in the URL and where those programs are located (directories)

5 Notice the mapping for the */CGI-SHL/* URL path:

```
/cgi-shl/     C:\WebSite\cgi-shl\
```

This maps the /CGI-SHL/ URL path to the C:\WEBSITE\CGI-SHL\ directory, which is where WebSite expects to find CGI programs.

CGI allows an HTML program to invoke an external program. With Web.PB, for example, you can use CGI to invoke PBCGI60.EXE. When you installed WebSite, PBCGI60.EXE was installed in the \CGI-SHL directory.

6 Notice the mapping for the */CGI-BIN/* URL path.

```
/cgi-bin/     C:\WebSite\cgi-shl\
```

The /CGI-BIN/ URL is the Netscape convention for referencing CGI programs.

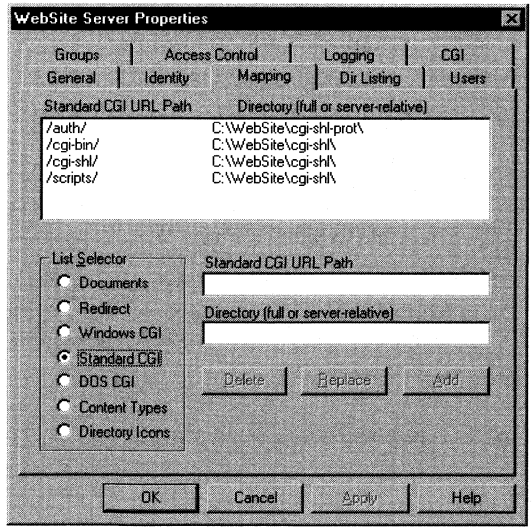
To allow WebSite to service various kinds of program requests, you can edit the CGI mappings. For example, the URL path /SCRIPTS/ is a Microsoft convention for referencing CGI programs. To handle requests that specify /SCRIPTS/ in a URL path, you need to map /SCRIPTS/ to \CGI-SHL (or whatever directory you use to manage CGI programs).

7 Map the *SCRIPTS* URL path to \CGI-SHL:

- Type the URL path /SCRIPTS/ in the Standard CGI URL Path box.
- Type the directory path C:\WEBSITE\CGI-SHL\ in the Directory (full or server-relative) box.
- Click Add and then OK.

You are asked whether you want to terminate all active connections and update the server immediately or wait until the server is idle to update the server. Since the server is idle, you can click Yes to update the server immediately.

Now the URL path /SCRIPTS/ is mapped to the directory path C:\WEBSITE\CGI-SHL\ as shown in the list of maps. You'll use the /SCRIPTS/ URL later in the tutorial.

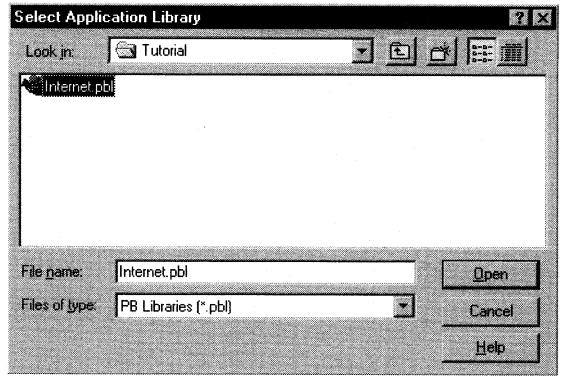


Look at the PowerBuilder server application

Where you are
Look at and modify WebSite server properties
> Look at the PowerBuilder server application
Update the library search path
Look at the DataWindow objects
Look at the server application's nonvisual user object
Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
Run the Web.PB application

Now you will look at the server application.

1 **Start the 32-bit Windows version of PowerBuilder.**
Open the Application painter.
Click the *Open* button in the PainterBar.
Find and select the *INTERNET.PBL* file in the TUTORIAL subdirectory of the Internet Tools (IT) installation directory.

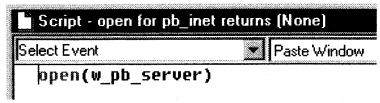


2 **Click the *Open* button.**
Select the *pb_inet* application.
Click *OK*.

The current application is now PB_INET.

3 Click the *Script* button in the PainterBar.

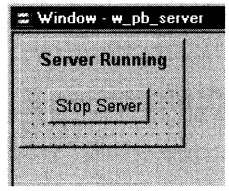
The script for the application object's Open event displays.



The w_pb_server window is the server window shown in the next step. This line of script opens the server window whenever the server application opens.

You will see shortly that whenever the server is running, it is listening for requests and is ready to provide services.

4 Close the *PowerScript* painter.
Open the *Window* painter.
Open the *w_pb_server* window.



Now you will look at the script for the Server window's Open event.

5 Select *Script* from the window's popup menu.
If you don't see the script for the Open event, select *open* from the Select Event listbox.

The script for the Server window's Open event displays.

```
string ErrText, ErrNo

// set connection information

mytransport = CREATE transport

mytransport.driver = "winsock"
mytransport.location = "localhost"
mytransport.application = "pbserver"

// start listening for client connections
mytransport.Listen()
```

```
// Check for errors
if (mytransport.ErrCode <> 0) then
   ErrText = mytransport.ErrText
   ErrNo = string(mytransport.ErrCode)
   MessageBox( "Application Open", &
      "Could not start 'Listener'.~r~n" &
      + ErrNo + ": " + ErrText )
   Close(this)
   return
end if
```

This script performs some common procedures that any distributed PowerBuilder server application would perform. It creates a Transport object that uses the WinSock driver and executes the Listen function to service requests. Once these simple steps have been performed, the server application can handle any requests that access the nonvisual user objects provided in the PBL.

6 Select *close* in the *Select Event* dropdown listbox.

The script for the Server window's Close event displays.

This script tells the server to stop listening for requests and destroys the Transport object that was created in the Open event of the Server window.

7 Close the PowerScript painter and the Window painter.

Update the library search path

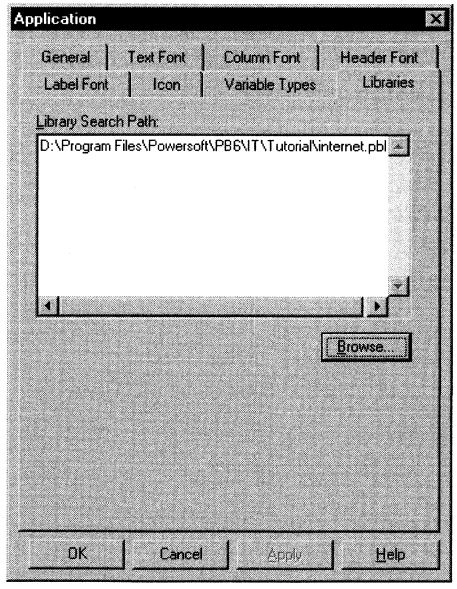
Where you are
Look at and modify WebSite server properties
Look at the PowerBuilder server application
> Update the library search path
Look at the DataWindow objects
Look at the server application's nonvisual user object
Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
Run the Web.PB application

In this exercise you will update the library search path to include the Web.PB class library. You will use the class library in Lesson 2 of the tutorial.

1 Open the Application painter if it is not open already.

2 Select *Properties* from the PainterBar and then select the *Libraries* tab.

The Application property sheet displays.



3 **Click Browse.**

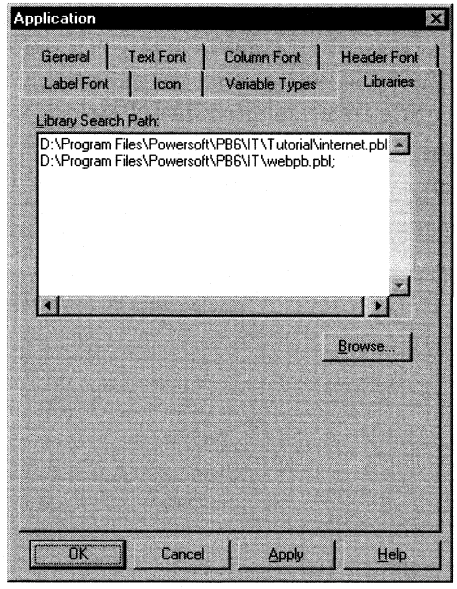
The Select Library dialog box displays.

4 **Navigate to the Internet Tools (IT) installation directory.**

5 **Select the *WEBPB.PBL* file.**

6 **Click *Open*.**

The Application property sheet redisplays with the Web.PB class library added to the list.



7 **Click *OK*.**

8 **Save the Application object.**

Look at the DataWindow objects

Where you are
Look at and modify WebSite server properties
Look at the PowerBuilder server application
Update the library search path
> Look at the DataWindow objects
Look at the server application's nonvisual user object
Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
Run the Web.PB application

DataWindow objects provided for the tutorial The INTERNET.PBL contains three DataWindow objects that you'll work with in Lesson 1. (There are also several other DataWindow objects that you'll use in Lesson 2 when you create an interactive Web.PB application.)

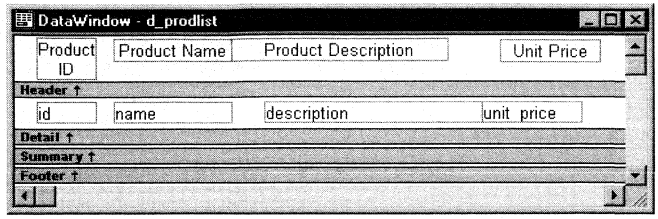
This DataWindow object	Does this
d_prodlist	Retrieves in Tabular style information about products
d_printer_sales	Retrieves in Crosstab style the unit sales of printers for each quarter
d_names	Retrieves in Tabular style the names of employees that have a last name that begins with a letter (entered by the user as a retrieval argument) and counts the number of names Uses the SQL LIKE operator in the Where criteria of the DataWindow definition

Later in the tutorial you'll see each of the DataWindow objects providing data to your Web.PB application.

What you will do Now you will look at each of the DataWindow objects you'll be using.

Powersoft Demo Database
PowerBuilder *must* be connected to the Powersoft Demo Database for data to display in the DataWindow objects.

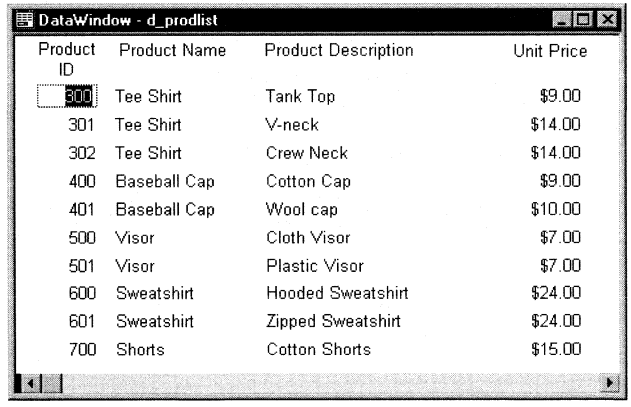
1 **Open the DataWindow painter.**
Select the *d_prodlist* DataWindow.
Click *OK*.



The d_prodlist DataWindow is a tabular DataWindow that uses the ID, Name, Description, and Unit_price columns in the Product table in the Powersoft Demo Database.

2 **Click the *Preview* button in the PainterBar.**

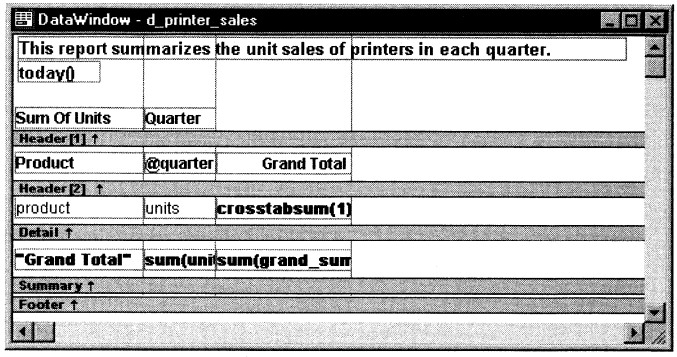
The DataWindow retrieves product data.



Product ID	Product Name	Product Description	Unit Price
300	Tee Shirt	Tank Top	$9.00
301	Tee Shirt	V-neck	$14.00
302	Tee Shirt	Crew Neck	$14.00
400	Baseball Cap	Cotton Cap	$9.00
401	Baseball Cap	Wool cap	$10.00
500	Visor	Cloth Visor	$7.00
501	Visor	Plastic Visor	$7.00
600	Sweatshirt	Hooded Sweatshirt	$24.00
601	Sweatshirt	Zipped Sweatshirt	$24.00
700	Shorts	Cotton Shorts	$15.00

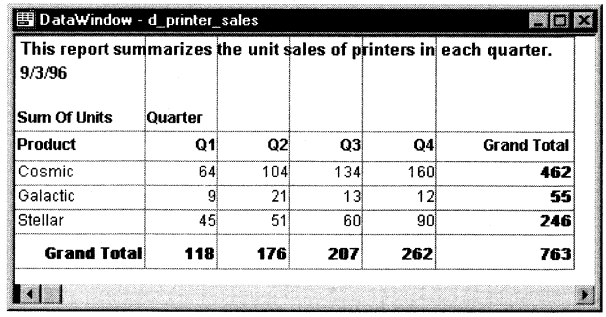
3 **Click the *Preview* button in the PainterBar.**
Click the *Open* button in the PainterBar.
Select the *d_printer_sales* DataWindow.
Click *OK*.

The d_printer_sales DataWindow displays in the DataWindow painter workspace. This DataWindow is a crosstab that uses the Quarter, Product, and Units columns from the Printer table in the Powersoft Demo Database.



4 **Click the *Preview* button in the PainterBar.**

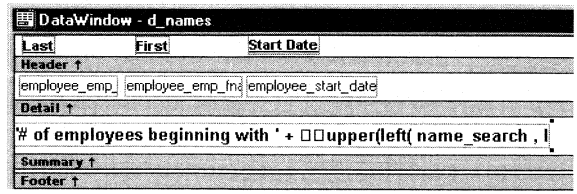
The DataWindow retrieves the data for printer sales.



Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	**462**
Galactic	9	21	13	12	**55**
Stellar	45	51	60	90	**246**
Grand Total	**118**	**176**	**207**	**262**	**763**

5 **Click the *Preview* button in the PainterBar.**
Click the *Open* button in the PainterBar.
Select the *d_names* DataWindow.
Click *OK*.

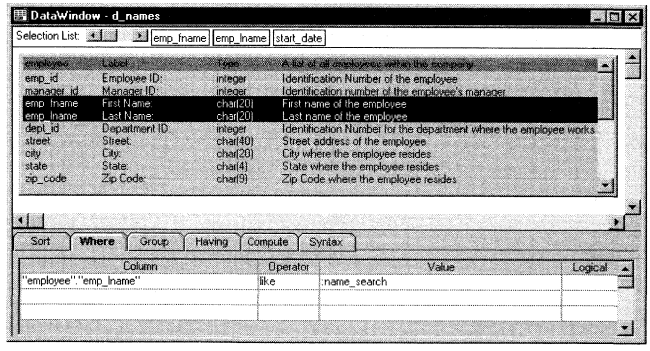
The d_names DataWindow displays in the DataWindow painter workspace.



In the Detail band you can see three columns (employee_emp_lname, employee_emp_fname, and employee_start_date) from the Employee table in the Powersoft Demo Database. In the Summary band you can see a computed field that counts the number of employees for whom data is retrieved.

6 **Select *Design>Data Source* from the menu bar.**
or
Click the *SQL Data Source* button in the PainterBar.

The Select painter workspace displays the definition of the DataWindow object.



You can see this expression for the Where criteria:

```
"employee"."emp_lname" like :name_search
```

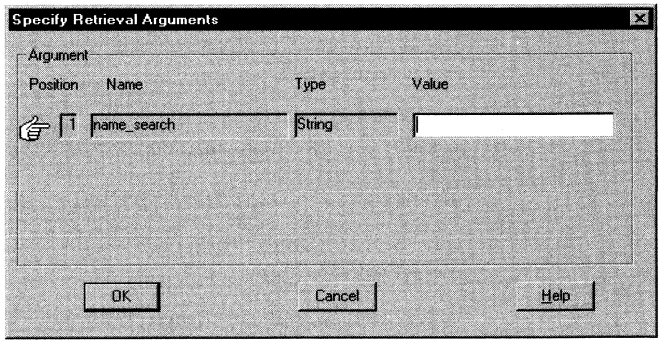
The LIKE operator in the expression allows for pattern matching. The name_search retrieval argument allows the user to specify one or more search letters for employees' last names. You'll also notice the name_search retrieval argument in the computed field definition in the DataWindow painter workspace.

7 Click the *SQL Select* button in the PainterBar.

You return to the DataWindow painter workspace.

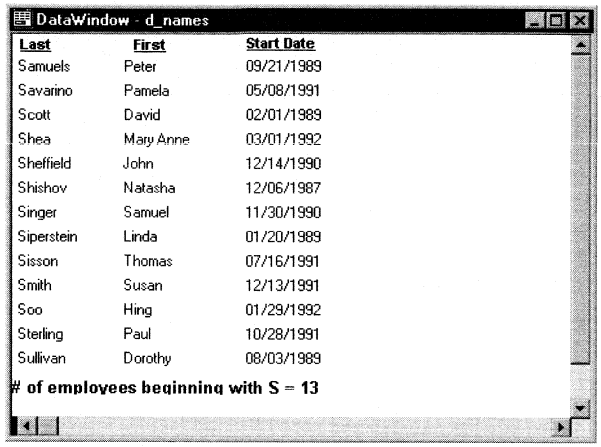
8 Click the *Preview* button in the PainterBar.

The Specify Retrieval Arguments dialog box displays and prompts you to supply a letter.



9 Enter *s%* in the *Value* box and click *OK*.

The DataWindow retrieves data for employees whose last name begins with S and displays the number of employee names retrieved.



10 Close the DataWindow painter.

Look at the server application's nonvisual user object

Where you are
Look at and modify WebSite server properties
Look at the PowerBuilder server application
Update the library search path
Look at the DataWindow objects
> Look at the server application's nonvisual user object
Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
Run the Web.PB application

User object provided for the tutorial The basic building blocks of a server application are the remote objects it contains. Each remote object is a custom class (nonvisual) user object that has methods that can be invoked by a Web.PB application.

When you create a server application from scratch, you need to write the methods for the nonvisual user objects. For this tutorial one nonvisual user object has been created for you, and its functions have been coded for you.

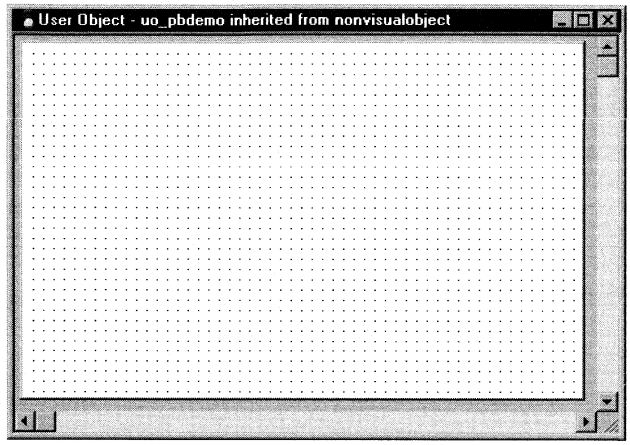
Each user object function constructs and returns HTML that formats the contents of a DataWindow object. Each function uses a DataStore to reference one of the DataWindow objects you looked at in the preceding exercise:

This user object function	References this DataWindow object
f_retrieve_list	d_prodlist
f_crosstab	d_printer_sales
f_name_search	d_names

What you will do Now you will look at these user object functions.

1 **Open the User Object painter in PowerBuilder.**
Select the *uo_pbdemo* user object.
Click *OK*.

The nonvisual user object uo_pbdemo displays in the workspace.



2 **Select *User Object Functions* from the user object's popup menu.**
Select the *f_retrieve_list* function.
Click *OK*.

The script for the f_retrieve_list function displays in the PowerScript painter.

```
//Retrieve a list of products and return the
//data in HTML format

string ls_html
string ls_search_arg
long ll_rows_retrieved

ds_names = CREATE datastore

ds_names.dataobject="d_prodlist"

ds_names.settransobject(gtr_trans)

ll_rows_retrieved= ds_names.retrieve()

IF ll_rows_retrieved > 0 THEN
  ls_html=ds_names.object.datawindow.data.htmltable
```

```
ELSE
   ls_html = 'No products found for this search'
END IF

RETURN ls_html
```

3 Look at the *f_retrieve_list* script.

This script creates an HTML string that displays a list of products that is retrieved from a database. The function uses a DataStore to access the product data. After creating an HTML header, f_retrieve_list constructs the HTML that displays the data by assigning the Data.HTMLTable property to a string variable.

4 Click the *Open* button in the PainterBar.
Select the *f_crosstab* function.
Click *OK*.

The script for the f_crosstab function displays in the PowerScript painter.

```
string ls_html
long ll_rows_retrieved

datastore ds_xtab
ds_xtab = CREATE datastore

ds_xtab.dataobject="d_printer_sales"

ds_xtab.settransobject(gtr_trans)

ll_rows_retrieved= ds_xtab.retrieve()

IF ll_rows_retrieved > 0 THEN
   ls_html = ls_html &
   + ds_xtab.object.datawindow.data.htmltable
ELSE
   ls_html = 'No data'
END IF

RETURN ls_html
```

5 Look at the *f_crosstab* script.

This script works just like the script for f_retrieve_list, except that it uses a DataWindow that has the Crosstab presentation style.

6 Click the *Open* button in the PainterBar.
Select the *f_name_search* function.
Click *OK*.

The script for the f_name_search function displays in the PowerScript painter.

```
string ls_html
string ls_search_arg
long ll_rows_retrieved

string ls_header = "<HTML><HEAD>"
string ls_title = "<TITLE>Employee List</TITLE>"
string ls_endheader = "</HEAD>"
string ls_h1 = "<BODY><H1>Employee List</H1>"
string ls_footer = "</BODY></HTML>"

ls_html = ls_header &
   + ls_title + ls_endheader + ls_h1

ls_search_arg = search_letter + '%'

ds_names = CREATE datastore

ds_names.dataobject="d_names"

ds_names.settransobject(gtr_trans)

ll_rows_retrieved = &
   ds_names.retrieve(ls_search_arg)

IF ll_rows_retrieved > 0 THEN
   ls_html = ls_html &
   + ds_names.object.datawindow.data.htmltable
ELSE
   ls_html = 'No employees found for this search'
END IF

ls_html = ls_html + ls_footer
```

```
RETURN ls_html
```

7 Look at the *f_name_search* script.

This script retrieves employee data by using a search argument to perform pattern matching. The function returns an HTML string that includes a header, title, and footer along with the retrieved data.

8 Close the PowerScript painter.
Close the User Object painter.

Use the Web.PB Wizard

Where you are
Look at and modify WebSite server properties
Look at the PowerBuilder server application
Update the library search path
Look at the DataWindow objects
Look at the server application's nonvisual user object
> Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
Run the Web.PB application

What you will do Now you will use the Web.PB Wizard to create three Web pages. Each Web page will call one of the functions in the nonvisual user object you looked at in the preceding exercise. You will also use the Wizard to update the PBWEB.INI file, which provides the Web.PB program with the information it needs to connect to the PowerBuilder server application.

About running the Web.PB Wizard In this lesson, you run the Web.PB Wizard from within PowerBuilder. When you install Web.PB, the PowerBuilder Setup program places the following line in the [PB] section of your PowerBuilder initialization file (PB.INI on Windows):

WebPBWizard=html_webpb,*pathname_of_WIZARD.PBD*

For example:

```
WebPBWizard=html_webpb,D:\Program Files\Powersoft\
   pb6\IT\wizard.pbd
```

The presence of this line in your PB.INI file causes the Web.PB button to display in the PowerBar. You click the Web.PB button to run the Web.PB Wizard.

Running the Web.PB Wizard standalone
You can run the Web.PB Wizard as a standalone executable outside PowerBuilder. When you install Web.PB, PBWIZARD.EXE is installed in the Internet Tools (IT) installation directory.

About creating the Web pages Two of the Web pages will use the HTML Anchor element <A> to display a hypertext link that a user could click to display data. One of the Web pages will use an HTML Form element <FORM> to provide an input form for entering a search pattern and retrieving data based on the pattern entered.

Here's a summary of the Web pages you'll create:

This Web page	Uses this HTML element	To call this user object function	Which creates a DataStore to access this DataWindow object
tut1.htm	Anchor element <A>	f_retrieve_list	d_prodlist
tut2.htm	Anchor element <A>	f_crosstab	d_printer_sales
tut3.htm	Form element <FORM>	f_name_search	d_names

About updating the PBWEB.INI file For the tutorial application to run, the PBWEB.INI file must have a server alias section that contains values for the server application name, location, and communications driver. The server alias provides a way for program requests to refer to the server application without having to specify connection parameters.

You are going to add the server alias section using the Web.PB Wizard. (You could also edit the PBWEB.INI file directly using any text editor.)

Here's what the section of the PBWEB.INI file will look like after you add an entry to allow the tutorial application to run.

```
[TUTORIAL]
application=pbserver
location=localhost
driver=winsock
```

The three values must match the property values for the Transport object, which were set in the script for the Open event of the server application window (see step 5 in "Look at the PowerBuilder server application" on page 33).

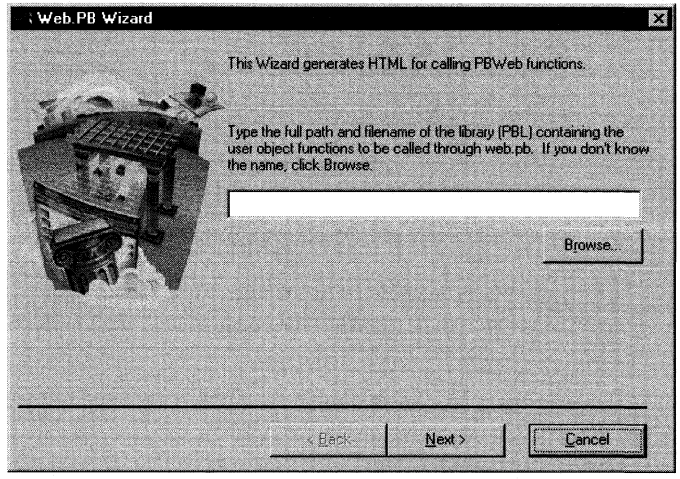
```
mytransport.driver = "winsock"
mytransport.location = "localhost"
mytransport.application = "pbserver"
```

Create the Web page TUT1.HTM

Now you will create a Web page to call the f_retrieve_list function that retrieves data from the tabular DataWindow d_prodlist and returns the data in HTML.

1 **Click the *Web.PB* button in the PowerBar to run the Web.PB Wizard.**
or
Click the Web.PB button on the PowerPanel dropdown toolbar (in the PowerBar).

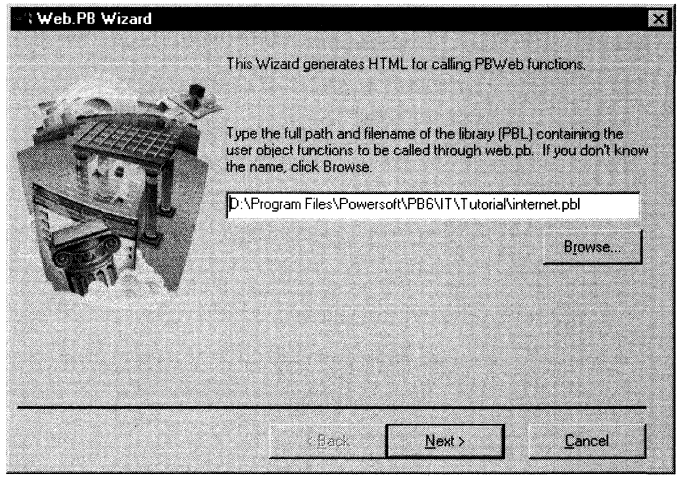
The Wizard displays a window that allows you to specify the PBL file for the server application.



If you've already used the Wizard, the Wizard displays the last PBL you accessed.

2 **Enter the full path and filename of the INTERNET.PBL file.**
or
Use the Browse button to locate and select the INTERNET.PBL file.

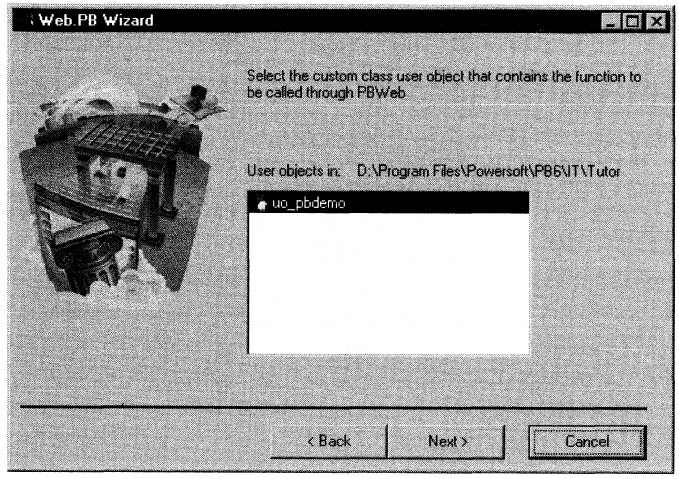
In this tutorial the path is D:\PROGRAM FILES\POWERSOFT\PB6\IT\TUTORIAL\INTERNET.PBL.



The INTERNET.PBL file contains the user object functions you want your Web page to use.

3 Click *Next.*

The Wizard displays the nonvisual user objects in the PBL that return strings or blobs. In the INTERNET.PBL file there's only one nonvisual user object, so the Wizard displays uo_pbdemo.



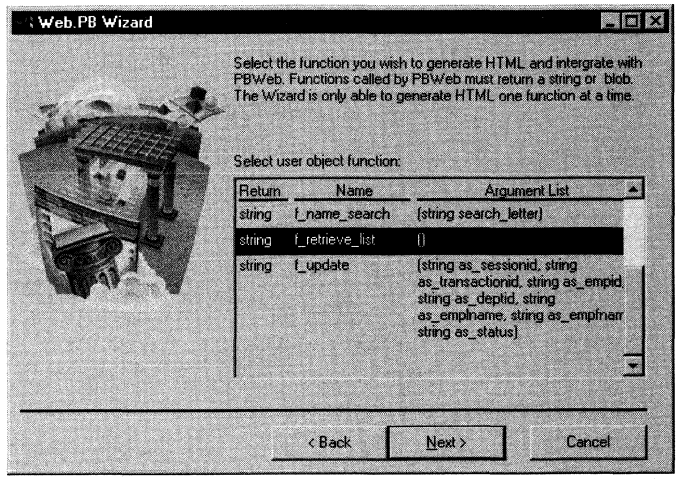
4 Click *Next.*

The Wizard displays all functions in the user object.

Selecting a user object with an ancestor not in the PBL
If you selected a user object with an ancestor not in the PBL, the Wizard now prompts you to name the ancestor PBL and then checks to see if the ancestor is nonvisual. If the ancestor isn't nonvisual, you must select another user object.

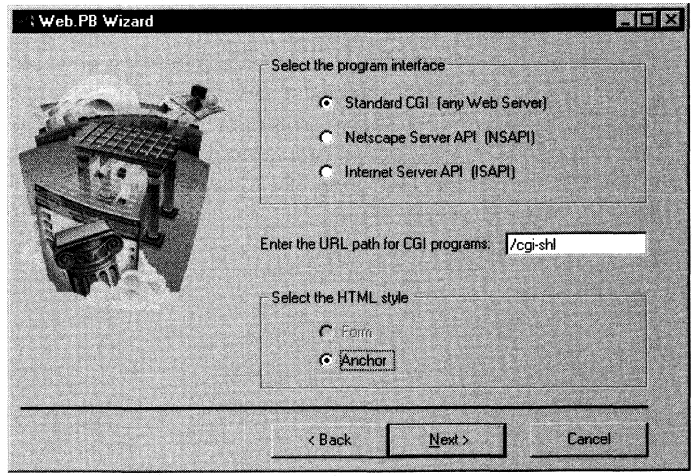
---

5 **Scroll through the user object function list and select the *f_retrieve_list* function.**



6 **Click *Next*.**

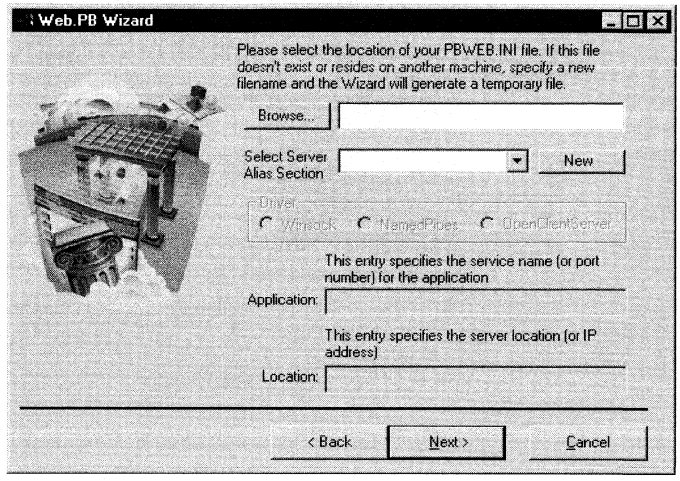
Now the Wizard needs information about your Web server, the server's location for CGI programs, and the HTML style (Form or Anchor) you want.



WebSite's convention for referencing CGI programs is /CGI-SHL/, so this URL displays automatically. Notice that the HTML style Anchor is selected; the Wizard won't allow you to use the Form style for a function that has no arguments.

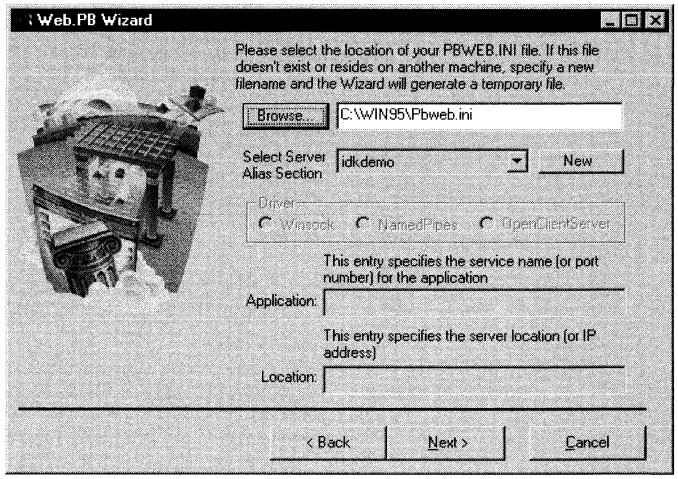
7 Accept the defaults, including *Standard CGI (any Web Server).* **Click** *Next.*

Now the Wizard needs the location of the PBWEB.INI file and the Server alias section name in the PBWEB.INI file. Since you haven't edited the PBWEB.INI file to add a section name, you'll use the Wizard to do this automatically.



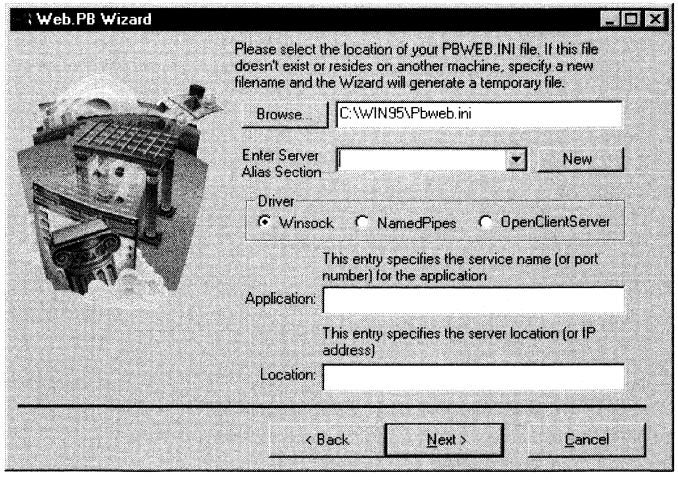
8 Enter the location of the PBWEB.INI file.
or
Click the *Browse* button and locate the PBWEB.INI file.

The path for the PBWEB.INI file is C:*Windows directory*\PBWEB.INI (for example, C:\WIN95\PBWEB.INI.)

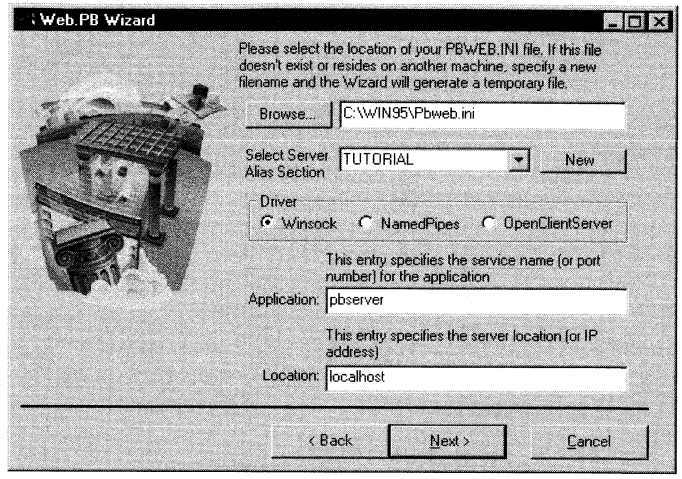


9 Click the *New* button.

The Enter Server Alias, Driver, Application, and Location boxes are enabled. Now you'll give the Wizard the information it needs to create the server alias section in the PBWEB.INI file.



10 Enter *TUTORIAL* **in the** *Select Server Alias Section* **box.**
Select *WinSock* **in the** *Driver* **box.**
Enter *pbserver* **in the** *Application* **box.**
Enter *localhost* **in the** *Location* **box.**

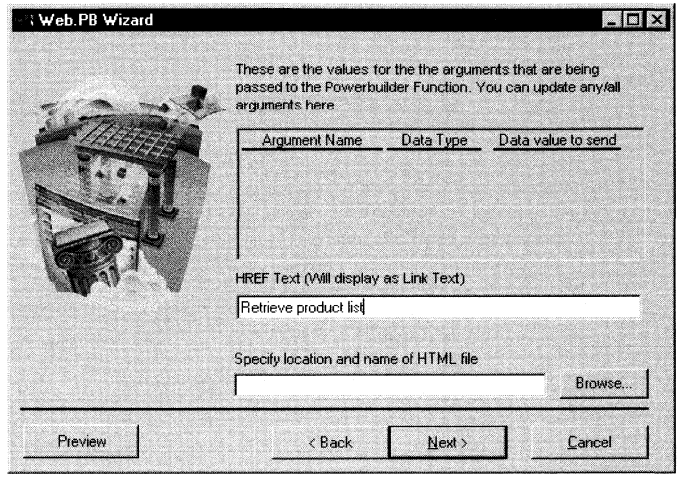


The three values for Driver, Application, and Location must match the property values for the Transport object, which were set in the script for the Open event for the server application window.

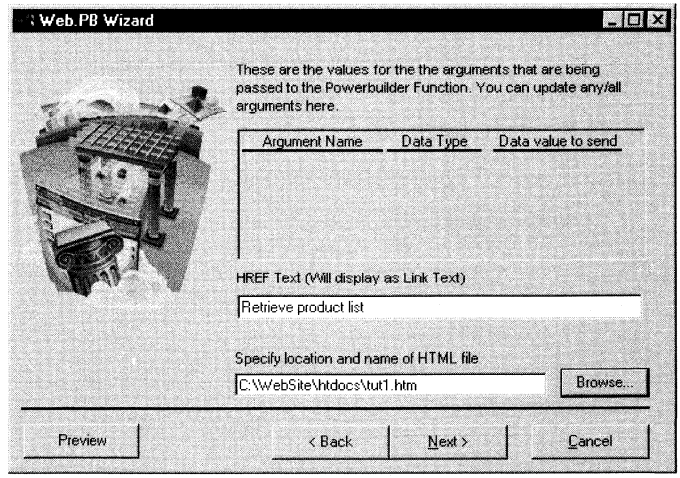
11 Click *Next.*

The function you selected has no arguments, so none display.

12 Enter *Retrieve product list* in the *HREF Text* box.



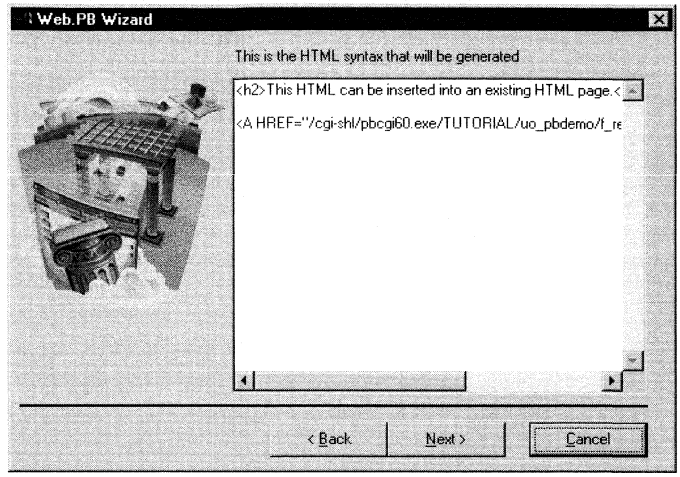
13 Enter *C:\WEBSITE\HTDOCS\TUT1.HTM* for the location and name of the HTML file you are creating.



You're using WebSite as your Web server, and \HTDOCS is where WebSite looks for HTML documents.

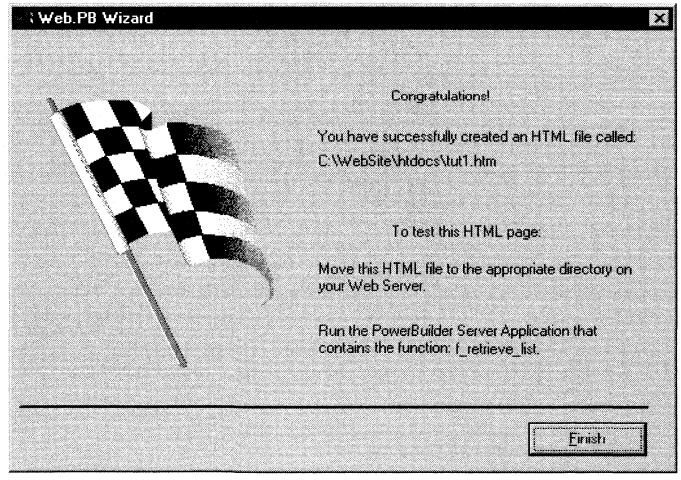
14 (Optional) Click *Preview.*

The Wizard displays the HTML syntax it will generate.



15 Click *Next.*

The Wizard tells you that you've completed the process of creating the first Web page.



16 Click *Finish.*

Create the Web page TUT2.HTM

Now you will create a Web page to call the f_crosstab function that retrieves the data for quarterly printer sales from the crosstab DataWindow d_printer_sales and returns the data in HTML.

The steps are similar to the ones you used for creating TUT1.HTM. But the server alias section in the PBWEB.INI file already exists, so you will not have to create it.

1 Run the Wizard.

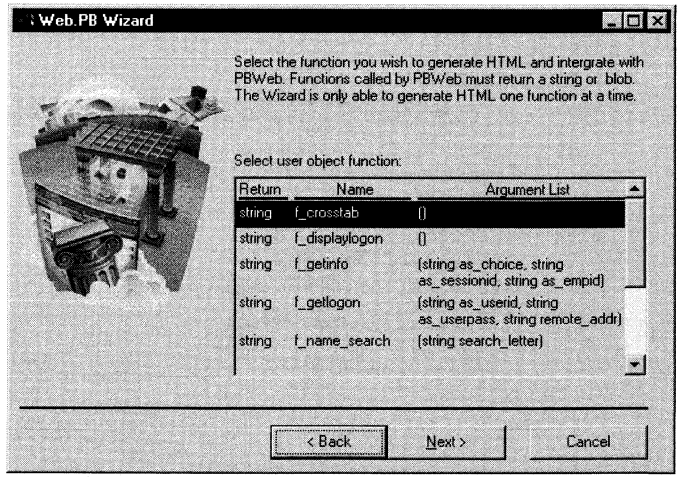
2 Click *Next*.

The Wizard displays uo_pbdemo.

3 Click *Next*.

The Wizard displays the functions in uo_pbdemo.

4 Select the *f_crosstab* function.



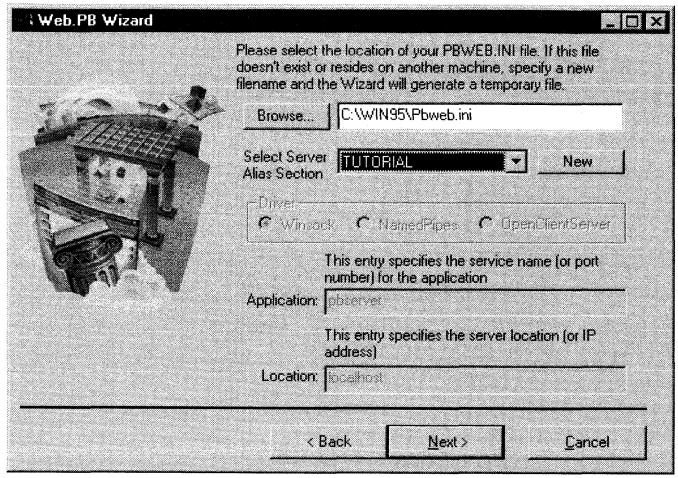
5 Click *Next*.

Accept the default values for the program interface (Standard CGI), the URL path for CGI programs (/cgi-shl), and the HTML style (Anchor).

6 **Click** *Next.*

Accept the default location of the PBWEB.INI file.

7 **Select** *TUTORIAL* **in the** *Select Server Alias Section* **dropdown listbox.**

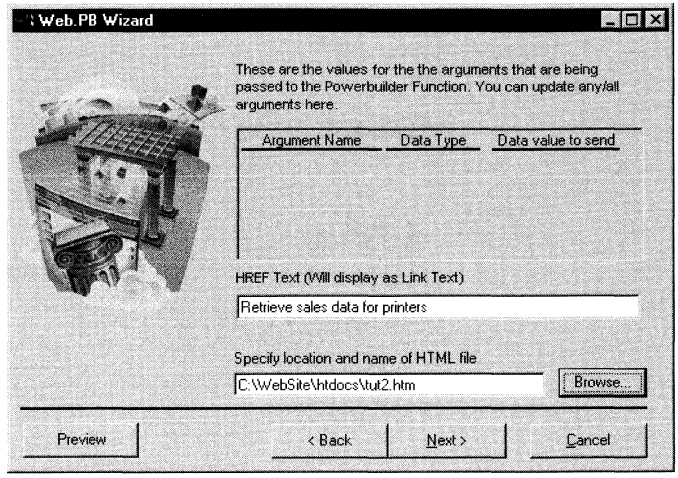


8 **Click** *Next.*

The function you selected has no arguments, so none display.

9 **Enter** *Retrieve sales data for printers* **in the** *HREF Text* **box.**

10 Enter *C:\WEBSITE\HTDOCS\TUT2.HTM* for the location and name of the HTML file you are creating.



11 Click *Next*.
Click *Finish*.

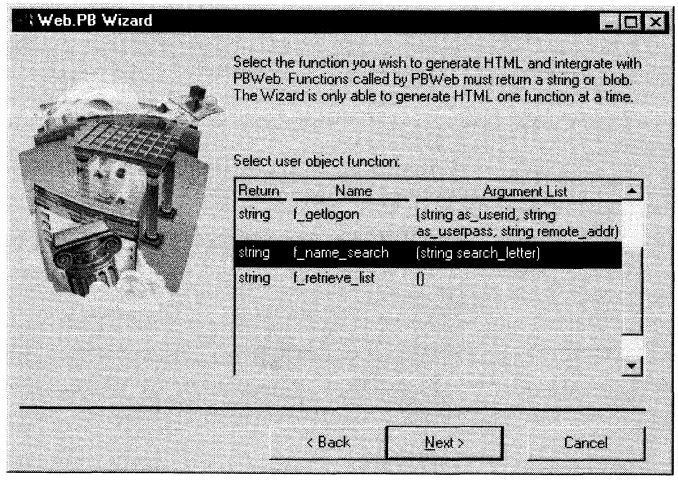
Create the Web page TUT3.HTM

Now you will create a Web page to call the f_name_search function that retrieves data from the tabular DataWindow d_names and returns the data in HTML. The f_name_search function has a retrieval argument.

The steps are the similar to ones you used for creating TUT2.HTM, but you'll use the Wizard to create a Web page with a Form element (rather than an Anchor element). When you run the application, you can use the form to enter a search pattern, such as the first letter or two of the last name.

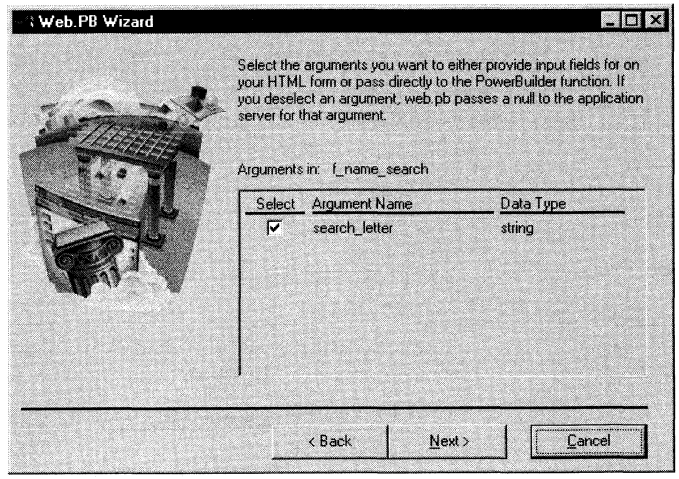
1 **Run the Wizard.**
Click *Next.*
Click *Next* **again.**

2 **Scroll through the user object function list and select the** *f_name_search* **function.**

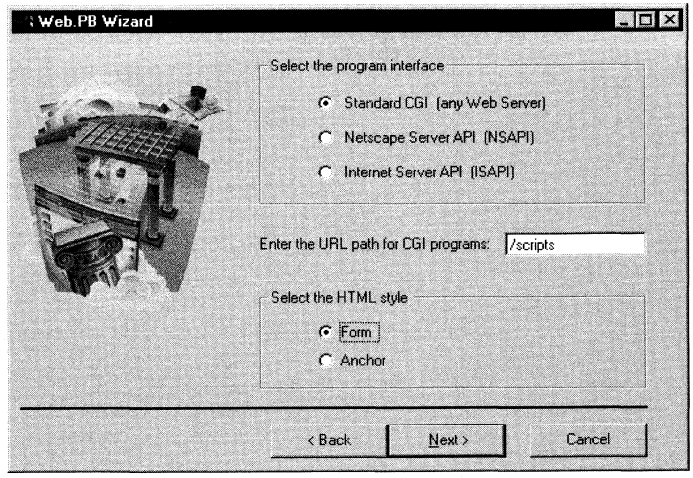


3 **Click** *Next.*

The function you selected has one argument that the Wizard displays and selects.



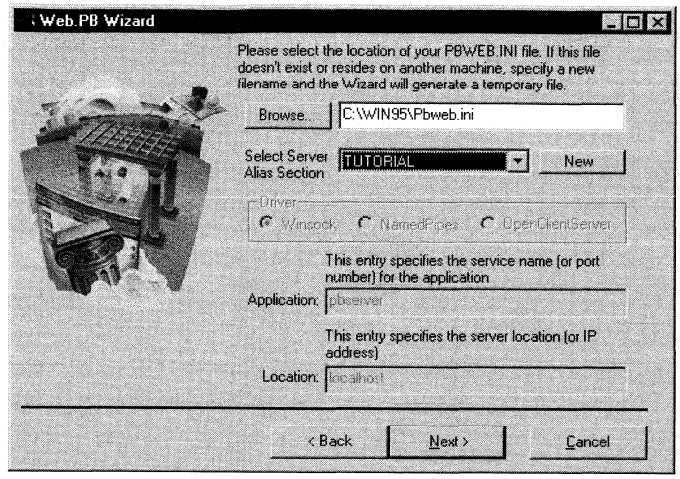
4 **Click** *Next.*
Enter */scripts* **in the** *Enter the URL path for CGI programs* **box.**
Accept the other default values.



5 **Click** *Next.*

Accept the default location of the PBWEB.INI file.

6 **Select** *TUTORIAL* **in the** *Select Server Alias Section* **dropdown listbox.**



7 **Click** *Next.*

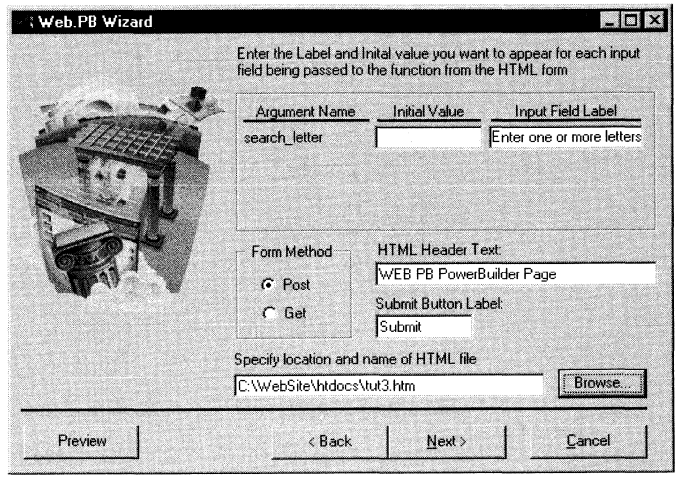
The Wizard prompts you for an initial value and a label for the input argument that will be passed to the function from the form.

An initial value isn't needed, but you'll modify the label.

8 **Enter** *Enter one or more letters:* **in the** *Input Field Label* **box.**

You could also modify the HTML Header Text and the Submit Button Label, but instead you'll accept the defaults.

9 **Enter** *C:\WEBSITE\HTDOCS\TUT3.HTM* **for the location and name of the HTML file you are creating.**



10 **Click** *Next.*
Click *Finish.*

Edit the HOSTS and SERVICES files

Where you are
Look at and modify WebSite server properties
Look at the PowerBuilder server application
Update the library search path
Look at the DataWindow objects
Look at the server application's nonvisual user object
Use the Web.PB Wizard
> Edit the HOSTS and SERVICES files
Run the Web.PB application

Before you can run the application, you must edit the TCP/IP HOSTS and SERVICES files. You can use any text editor.

The HOSTS file associates IP addresses with host names. The SERVICES file associates port numbers with application names. The HOSTS and SERVICES files are in your main Windows directory.

1 **Open Notepad or any text editor.**

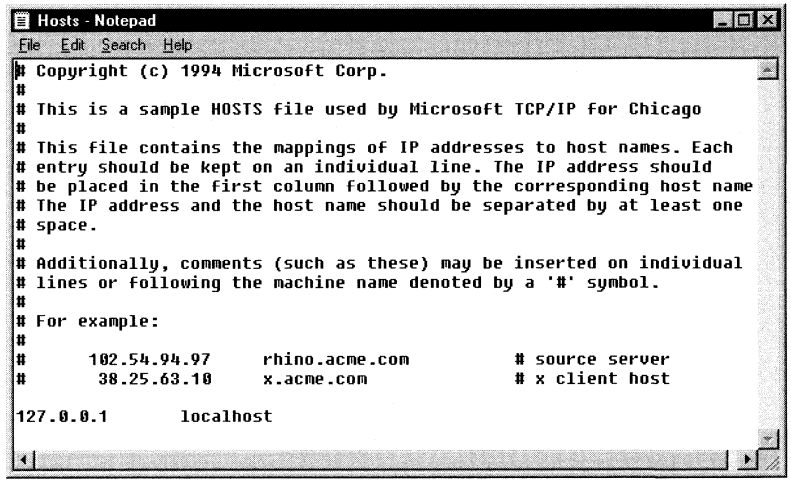
2 **Open the *HOSTS* file and display the last line of the file.**

3 **Add the IP address *127.0.0.1* and the host entry *LOCALHOST* to the end of the file if it is not there already.**

The entry will look like this:

```
127.0.0.1     localhost
```

Here's the entry in the HOSTS file.



The host address 127.0.0.1 is a special address used to designate the address of the local host. This addressing allows the host to address itself in the same way it addresses a remote host.

4 Save the HOSTS file.

5 Open the SERVICES file and display the last line of the file.

6 Add an entry to the end of the file.

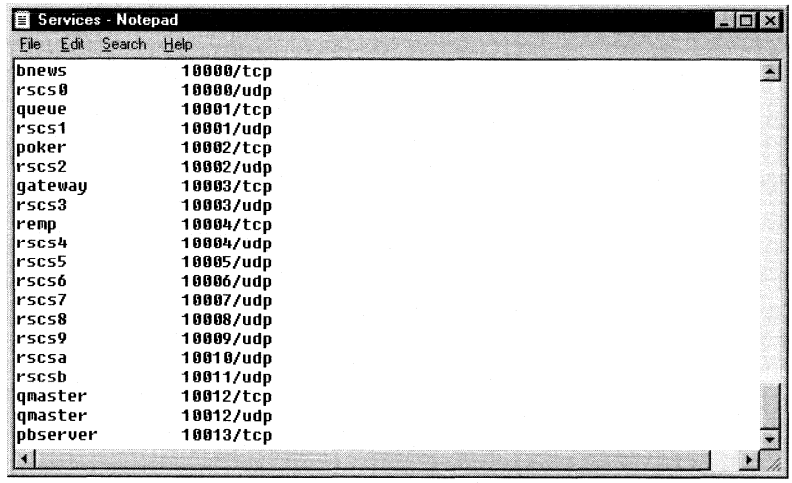
The entry will look like this, but your port number may be different:

```
pbserver     10013/tcp
```

7 **Press** ENTER.

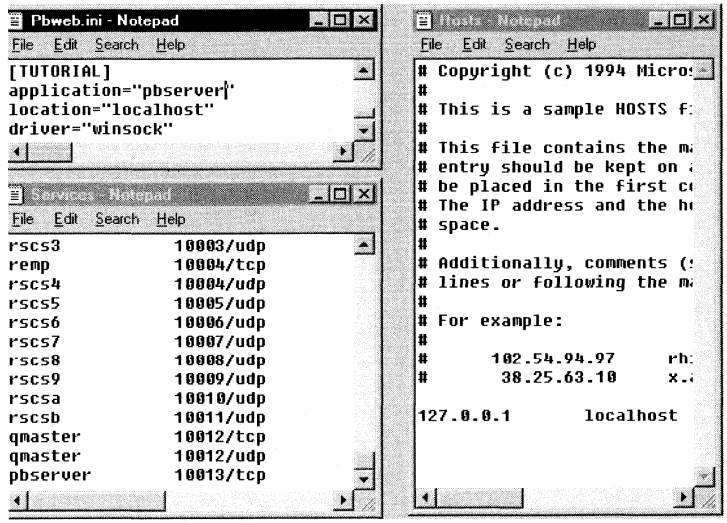
The name pbserver is the server application name that's in the PBWEB.INI file. In your SERVICES file, use a number that is higher than the port number of the last entry. In this tutorial the number selected is 10013 (because it follows 10012, which is the port number of the last entry added).

Here's the entry in the SERVICES file.



8 **Save the SERVICES file.**

9 Now open the PBWEB.INI, SERVICES, and HOSTS files in separate windows.
Arrange them on your screen so you can see the changes you've made in all three files.



The Web.PB program looks at the server alias TUTORIAL in the PBWEB.INI file for information. The server alias TUTORIAL specifies the server application as pbserver and the location as localhost.

The SERVICES file provides the port number for the pbserver application. The HOSTS file provides the IP address for localhost. If the port number and IP address were in the PBWEB.INI file, the CGI program would not look at the HOSTS and SERVICES files.

Run the Web.PB application

Where you are
Look at and modify WebSite server properties
Look at the PowerBuilder server application
Update the library search path
Look at the DataWindow objects
Look at the server application's nonvisual user object
Use the Web.PB Wizard
Edit the HOSTS and SERVICES files
> Run the Web.PB application

Now you'll run the Web.PB application. The application uses the HTML files the Wizard created to call three functions in the server application.

Call the f_retrieve_list function

1 Start WebSite.

WebSite is your Web server.

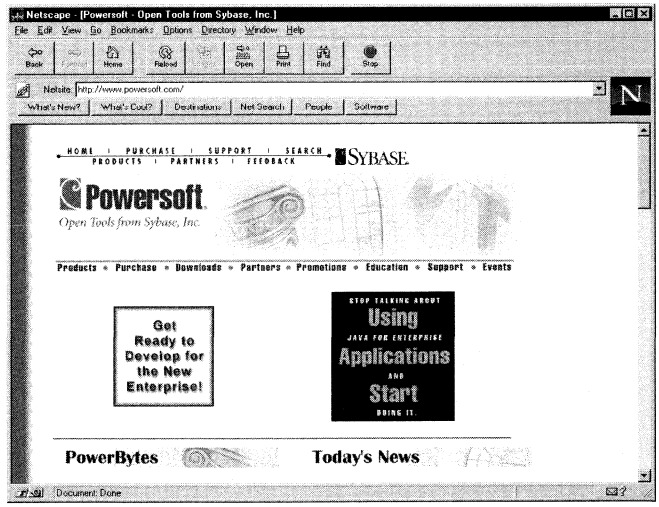
2 Run the PowerBuilder server application *pb_inet*.

Whenever the server is running, it is listening for client requests.

Another way to test the server application
You could also test the server application in a process separate from PowerBuilder by creating an executable and running it.

3 Start Netscape.

In this tutorial, Netscape is the Web browser.



4 In Netscape, select *File>Open Location* from the menu bar.

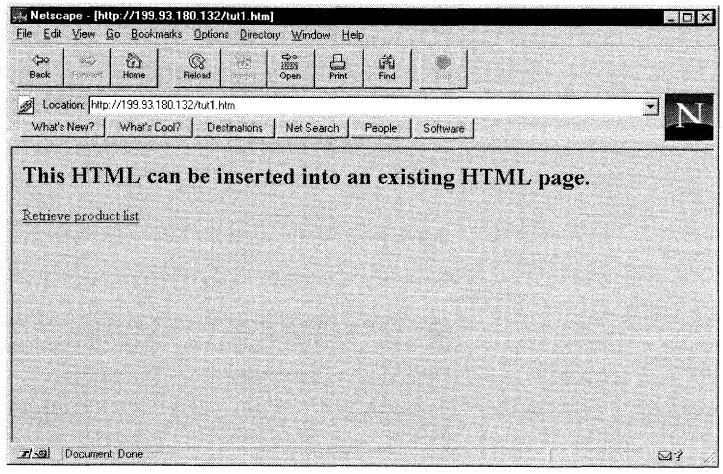
5 Enter *http://199.93.180.132/tut1.htm* in the *Open Location* box, substituting your IP address for the address shown here.

If you use a dynamic host configuration protocol (DHCP) server to generate IP addresses dynamically, your IP address may change across sessions. To find out what your current IP address is, you can execute the PING command in a DOS window.

Specifying a host name instead of an IP address

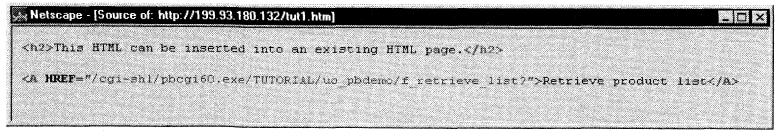
If your host name is registered with the Domain Name Service (DNS), you can specify the host name instead of your IP address in the Open Location box. Alternatively, you can specify the host name LOCALHOST.

Netscape displays the TUT1.HTM page. Remember that TUT1.HTM has an HTML Anchor element, which is a link to the list of products.



6 Select *View>Document Source* from the menu bar.

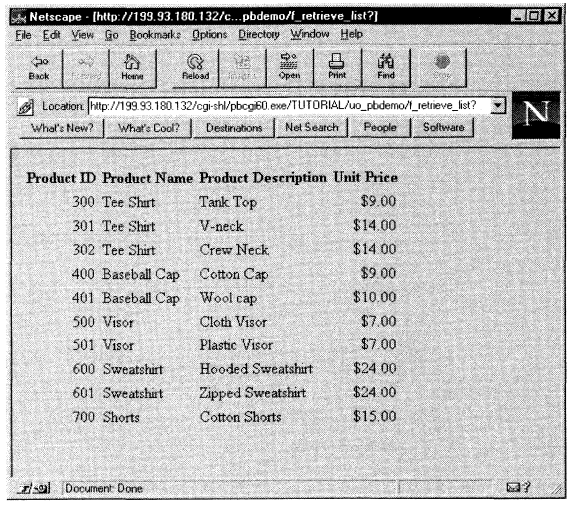
Netscape displays the HTML source for the Web page. Notice the Anchor element specified by <A.



7 Close the HTML source window.

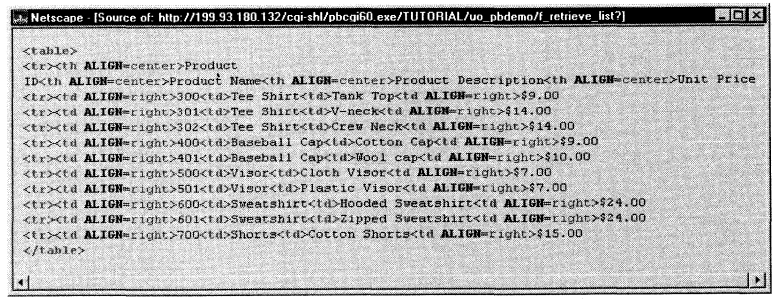
8 Click the *Retrieve product list* link.

The process of executing the function may take a few moments. After processing has finished, the product list displays in the browser window.



9 Select *View>Document Source* from the menu bar.

Netscape displays the HTML source for the Web page.



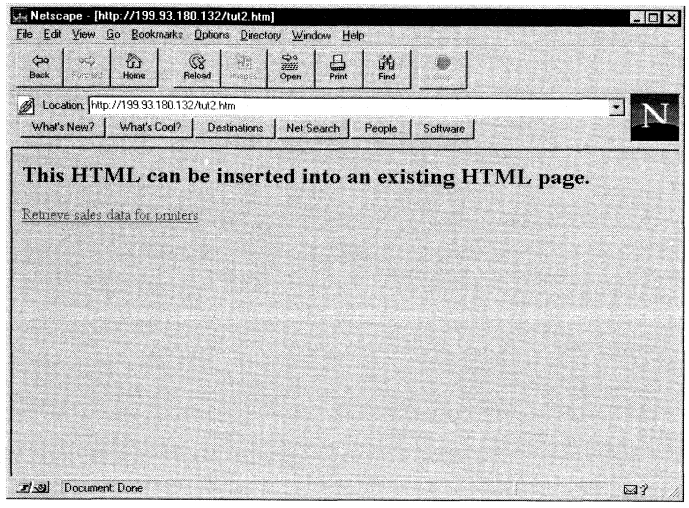
10 Close the HTML source window.

Call the f_crosstab function

1 **In Netscape, select** *File>Open Location* **from the menu bar.**

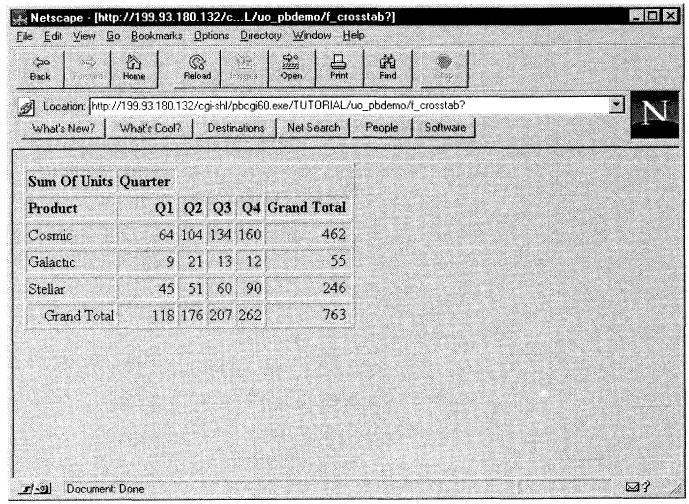
2 **Enter** *http://199.93.180.132/tut2.htm* **in the** *Open Location* **box, substituting your IP address (or a host name) for the address shown here.**

Netscape displays the TUT2.HTM page. Remember that TUT2.HTM has an HTML Anchor element, which is a link to quarterly printer sales.



3 Click the *Retrieve sales data for printers* link.

The quarterly printer sales display.



Sum Of Units	Quarter				
Product	Q1	Q2	Q3	Q4	Grand Total
Cosmic	64	104	134	160	462
Galactic	9	21	13	12	55
Stellar	45	51	60	90	246
Grand Total	118	176	207	262	763

Call the f_name_search function

1 **In Netscape, select** *File>Open Location* **from the menu bar.**

2 **Enter** *http://199.93.180.132/tut3.htm* **in the** *Open Location* **box, substituting your IP address (or a host name) for the address shown here.**

Netscape displays the TUT3.HTM page. Remember that TUT3.HTM has an HTML Form element, which is a form in which a user enters the first letter of the last name of employees.

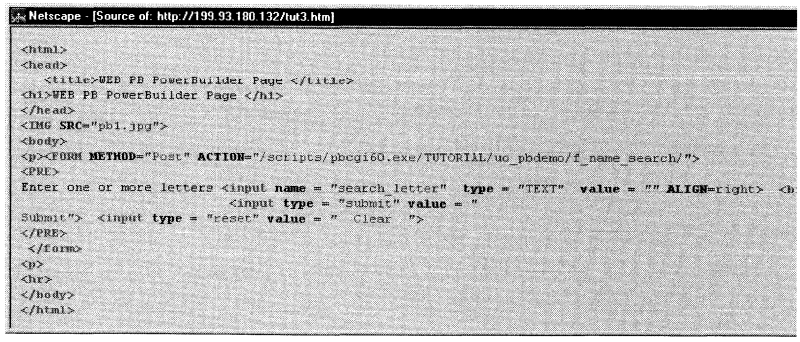
PB1.JPG picture file

The TUT3.HTM page references a picture file named PB1.JPG. To display this picture, the PB1.JPG file must be in the directory in which you are saving HTML files (\HTDOCS is the default directory for WebSite).

FOR INFO For instructions, see "Copying the PB1.JPG file" on page 13.

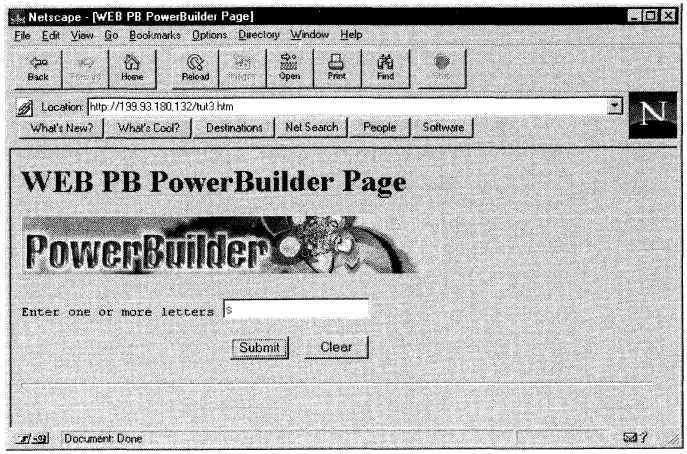
3 **Select** *View>Document Source* **from the menu bar.**

Netscape displays the HTML source for the Web page. Notice the <FORM> element.



4 **Close the HTML source window.**

5 Enter the letter *s* in the form.

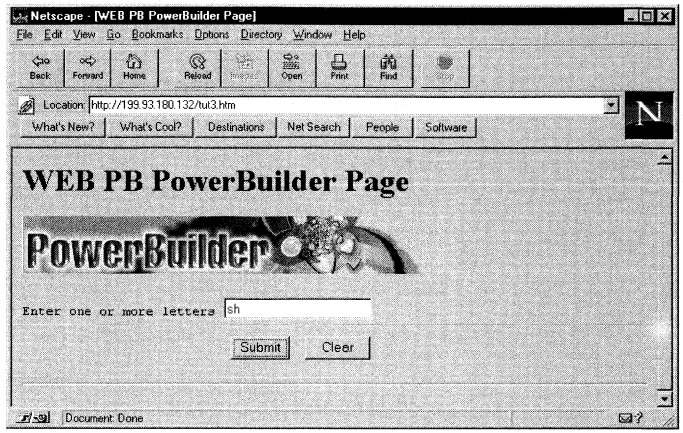


6 Click the *Submit* button.

The employee data for employees with last name starting with *s* displays.

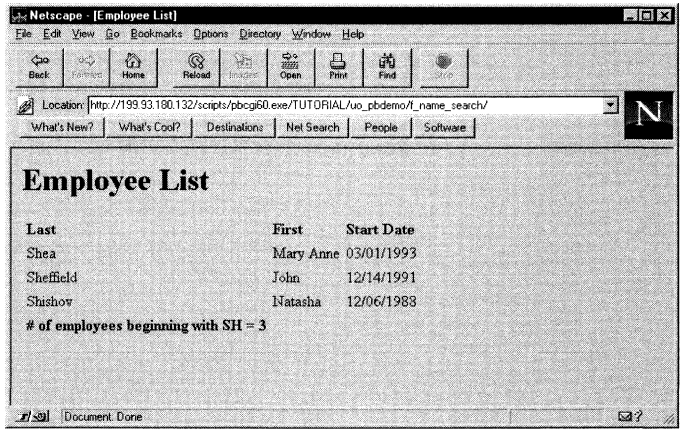
You may need to scroll to the bottom of the page to see the count for the number of employees with last names beginning with S.

7 In Netscape, click the *Back* button on the toolbar. Enter the letters *sh* in the form.



8 Click the *Submit* button.

The employee data for employees with last names starting with SH displays.



9 Stop the PowerBuilder server application.

Lesson 2: creating an interactive Web.PB application

What you will do

In this lesson you will use the Web.PB class library to build an interactive Web application. The Web.PB class library has several objects you can use to generate HTML syntax and maintain session and transaction information. Using the services of the class library allows for greater interaction with the user.

Availability of Web.PB class library

The Web.PB class library including the session management database is available on the Windows and PowerMacintosh platforms but *not* currently on the UNIX platform.

Functions provided for the tutorial

The uo_pbdemo user object in the INTERNET.PBL file has several functions that take advantage of the services of the class library.

In this lesson you will use these functions to:

- Display a logon screen in an HTML page
- Get the user's logon information
- Display a screen that allows the user to specify the type of information to retrieve from the database
- Update the employee table with changes made by the user

If you were working from scratch

If you were creating an application from scratch, you would need to create your own DataWindow objects and write scripts to perform the processing you want. For the tutorial, you do not need to this because the INTERNET.PBL file already has what you need. In the tutorial, you'll look at the DataWindow objects and the functions provided to familiarize yourself with the programming techniques used in the sample application.

Look at the Constructor event

Where you are

> Look at the Constructor event
Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the script for the Constructor event of the uo_pbdemo user object.

1 **Open the User Object painter in PowerBuilder.**
Select the *uo_pbdemo* user object.
Click *OK*.

The nonvisual uo_pbdemo user object displays in the workspace.

2 **Select *Script* from the user object's popup menu.**
Click the *Select Event* listbox and select *constructor*.

The script for the Constructor event displays in the PowerScript painter.

```
gtr_trans = CREATE transaction
gtr_trans.dbms= 'odbc'
gtr_trans.dbparm="ConnectString="&
+"'DSN=Powersoft Demo DB V6;UID=dba;PWD=sql"
CONNECT USING gtr_trans;

sqlca.dbms= 'odbc'
sqlca.dbparm="ConnectString="&
+"'DSN=Powersoft WebPB;UID=dba;PWD=sql"
CONNECT USING sqlca;
IF sqlca.sqlcode <> 0 THEN
   messagebox("sqlca Error:" &
   + String(sqlca.SQLDBCode), sqlca.SQLErrText)
END IF
```

3 **Look at the script.**

The script uses two Transaction objects to connect to two databases:

- The **gtr_trans** Transaction object connects to the Powersoft Demo Database, which contains the business data that the functions in uo_pbdemo access
- The **SQLCA** Transaction object connects to the webpb database, which maintains session and transaction information

Look at the Destructor event

Where you are
Look at the Constructor event
> Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the script for the Destructor event of the uo_pbdemo user object.

1 **Click the *Select Event* listbox and select *destructor*.**

The script for the Destructor event displays in the PowerScript painter.

2 **Look at the script.**

The script disconnects from the Powersoft Demo Database and the webpb database.

```
DISCONNECT USING gtr_trans;
DISCONNECT USING SQLCA;

DESTROY gtr_trans
```

3 **Close the PowerScript painter.**

Look at the instance variables

Where you are
Look at the Constructor event
Look at the Destructor event
> Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the instance variables for the uo_pbdemo user object.

1 Select *Instance Variables* from the user object's popup menu.

PowerBuilder displays the Declare Instance Variables dialog box, which shows the instance variables that were defined for the uo_pbdemo object.

2 Note the following variable declarations:

```
u_html_form inv_html_form
u_session inv_session
u_transaction inv_transaction
```

These variables are used to access objects in the class library. The u_html_form object provides functions for generating HTML syntax. The u_session and u_transaction objects provide functions for maintaining session and transaction information across Web.PB method calls. All three of these objects are autoinstantiated (like PowerBuilder structure objects), so you do *not* need to issue CREATE or DESTROY statements for them.

3 Click *Cancel* to close the Declare Instance Variables dialog box.

Look at the f_displaylogon function

Where you are
Look at the Constructor event
Look at the Destructor event
Look at the instance variables
> Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the f_displaylogon function.

1 **Select *User Object Functions* from the user object's popup menu.**
Select the *f_displaylogon* function.
Click *OK*.

The script for the f_displaylogon function displays in the PowerScript painter.

2 **Look at the script, paying particular attention to these lines:**

```
ls_inputid = &
   inv_html_form.f_MakeSLE("as_userid", 30, 30)
ls_inputpassword = &
   inv_html_form.f_MakeSLE("as_userpass", &
   30, 30, "", TRUE)
ls_formmethodaction = &
   inv_html_form.f_BeginForm &
   ("/cgi-bin/pbcgi60.exe/TUTORIAL" &
   + "/uo_pbdemo/f_getlogon", 0)
ls_inputtypevalue =
   inv_html_form.f_MakeSubmit("Begin Logon")
```

These lines call several functions of the u_html_form user object in the class library. The first two statements call the f_MakeSLE function to create two single line edit elements. The third statement calls the f_BeginForm function to create the FORM and ACTION elements. The final statement calls f_MakeSubmit to create a Submit button.

HTML returned The f_displaylogon function returns the following HTML syntax to the browser:

```
<HTML>
```

```
<HEAD>
<TITLE>Logon to Employee Information
System</TITLE></HEAD>
<BODY><H1>Type your ID and password. </H1>
<FORM ACTION="/cgi-bin/pbcgi60.exe/TUTORIAL
   /uo_pbdemo/f_getlogon" METHOD="GET">
   <P>Internet e-mail address:<INPUT MAXLENGTH =
   "30"
   NAME = "as_userid" SIZE = "30" TYPE = "TEXT">
   <P>Password:<INPUT MAXLENGTH = "30" NAME =
   "as_userpass" SIZE = "30" TYPE = "PASSWORD"
   VALUE="">
   <P><INPUT TYPE="SUBMIT" VALUE="Begin Logon">
</FORM>
</BODY>
</HTML>
```

Look at the f_getlogon function

Where you are
Look at the Constructor event
Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
> Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the script for the f_getlogon function, which is invoked by the Submit button in the form created by f_displaylogon.

1 **Click the *Open* button in the PainterBar.**
Select the *f_getlogon* function.
Click *OK*.

The script for the f_getlogon function displays in the PowerScript painter.

2 **Look at this line:**

```
ll_sessionid = inv_session.f_GenerateID(SQLCA)
```

This line calls the f_GenerateID function of the u_session user object. F_GenerateID creates a new session ID by incrementing the number stored in the www_new_session_id table in the webpb database.

3 **Scroll down until you see this line:**

```
inv_session.f_NewSession(SQLCA, remote_addr, &
   ll_sessionid)
```

This line registers a new session by adding a row to the www_session table.

4 **Now look at these lines:**

```
inv_session.f_SetArgumentValue(SQLCA, &
   ll_sessionid, "userid", as_userid)
inv_session.f_SetArgumentValue(SQLCA, &
```

```
    ll_sessionid, "password", as_userpass)
```

These statements save the user ID and password as session arguments in the www_session_argument table.

5 Look at this line:

```
ls_sessionid = &
   inv_html_form.f_MakeHidden("as_sessionid", &
   String(ll_sessionid))
```

This line creates syntax for a hidden field. The field contains the session ID created by the call to the f_GenerateID function.

6 Now look at these lines:

```
ls_rb1 = "<P>" + &
   inv_html_form.f_MakeRadio("as_choice", &
   "department", TRUE) + "Department Info"
   //ls_rb1 = '<P> <input type="radio"
   //name="as_choice" value="department"
   //CHECKED>Department Info'
ls_rb2 = "<P>" + &
   inv_html_form.f_MakeRadio("as_choice", &
   "employees", FALSE) + "Employee List"
   //ls_rb2 = '<P> <input type="radio"
   //name="as_choice" value="employees">Employee
   //List'
ls_rb3 = "<P>" + &
   inv_html_form.f_MakeRadio("as_choice", &
   "employee", FALSE) + "Employee Info "&
   + "(specify ID)"
   //ls_rb3 = '<P> <input type="radio"
   //name="as_choice" value="employee">Employee
   //Info (specify ID)'
ls_rb4 = "<P>" + &
   inv_html_form.f_MakeRadio("as_choice", &
   "sales", FALSE) + "Sales Report"
   //ls_rb4 = '<P> <input type="radio"
   //name="as_choice" value="sales">Sales Report'
ls_empid = '<P>' &
```

```
    + inv_html_form.f_MakeSLE("as_empid", 5, 5)
```

The first four statements call the f_MakeRadio function to create several radio buttons. These buttons allow the user to specify the type of information to retrieve from the database. The last statement calls f_MakeSLE to create a single line edit so the user can pass an employee ID as an argument.

HTML returned The f_getlogon function returns the following HTML syntax to the browser:

```
<HTML>
<HEAD>
<TITLE>Logon Was Successful</TITLE></HEAD>
<BODY><H1>Select a display option. Then click
OK.</H1><P>
<FORM ACTION="/cgi-bin/pbcgi60.exe/TUTORIAL
   /uo_pbdemo/f_getinfo" METHOD="GET">
   <P><INPUT CHECKED NAME="as_choice" TYPE="RADIO"
   VALUE = "department">
   Department Info<P><INPUT NAME="as_choice"
   TYPE="RADIO" VALUE = "sales">
   Sales Report<P><INPUT NAME="as_choice"
   TYPE="RADIO" VALUE = "employees">
   Employee List<P><INPUT NAME="as_choice"
   TYPE="RADIO" VALUE = "employee">
   Employee Info (specify ID)<P><INPUT MAXLENGTH =
   "5" NAME = "as_empid" SIZE = "5" TYPE = "TEXT">
   <P><INPUT NAME="as_sessionid" TYPE="HIDDEN"
   VALUE = "25"><INPUT TYPE="SUBMIT"
   VALUE="OK"><P>Select an item and click OK.
</FORM>
```

Look at the f_getinfo function

Where you are
Look at the Constructor event
Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
> Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the script for the f_getinfo function, which is invoked by the Submit button in the form created by f_getlogon.

1 **Click the *Open* button in the PainterBar.**
Select the *f_getinfo* function.
Click *OK*.

The script for the f_getinfo function displays in the PowerScript painter.

2 **Find this block of lines:**

```
IF NOT inv_session.f_VerifySessionID &
   (SQLCA, Long(as_sessionid)) THEN
   ls_html += inv_html_form.f_BeginForm &
      ("/cgi-bin/pbcgi60.exe/TUTORIAL" &
      + "uo_pbdemo/f_displaylogon", 0)
   ls_html "= "<P>Session has expired. " &
      + "Return to Logon form."
   ls_html += "<P>" &
       + inv_html_form.f_MakeSubmit("Begin Logon")
   ls_html += inv_html_form.f_EndForm()
   RETURN ls_html
END IF
```

These statements invoke the f_VerifySessionID function to ensure that the specified session is valid. If the session is invalid, the code invokes the f_displaylogon function to make the user log on again.

A session is valid if it exists in the www_session_table and has been updated within the last hour.

3 **Scroll down until you see this line:**

```
inv_session.f_UpdateLastAccess &
   (SQLCA, Long(as_sessionid), Datetime(Now()))
```

This statement updates the www_session table to record the date and time that the specified session was last used.

4 Scroll down until you see this CASE expression:

```
CASE "employee"
```

This CASE expression determines what happens when the user enters an employee ID and selects the radio button labeled Employee Info (specify ID).

5 Find this statement inside the CASE expression:

```
//Emp_id:
ls_return = ls_return + "<P>" + "Employee ID: " &
   + as_empid &
   + inv_html_form.f_MakeHidden("as_empid", &
   + as_empid)
```

If the requested employee row exists in the database, this statement adds the employee ID that was retrieved as a text field in the form. It also creates a hidden field to pass the employee ID value to the f_update function.

6 Scroll down until you see these lines:

```
ll_transid = inv_transaction.f_GenerateID(SQLCA)
inv_transaction.f_NewTransaction &
   (SQLCA, Long(as_sessionid), ll_transid)
inv_transaction.f_SetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   ll_transid, "deptid", ls_deptid)
inv_transaction.f_SetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   ll_transid, "emplname", ls_emplname)
inv_transaction.f_SetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   ll_transid, "empfname", ls_empfname)
inv_transaction.f_SetArgumentValue &
   (SQLCA, Long(as_sessionid), &
```

```
    ll_transid, "status", ls_status)
```

The first two statements create a new transaction ID and register the new transaction by adding a row to the www_transaction table. The next four statements save the department ID, last name, first name, and status values retrieved for the requested employee in the www_transaction_argument table.

7 Now look at the following lines:

```
ls_return += &
   inv_html_form.f_MakeHidden &
   ("as_sessionid",String(ll_sessionid))
ls_return += &
   inv_html_form.f_MakeHidden &
   ("as_transactionid", String(ll_transid))
```

These lines create hidden fields to pass the session ID and transaction ID to the f_update function. If the user changes any of the employee data values and presses the Update button, the f_update function uses the session ID and transaction ID to access the original employee values in the webpb database.

HTML returned The HTML syntax the f_getinfo function returns to the browser varies depending on which employee ID is specified. For example, if you request employee 102, the f_getinfo function returns the following HTML:

```
<HTML>
<HEAD><TITLE>Employee Information</TITLE></HEAD>
<BODY><H1>Employee Information</H1>
<FORM ACTION="/cgi-bin/pbcgi60.exe/TUTORIAL
   /uo_pbdemo/f_update" METHOD="GET">
   <P>Employee ID: 102<INPUT NAME="as_empid"
   TYPE="HIDDEN" VALUE = "102">
   <P>Department ID: <INPUT MAXLENGTH = "5" NAME =
   "as_deptid" SIZE = "5" TYPE = "TEXT"
   VALUE="100">
   <P>Last Name: <INPUT MAXLENGTH = "30" NAME =
   "as_emplname" SIZE = "30" TYPE = "TEXT"
   VALUE="Whitney">
   <P>First Name: <INPUT MAXLENGTH = "30" NAME =
   "as_empfname" SIZE = "30" TYPE = "TEXT"
   VALUE="Fran">
   <P>Status: <INPUT CHECKED NAME="as_status"
```

```
    TYPE="RADIO" VALUE = "Active">
    Active<P><INPUT NAME="as_status" TYPE="RADIO"
    VALUE = "On Leave">
    On Leave<P><INPUT NAME="as_status" TYPE="RADIO"
    VALUE = "Terminated">
    Terminated<P><INPUT TYPE="SUBMIT"
    VALUE="Update">
    <INPUT NAME="as_sessionid" TYPE="HIDDEN"
    VALUE = "26">
    <INPUT NAME="as_transactionid" TYPE="HIDDEN"
    VALUE = "48">
</FORM>
</BODY>
</HTML>
```

Look at the f_update function

Where you are
Look at the Constructor event
Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
> Look at the f_update function
Use the Web.PB Wizard to create the logon page
Run the application

Now you will look at the script for the f_update function, which is invoked by the Submit button in the form created by f_getinfo.

1 Click the *Open* button in the PainterBar.
Select the *f_update* function.
Click *OK*.

The script for the f_update function displays in the PowerScript painter.

2 Find this block of lines:

```
ls_deptid = inv_transaction.f_GetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   Long(as_transactionid), "deptid")
ls_emplname - inv_transaction.f_GetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   Long(as_transactionid), "emplname")
ls_empfname = inv_transaction.f_GetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   Long(as_transactionid), "empfname")
ls_status = inv_transaction.f_GetArgumentValue &
   (SQLCA, Long(as_sessionid), &
   Long(as_transactionid), "status")
```

These statements use the session ID and transaction ID passed in as hidden fields to access the original employee values stored in the webpb database.

3 Now look at these lines:

```
IF NOT as_deptid = ls_deptid THEN
   lb_change = TRUE
```

```
ELSEIF NOT as_emplname = ls_emplname THEN
   lb_change = TRUE
ELSEIF NOT as_empfname = ls_empfname THEN
   lb_change = TRUE
ELSEIF NOT ls_status_code = ls_status THEN
   lb_change = TRUE
END IF
```

These lines set a flag if the user changed any of the employee values retrieved from the database.

4 Now look at these lines:

```
IF lb_change = TRUE THEN
   UPDATE "employee"
      SET "employee"."dept_id" = :ll_deptid,
      "employee"."emp_lname" = :as_emplname,
      "employee"."emp_fname" = :as_empfname,
      "employee"."status" = :ls_status_code
      WHERE "employee"."emp_id" = :ll_empid
      USING gtr_trans;

   IF gtr_trans.SQLCode = 0 THEN
      COMMIT using gtr_trans;
      RETURN "Update succeeded!" &
      + "<P>Session ID: " + as_sessionid &
      + "<P>Transaction ID: " + as_transactionid &
      + "<P>Saved Dept ID: " + ls_deptid &
      + "<P>New Dept ID: " + as_deptid &
      + "<P>Saved Emp Lname: " + ls_emplname &
      + "<P>New Emp Lname: " + as_emplname &
      + "<P>Saved Emp Fname: " + ls_empfname &
      + "<P>New Emp Fname: " + as_empfname &
      + "<P>Saved Status: " + ls_status &
      + "<P>New Status: " + ls_status_code
   ELSE
      RETURN "Update failed!" &
      + "<P>" + string(gtr_trans.SQLCode) &
      + "<P>" + string(gtr_trans.SQLDBCode) &
      + "<P>" + gtr_trans.SQLErrText
   END IF
ELSE
   RETURN "No changes were made."
```

```
END IF
```

These statements check to see whether the update flag is set. If it is, they update the database with the user's changes. If the update flag is not set, the script returns a message indicating that no changes were made.

HTML returned The f_update function returns HTML to the browser that indicates the success or failure of the update operation.

Use the Web.PB Wizard to create the logon page

Where you are
Look at the Constructor event
Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
> Use the Web.PB Wizard to create the logon page
Run the application

Now you will use the Web.PB Wizard to create a Web page that will include a hypertext link to invoke the f_displaylogon function.

1 **Run the Wizard.**

2 **If necessary, enter the full path of the INTERNET.PBL file.**
or
Use the Browse button to locate and select the PBL.

3 **Click** *Next.*

The Wizard displays the nonvisual user objects in the PBL that return strings or blobs.

4 **Select** *uo_pbdemo.*
Click *Next.*

5 **Select the** *f_displaylogon* **function.**
Click *Next.*

6 **Click the** *Standard CGI (any Web Server)* **radio button.**
Click *Next.*

7 **Select** *TUTORIAL* **in the** *Select Server Alias Section* **listbox.**
Click *Next.*

8 **Type** *Display Logon Form* **in the** *HREF Text* **box.**

9 **Enter *C:\WEBSITE\HTDOCS\LOGON.HTM* for the location and name of the HTML file you are creating.**

The Wizard displays the name of the HTML file in the Specify Location and Name of HTML File box.

10 **Click *Next*.**
Click *Finish*.

Run the application

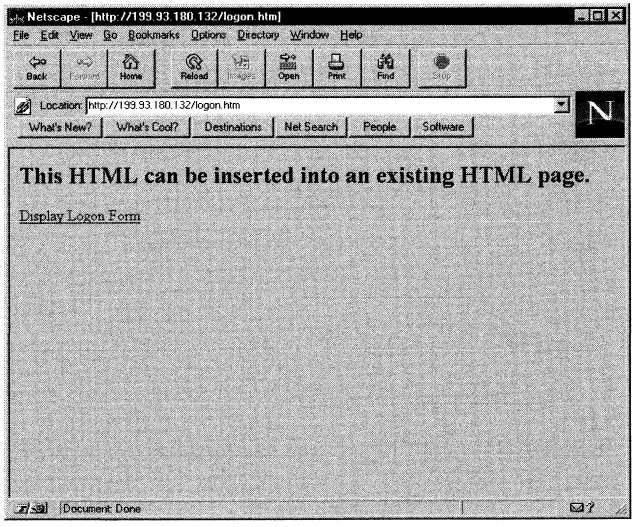
Where you are
Look at the Constructor event
Look at the Destructor event
Look at the instance variables
Look at the f_displaylogon function
Look at the f_getlogon function
Look at the f_getinfo function
Look at the f_update function
Use the Web.PB Wizard to create the logon page
> Run the application

Now you will run the application.

1 **Start the WebSite server if it is not already running.**
Run the PowerBuilder server application.
In Netscape, select *File>Open Location* **from the menu bar.**

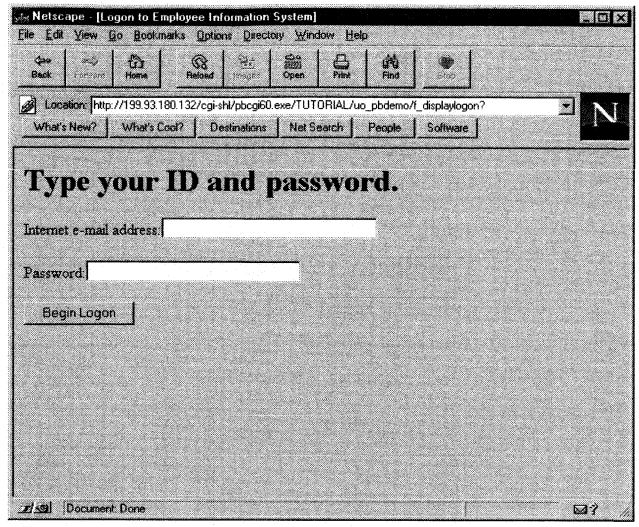
2 **Type** *http://199.93.180.132/logon.htm*, **substituting your IP address (or a host name) for the address shown here.**

The browser displays the LOGON.HTM page.



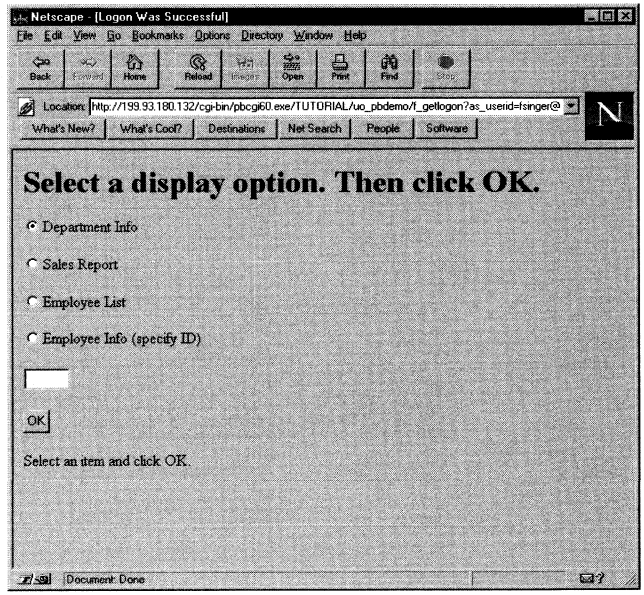
3 Click the *Display Logon Form* link.

The browser displays the logon form.



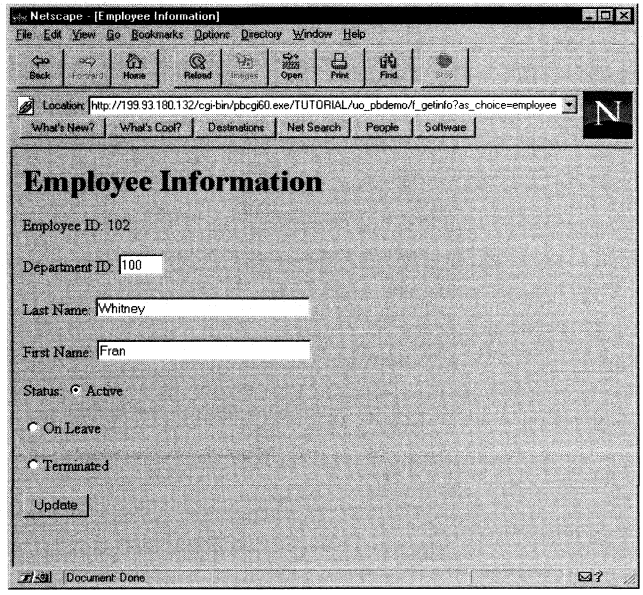
4 **Type your e-mail address in the** *Internet e-mail address* **box.**
Type some text in the *Password* **box.**
Click *Begin Logon.*

The browser displays a form that allows you to select the data you want to see.



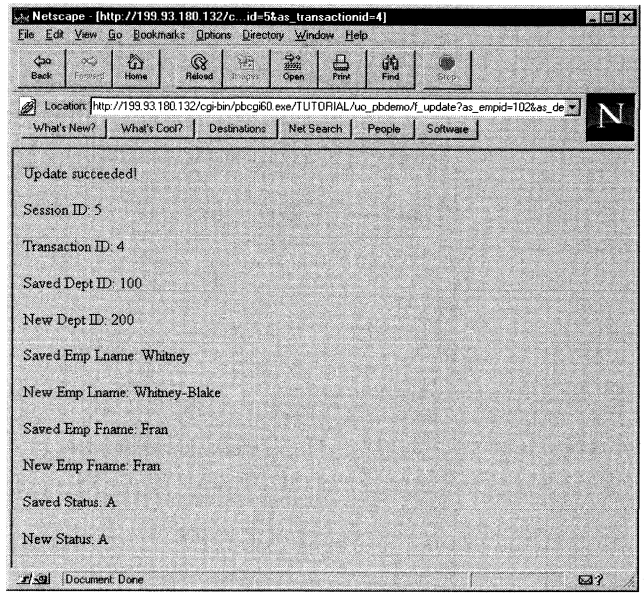
5 **Click** *Employee Info (specify ID).*
Type *102* **in the single line edit box.**
Click *OK.*

The browser displays the employee information form, showing data for employee 102.



6 Change one or more of the data values for employee 102. Click *Update*.

The browser displays a new HTML page that indicates the success or failure of the update operation.



What to do next

Congratulations. You have completed the Web.PB tutorial. Now you know the basics of application development with Web.PB.

To further your education on Web.PB, read the remaining chapters in this part.

CHAPTER 4

Using Web.PB

About this chapter

This chapter describes how to use Web.PB to build Web applications.

Contents

Configuring your Web server for Web.PB

Supported program interfaces

Web.PB works with a variety of program interfaces depending on the operating system running on the Web server where Web.PB is installed. To use Web.PB, your Web server must support the program interface in use on your platform:

Web.PB program interface	Supported platforms
Common Gateway Interface (CGI)	Windows UNIX
Internet Server API (ISAPI)	Windows
Netscape Server API (NSAPI)	Windows
WebSTAR API (WSAPI)	Power Macintosh

Web.PB program files

Each supported Web.PB program interface has a corresponding program file that the Web server calls to service Web.PB requests from a browser. The Web.PB program files *must* reside on the Web server machine.

Web.PB supplies different program interface files on different operating system platforms:

Program interface	On Windows	On UNIX	On Power Macintosh
Standard CGI	PB6\IT\BIN\PBCGI60.EXE	pb6/pb6web/bin/pbcgi60.exe (invoked by pbcgi60.script)	—
ISAPI for the Microsoft Internet Information Server (IIS)	PB6\IT\BIN\PBISA60.DLL	—	—
NSAPI for the Netscape Commerce and Communications servers	PB6\IT\BIN\PBNS160.DLL	—	—
NSAPI for the Netscape FastTrack and Enterprise servers	PB6\IT\BIN\PBNS260.DLL	—	—
WSAPI for servers that support the WebSTAR API	—	—	Powersoft 6.0 Folder: Internet Tools:WebPB: WebPB

Setting up the Web.PB program files

CGI support (Windows and UNIX) To allow Web.PB to use standard CGI to work with your Web server, make sure that the PBCGI60.EXE file (on Windows) or the pbcgi60.script file (on UNIX) is in the directory where the Web server expects to find CGI programs.

Most Web servers require that CGI programs be located in a subdirectory of the server's root directory. The name of this subdirectory depends on the Web server you are using and how you configure the server.

ISAPI or NSAPI support (Windows) To provide support for either ISAPI or NSAPI, you must copy the PBISA60.DLL, PBNS160.DLL, or PBNS260.DLL file (depending on the interface you are using) to the program directory.

WSAPI support (Macintosh) To provide support for WSAPI on Power Macintosh, you need a Web server that supports the WebSTAR API, such as StarNine WebSTAR or Microsoft Personal Web Server.

You must also copy the WebPB shared library to your Web server's Plug-Ins folder. If the PowerBuilder Installer finds a Web server on your machine, it will install a copy of the WebPB shared library in the server's Plug-Ins folder as part of the Web.PB installation.

CGI mappings

To allow the Web server to service all program requests, you may need to edit the CGI mappings for your Web server.

Additional configuration steps

Some Web servers require you to perform additional steps to register Web.PB. The rest of this section provides instructions for configuring the following Web servers to run Web.PB applications:

Web server	Platform	See
WebSite	Windows	"Configuring WebSite (Windows)" on page 110
Microsoft Internet Information Server (IIS)	Windows	"Configuring the Microsoft Internet Information Server (Windows)" on page 111
Netscape Commerce and Communications servers	Windows	"Configuring the Netscape Commerce and Communications servers (Windows)" on page 112
Netscape FastTrack and Enterprise servers	Windows	"Configuring the Netscape FastTrack and Enterprise servers (Windows)" on page 115
Apache HTTP server	UNIX	"Configuring the Apache HTTP server (UNIX)" on page 117

Web server	Platform	See
Netscape Enterprise Server 2.0	UNIX	"Configuring the Netscape Enterprise Server (UNIX)" on page 121

Configuring WebSite (Windows)

To configure the WebSite server on Windows to run Web.PB applications, perform the following setup procedures.

Editing the CGI mappings

To allow WebSite to service various kinds of program requests, you may need to edit the CGI mappings. For example, to handle requests that specify /SCRIPTS/ in a URL path, you need to map /SCRIPTS/ to \CGI-SHL (or whatever directory you use to manage CGI programs).

❖ **To edit the CGI mappings for WebSite:**

1. Make sure that WebSite is installed and operating correctly.
2. Run the Server Admin program.
3. Select the Mapping tab.
4. In the List Selector groupbox, select the Standard CGI radio button.
5. Type the URL path in the Standard CGI URL Path box (for example, /SCRIPTS/).
6. Type the directory path in the Directory (full or server-relative) box (for example, C:\WEBSITE\CGI-SHL\).
7. Click OK.

Providing CGI support

To handle standard CGI requests, you must put PBCGI60.EXE in the \CGI-SHL directory (where WebSite expects to find CGI programs). Before you begin working with Web.PB, make sure that PBCGI60.EXE is in the correct directory.

Configuring the Microsoft Internet Information Server (Windows)

To configure the Microsoft Internet Information Server (IIS) on Windows to run Web.PB applications, perform the following setup procedures.

Editing the CGI mappings

The default installation of the Microsoft Internet Information Server (IIS) creates a directory called \INETSRV and uses this as its root directory. IIS typically manages CGI programs in a subdirectory of \INETSRV called \SCRIPTS. Whenever an HTML document makes a program request in a URL path, IIS uses the mapping list to map the URL path to the \SCRIPTS directory (or whatever directory you use to manage CGI programs.)

To ensure that the Web server can service program requests that specify the /SCRIPTS/ URL path, you may need to edit the CGI mappings.

❖ **To edit the CGI mappings for IIS:**

1. Make sure the Microsoft Internet Information Server is installed and operating correctly.
2. Run the Internet Service Manager program.
3. Select the entry for your Web server that shows WWW in the Service column and Running in the State column.
4. Select Properties>Service Properties from the menu bar.
5. Select the Directories tab.
6. Verify that there is a directory item with an alias called /SCRIPTS/.
7. Select the /SCRIPTS/ alias and click the Edit Properties button.
8. Make sure that Execute is the only item selected in the Access group box.
9. If you've made any changes to the definition for the /SCRIPTS/ alias, click OK. Otherwise, click Cancel.

If you do not find a directory item with an alias called /SCRIPTS/, you need to add one. The /SCRIPTS/ alias needs to point to the \SCRIPTS directory (or whatever directory you use to manage CGI programs).

Providing standard CGI support

To provide support for standard CGI, you must put PBCGI60.EXE in the directory where CGI programs reside. This is the directory you specified for the /SCRIPTS/ alias in the Internet Service Manager program.

Providing ISAPI support

To provide support for ISAPI, you must put PBISA60.DLL in the directory where CGI programs reside.

Configuring the Netscape Commerce and Communications servers (Windows)

To configure the Netscape Commerce or Communications server on Windows to run Web.PB applications, perform the following setup procedures.

Editing the CGI mappings

The default installation of the Netscape Commerce and Communications servers creates a directory called \NS-HOME and uses this as its root directory. The Netscape servers typically manage CGI programs in a subdirectory of \NS-HOME called \BIN. Whenever an HTML document makes a program request in a URL path, the server uses the CGI mapping list to map the URL path specified to the \BIN directory.

To ensure that the Web server can service program requests that specify /CGI-BIN/ in the URL path, you need to edit the CGI mappings.

❖ **To edit the CGI mappings for the Netscape Commerce and Communications servers:**

1. Make sure the Netscape server is installed and operating correctly. Be sure the server is down.
2. From the Netscape folder, start the Configuration program.

   Netscape displays an HTML form that allows you to perform various system administration tasks for the server.
3. Click the link for the server and port you want to configure.
4. Scroll to the CGI and Server-Parsed HTML section.

5 In the CGI and Server-Parsed HTML section, click the link labeled Specify a Directory That Will Contain CGI Programs Only.

6 Scroll to the URL Prefix section of the form.

7 In the URL Prefix section, fill in the entry field with the prefix that maps to the directory for CGI programs.

HTML documents can specify the URL Prefix to invoke CGI programs such as Web.PB. The URL prefix is a relative URL path (relative to the server root). Typically, you want the URL prefix to be CGI-BIN.

8 In the CGI Directory section, fill in the entry field with the fully qualified path for the directory that contains CGI programs.

Typically, you want the CGI Directory to point to the \BIN subdirectory of the Netscape server's root directory.

9 Click the Make These Changes button to save your changes.

Providing standard CGI support

To provide support for standard CGI, you must put PBCGI60.EXE in the directory where CGI programs reside. This is the directory you specified in the CGI Directory entry field in the Configuration program.

Providing NSAPI support

Copy PBNS160.DLL

To provide NSAPI support for Netscape Commerce or Communications servers, you must put PBNS160.DLL in the directory where CGI programs reside.

Copy WEBAGENT.PBW

To trigger the PBNS160.DLL file, you need to include a file on the server machine that has the extension PBW, which is the Powersoft Web.PB file extension. Any URL requests that reference files with this extension trigger Web.PB. Whenever an HTML document specifies a URL reference to this file, the server invokes PBNS160.DLL to service the request. The content of the PBW file is not important; the file can be empty.

Web.PB provides a file called WEBAGENT.PBW that you can use. Put this file in a directory on the server machine that you would like users to be able to access. A logical place for WEBAGENT.PBW is the directory where you maintain HTML documents.

Edit the MIME.TYPES file

To use Web.PB with NSAPI, you need to register PBNS160.DLL as a custom server function that is associated with the file extension PBW. To do this, you first create a new entry in the MIME.TYPES file.

❖ **To edit the MIME.TYPES file:**

1 In the \NS-HOME\HTTPD-*port#*\CONFIG directory, open the MIME.TYPES file using a text editor that handles long filenames.

2 At the end of the MIME.TYPES file, add the following entry:

```
type=magnus-internal/pbw exts=pbw
```

3 Save the file.

Update the Windows registry

To make Web.PB handle NSAPI requests, you must update the Windows registry.

❖ **To update the Windows registry:**

1 Run REGEDT32.EXE.

2 Under HKEY_LOCAL_MACHINE\Software\Netscape\Httpd-*port#*\CurrentVersion\Startup, add a new key called InitFunction01, specifying REG_SZ (for string data) as the class.

3 Add these value names and values (strings) to the key just added:

Value name	Value (string)
fn	load-modules
shlib	{fully qualified path to PBNS160.DLL} The path must use forward slashes, not backslashes (for example, C:/NETSCAPE/NS-HOME/BIN/PBNS160.DLL)
funcs	PBWebAgent

4 Under HKEY_LOCAL_MACHINE\Software\Netscape\Https-*port#*\CurrentVersion\Objects\Object1 (or whatever object has the lowest number), check to see that you have a Service Directive key. If you do, go to step 7.

5 If you do not have a Service Directive key, add one below the object with the lowest number. Give the key a value that has the format Directive*nn*, where *nn* is a two-digit number that has not been specified previously.

6 Add the following value name and value (string) to the key just added:

Value name	Value (string)
DirectiveName	Service

7 Under the Service Directive key, add a Function key. Give the key a value that has the format Function*nn*, where *nn* is a two-digit number that has not been specified previously. The number you specify needs to be the lowest number in the Function key list.

8 Add these value names and values (strings) to the Function key just added:

Value name	Value (string)
fn	PBWebAgent
method	(GET|POST|HEAD)
type	magnus-internal/pbw

Configuring the Netscape FastTrack and Enterprise servers (Windows)

To configure the Netscape FastTrack or Enterprise Server on Windows to run Web.PB applications, perform the following setup procedures.

Editing the CGI mappings

The default installation of the Netscape FastTrack and Enterprise servers creates a directory called \SERVER and uses this as its root directory. The Netscape servers typically manage CGI programs in a subdirectory of \SERVER called \BIN. Whenever an HTML document makes a program request in a URL path, the server uses the CGI mapping list to map the URL path specified to the \BIN directory.

To ensure that the Web server can service program requests that specify /CGI-BIN/ in the URL path, you need to edit the CGI mappings.

❖ **To edit the CGI mappings for the Netscape FastTrack and Enterprise servers:**

1 Make sure the Netscape server is installed and operating correctly. Be sure the server is down.

2 From the Netscape folder, start the Administer Netscape Server program.

Netscape displays an HTML form that allows you to perform various system administration tasks for the server.

3 Select the server name you want to configure.

4 Click the Programs button in the top frame.

5 Fill in the URL prefix entry field with the prefix that maps to the home directory for CGI programs.

   HTML documents can specify the URL Prefix to invoke CGI programs such as Web.PB. The URL prefix is a relative URL path (relative to the server root). Typically, you want the URL prefix to be CGI-BIN.

6 Fill in the CGI Directory entry field with the fully qualified path for the directory that contains CGI programs.

   Typically, you want the CGI directory to point to the \BIN subdirectory of the Netscape server's root directory.

7 Click the OK button.

8 Click the Save And Apply button.

Providing standard CGI support

To provide support for standard CGI, you must put PBCGI60.EXE in the directory where CGI programs reside. This is the directory you specified in the CGI Directory entry field in the Administer Netscape Server program.

Providing NSAPI support

Copy PBNS260.DLL

To provide NSAPI support for Netscape FastTrack or Enterprise servers, you must put PBNS260.DLL in the directory where CGI programs reside.

Copy WEBAGENT.PBW

To trigger the PBNS260.DLL file, you need to include a file on the server machine that has the extension PBW, which is the Powersoft Web.PB file extension. Any URL requests that reference files with this extension trigger Web.PB. Whenever an HTML document specifies a URL reference to this file, the server invokes PBNS260.DLL to service the request. The content of the PBW file is not important; the file can be empty.

Web.PB provides a file called WEBAGENT.PBW that you can use. Put this file in a directory on the server machine that you would like users to be able to access. A logical place for WEBAGENT.PBW is the directory where you maintain HTML documents.

Edit the MIME.TYPES file

To use Web.PB with NSAPI, you need to register PBNS260.DLL as a server plug-in function that is associated with the file extension PBW. To do this, you first create a new entry in the MIME.TYPES file.

❖ **To edit the MIME.TYPES file:**

1 In the \SERVER\HTTPS-*hostname*\CONFIG directory, open the MIME.TYPES file using a text editor that handles long filenames.

2 At the end of the MIME.TYPES file, add the following line:

```
type=magnus-internal/pbw exts=pbw
```

3 Save the file.

Edit the OBJ.CONF file

To make Web.PB handle NSAPI requests, you must edit the OBJ.CONF file.

❖ **To edit the OBJ.CONF file:**

1 In the \SERVER\HTTPS-*hostname*\CONFIG directory, open the OBJ.CONF file using a text editor that handles long filenames.

2 Add the following line to the Init section:

```
fn="load-modules" shlib="path/PBNS260.DLL"
   funcs="PBWebAgent"
```

The path to the PBNS260.DLL file needs to be fully qualified (for example, C:/NETSCAPE/SERVER/BIN/PBNS260.DLL). The path must use forward slashes, not backslashes.

3 Add the following Service directive entry to the <Object name="default"> section:

```
Service fn="PBWebAgent" method="(GET|POST)"
   type="magnus-internal/pbw"
```

4 Save the file.

Configuring the Apache HTTP server (UNIX)

Apache is a freely available HTTP server that you can download from the World Wide Web and install on UNIX machines. To configure the Apache HTTP server on UNIX to run Web.PB applications, perform the following setup procedures.

See Apache documentation for information

FOR INFO For complete information about configuring the Apache HTTP server, see the Apache documentation available from the Apache HTTP Server Project site on the World Wide Web (currently http://www.apache.org).

Editing the CGI mappings

The Apache HTTP server typically manages CGI programs in a subdirectory of the server's root directory called /cgi-bin. Whenever an HTML document makes a program request in a URL path, the server maps the specified URL path to the /cgi-bin directory.

To make sure Apache can service programs that specify /CGI-BIN/, /CGI-SHL/, or /SCRIPTS/ in the URL path, you must create the /cgi-bin subdirectory (if it doesn't already exist), specify the CGI mappings, and restart the server.

Create the /cgi-bin subdirectory

The Apache HTTP server expects CGI programs to be in a subdirectory of the server's root directory called /cgi-bin. If the /cgi-bin directory doesn't already exist, you must manually create it before editing the CGI mappings.

❖ **To create the /cgi-bin subdirectory if it doesn't already exist:**

- Create a subdirectory called /cgi-bin under the Apache root directory. Make sure the user running the Web server process has execute permission on this directory.

  For example, in a C shell type:

  ```
  mkdir /usr/apache/cgi-bin
  ```

Specify the CGI mappings

The Apache HTTP server provides configuration files in a subdirectory called /conf under the server's root directory. You must copy and edit these files as described in the following procedure to enable Apache to service CGI programs.

Other editing may be required in configuration files

To correctly configure the Apache server, you may need to edit the configuration files to specify values for directives *other* than the ones described here for CGI.

FOR INFO For detailed instructions, see the comments in these files and your Apache documentation.

❖ **To specify the CGI mappings for the Apache HTTP server:**

1 Make sure the Apache HTTP server is installed and operating correctly on your UNIX Web server machine.

FOR INFO For instructions, see your Apache documentation.

2 If you haven't already done so, copy the distribution versions of the Apache configuration files to the following files (all files are located in the /conf subdirectory under the server's root directory):

Copy this file	To this file
conf/httpd.conf-dist	conf/httpd.conf
conf/srm.conf-dist	conf/srm.conf
conf/access.conf-dist	conf/access.conf

3 Make sure the httpd.conf file is correctly edited to set up general server attributes.

You must edit httpd.conf to set up general server attributes such as the port number, user and group identification, server's root directory, and server name.

FOR INFO For detailed instructions, see your Apache documentation.

4 Edit the srm.conf file to uncomment and specify values for both of the following directives pertaining to CGI support:

ScriptAlias The ScriptAlias directive specifies one or more aliases that map to the home directory for CGI programs. This alias will appear as the directory specification for CGI programs in URLs. Use the following syntax:

ScriptAlias *URL_path* *CGI_program_directory*

For example:

```
ScriptAlias  /cgi-bin  /usr/apache/cgi-bin
ScriptAlias  /cgi-shl  /usr/apache/cgi-bin
ScriptAlias  /scripts  /usr/apache/cgi-bin
```

AddHandler Uncomment the AddHandler directive to map the .cgi file extension to the handler for CGI programs. This directive tells Apache to treat files ending with .cgi as CGI programs.

```
AddHandler  cgi-script  .cgi
```

5 Edit the access.conf file to uncomment and specify values for the following directive pertaining to CGI support:

<Directory> (for CGI programs) Specify the full path of the directory containing CGI programs. This is the name of the subdirectory you created earlier and specified for the ScriptAlias directive in the srm.conf file (see step 4). Typically, you want this directory to point to the /cgi-bin directory under the server's root directory. For example:

```
<Directory /usr/apache/cgi-bin>
AllowOveride None
Options None
```

You can also specify which hosts are allowed and denied access to programs in this directory. For example, the lines shown in bold tell Apache to evaluate allow directives before deny directives and to allow all hosts access to this directory.

```
<Directory /usr/apache/cgi-bin>
AllowOveride None
order allow,deny
allow from all
Options None
```

Restart the server

You must restart the Apache HTTP server after specifying the CGI mappings.

❖ **To restart the Apache HTTP server:**

1 Find the process ID (PID) of the parent httpd process in the logs/httpd.pid file. For example, type:

```
cat /usr/apache/logs/httpd.pid
```

The PID for the parent httpd process displays. For example:

```
5958
```

2 Using this PID, issue the following command to restart the server and read the new configuration parameters:

kill -HUP *httpd_pid*

For example, type:

```
kill -HUP 5958
```

Providing standard CGI support

In Web.PB on UNIX, the pbcgi60.script file runs the pbcgi60.exe binary image in the Web.PB /bin directory. To provide support for standard CGI, you must copy the pbcgi60.script file to the directory where your Web server expects to find CGI programs. Typically, this is the /cgi-bin subdirectory that you created under your Web server's root directory.

The pbcgi60.exe file should remain in the Web.PB /bin directory.

❖ **To provide standard CGI support with Apache HTTP server:**

- Copy the pbcgi60.script file from the Web.PB /bin directory to the /cgi-bin subdirectory under the Apache server's root directory.

  Make sure that the user running the Web server process has execute permission on this script.

  For example, type:

```
cp /usr/pb6/pb6web/bin/pbcgi60.script
   /usr/apache/cgi-bin
```

Configuring the Netscape Enterprise Server (UNIX)

To configure the Netscape Enterprise Server Version 2.0 on UNIX to run Web.PB applications, perform the following setup procedures.

Editing the CGI mappings

The default installation of the Netscape Enterprise Server Version 2.0 on a UNIX Web server creates a directory called /ns-home and uses this as its root directory. The Netscape Enterprise Server typically manages CGI programs in a subdirectory of /ns-home called /cgi-bin. Whenever an HTML document makes a program request in a URL path, the server maps the specified URL path to the /cgi-bin directory.

To ensure that the Netscape Enterprise Server can service programs that specify /CGI-BIN/ in the URL path, you must create the /cgi-bin directory (if it doesn't already exist), specify the CGI mappings, and restart the server.

Create the /cgi-bin subdirectory

The Netscape Enterprise Server expects CGI programs to be in a subdirectory of the /ns-home root directory called /cgi-bin. If the /cgi-bin directory doesn't already exist, you must manually create it before editing the CGI mappings.

❖ **To create the /cgi-bin directory if it doesn't already exist:**

- Create a subdirectory called /cgi-bin under the Netscape Enterprise Server /ns-home root directory. Make sure the user running the Web server process has execute permission on this directory.

  For example, in a C shell type:

  ```
  mkdir /usr/ns-home/cgi-bin
  ```

Specify the CGI mappings

You use the Netscape Enterprise Server's Administration Server to specify the CGI mappings.

❖ **To specify the CGI mappings for the Netscape Enterprise Server:**

1. Make sure the Netscape Enterprise Server is installed and operating correctly on your UNIX Web server machine.

   FOR INFO For instructions, see your Netscape Enterprise Server documentation.

2. Start the Administration Server if it is not already running. From the server's root directory, type:

   ```
   ./start-admin
   ```

3. Start your Web browser and enter the URL for the Administration Server, using the following URL syntax:

   http://*servername.your_domain.domain:admin_port_number*

   For example:

   ```
   http://server01.sybase.com:12345
   ```

   Make sure you use the port number for the Administration Server that you specified during the installation. This is *not* the same as the port number for the Netscape Enterprise Server itself.

4. Supply the user ID and password for the Administration Server when prompted to do so.

   The Netscape Server Selector displays.

5. In the Server Selector window, click the name of the server you want to configure.

6. Click the Programs button in the top frame.

   The CGI Directory form displays for you to specify CGI mappings.

7. Specify both of the following values in the CGI Directory form:

URL prefix This prefix maps to the home directory for CGI programs. This will appear as the directory specification for CGI programs in URLs. The URL prefix is a path relative to the server's root directory. Typically, you want the URL prefix to be cgi-bin. For example:

```
cgi-bin
```

If you want, you can also enter **scripts** and **cgi-shl** as URL prefixes and map each of these to the /cgi-bin directory. This ensures that Netscape Enterprise will be able to service programs that specify /SCRIPTS/ or /CGI-SHL/ in the URL path.

CGI directory This is the location of the directory containing CGI programs and is the name of the subdirectory you created earlier. Typically, you want the CGI directory to point to the /cgi-bin subdirectory under the server's root directory. For example:

```
/usr/ns-home/cgi-bin
```

8 Click OK. Then save and apply your changes.

Restart the server

You must restart the Netscape Enterprise Server after specifying the CGI mappings.

❖ **To restart the Netscape Enterprise Server:**

1 Click the Choose button in the top frame to return to the Server Selector.

2 Click the On icon (toggle switch) to the left of the server's name.

The icon changes to Off to indicate that the server has stopped.

3 Click the Off icon to restart the server.

The icon changes to On to indicate that the server has restarted.

Providing standard CGI support

In Web.PB on UNIX, the pbcgi60.script file runs the pbcgi60.exe binary image in the Web.PB /bin directory. To provide support for standard CGI, you must copy the pbcgi60.script file to the directory where your Web server expects to find CGI programs. Typically, this is the /cgi-bin subdirectory that you created under your Web server's root directory.

The pbcgi60.exe file should remain in the Web.PB /bin directory.

❖ **To provide standard CGI support with Netscape Enterprise Server:**

- Copy the pbcgi60.script file from the Web.PB /bin directory to the /cgi-bin subdirectory under the Netscape Enterprise Server /ns-home root directory.

  Make sure that the user running the Web server process has execute permission on this script.

  For example, type:

```
cp /usr/pb6/pb6web/bin/pbcgi60.script
   /usr/ns-home/cgi-bin
```

Building the server application

The functional requirements for a Web.PB server application are identical to those for a normal distributed PowerBuilder server.

The server application has two main components:

- Remote objects
- Transport object

About remote objects

The basic building blocks of a server application are the remote objects it contains. Each **remote object** is a custom class (nonvisual) user object that has methods that can be invoked by the Web.PB program or by one or more PowerBuilder client applications.

A typical server application contains several nonvisual objects that package business logic into reusable components. You can create these objects from scratch or build them by inheriting characteristics from existing objects.

FOR INFO For complete instructions on creating custom class user objects, see the *PowerBuilder User's Guide*.

About the Transport object

The **Transport object** makes it possible for the server application to receive client connections and process client requests for services. The Transport object is instantiated in a script in the server application. The properties of the Transport object provide the information PowerBuilder needs to process client requests. For example, the Transport object can identify the server application and the communications driver that will be used, and specify how data will be passed across the network.

FOR INFO For complete information about the architecture of a distributed application, see the part on building distributed applications in *Application Techniques*.

WebPB=1 trace option for Transport object

You can use the WebPB=1 trace option with the Transport object to log all Web.PB activity against the server application.

FOR INFO For how to use this and other trace options for the Transport object, see the part on building distributed applications in *Application Techniques*.

Writing the user object methods

The primary task in building a server application is writing the methods for the nonvisual objects.

About arguments

The methods associated with a nonvisual object in a Web.PB server application can pass arguments that use any of the standard data types (except structures). Web.PB will pass a NULL value to the object method for any argument for which the program request did not provide a value. In your method script, you may want to use the IsNull function to determine whether the arguments have NULL values and supply your own default values instead.

About return values

The methods can return values of the following data types:

- String
- Blob

By handling both string and blob values, Web.PB lets you create dynamic content for your Web pages that includes HTML text, references to nontextual MIME types such as images and sounds, and binary data. Methods that return blob values can perform URL redirection.

How strings are processed

When Web.PB invokes an object method that returns a string, it calls the method *once* for each program request and returns a single string value. Web.PB inserts a content type of TEXT/HTML into the string value that is sent back to the client browser.

How blobs are processed

When Web.PB invokes an object method that returns a blob, it calls the method *repeatedly* until the method returns a NULL blob value (or a blob value that has a length of 0). The method must insert the appropriate content type for the data it is returning. For example, a method that returns a binary image would specify IMAGE/GIF or IMAGE/JPEG along with the binary data. In addition to specifying the content type, the method must delimit each separate portion of the data returned with the appropriate carriage-return and line-feed characters.

No function overloading

The methods you write for nonvisual objects in a Web.PB server application do *not* support function overloading.

Using DataStores

The object methods in a server application can use **DataStores** to interact with databases. DataStores are nonvisual DataWindow controls that act just like DataWindow controls except that they do not have visual attributes.

DataStores can be useful in a server application: they let you perform database processing on a remote server instead of on each client machine.

Converting DataWindow data into HTML

Some DataWindow presentation styles translate better into HTML than others. The following presentation styles produce good results:

Tabular
Group

Freeform
Crosstab
Grid

The Composite, Graph, and OLE 2.0 presentation styles and nested reports produce HTML output that is based on the result only and not on the presentation style. DataWindows that have overlapping objects may not produce the results you want.

RTE presentation style is not supported
A server application cannot contain a DataStore that has a DataWindow object that uses the RTE presentation style. Rich text processing is not supported in distributed applications.

About shopping cart applications

The methods you write for nonvisual user objects in a Web.PB server application can pass arrays as arguments. This lets you build Web.PB applications that use a shopping cart metaphor.

Shopping cart applications let users browse through a virtual store on the Web and add items to their shopping cart for possible purchase. The items in the shopping cart become the elements in the array that is passed to the user object method.

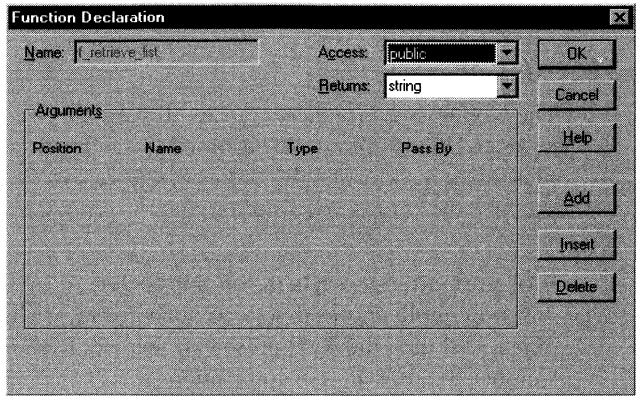
FOR INFO For an example showing how to pass an array as an argument to build a simple shopping cart application, see "Example 6: passing an array as an argument" on page 140.

Example 1: returning an HTML string

Description

This example shows the definition for the f_retrieve_list function of the nonvisual user object uo_pbdemo. The function creates an HTML string that displays a list of employees retrieved from a database. The function uses a DataStore to access the employee data. After creating an HTML header, f_retrieve_list constructs the HTML that displays the employee data by assigning the Data.HTMLTable property to a string variable.

Function declaration



Script

Here is the script for the f_retrieve_list function.

```
//Retrieve a list of employees and return the data
//in HTML format.

string ls_html
string ls_search_arg
long ll_rows_retrieved

string ls_header = "<HTML><HEAD>"
string ls_title = "<TITLE>Employee List</TITLE>"
string ls_endheader = "</HEAD>"
string ls_h1 = "<BODY><H1>Employee List</H1>"
string ls_footer = "</BODY></HTML>"

ls_html = ls_header + ls_title + ls_endheader + ls_h1

ds_names = create datastore
ds_names.dataobject="d_emplist"
ds_names.settransobject(sqlca)
ll_rows_retrieved= ds_names.retrieve()

IF ll_rows_retrieved > 0 THEN
   ls_html = ls_html +
      ds_names.object.datawindow.data.htmltable
   ls_html = ls_html + ls_footer
ELSE
   ls_html = 'No employees found for this search'
END IF
```

```
RETURN ls_html
```

HTML

On Windows Here is an HTML example that shows how to invoke the f_retrieve_list function when using Web.PB on a Windows Web server.

```
<A HREF="/scripts/pbcgi60.exe/PBWEBTEST/uo_pbdemo
/f_retrieve_list">Retrieve employee list</A>
```

On UNIX Here is an HTML example that shows how to invoke the f_retrieve_list function when using Web.PB on a UNIX Web server. You must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<A HREF="/scripts/pbcgi60.script/PBWEBTEST/uo_pbdemo
/f_retrieve_list">Retrieve employee list</A>
```

On Macintosh Here is an HTML example that shows how to invoke the f_retrieve_list function when using Web.PB on a Macintosh Web Server. You must use webpb.pbw$ as the Web.PB argument to invoke the WebPB shared library in your Web server's Plug-Ins folder.

```
<A HREF="/webpb.pbw$/PBWEBTEST/uo_pbdemo
/f_retrieve_list">Retrieve employee list</A>
```

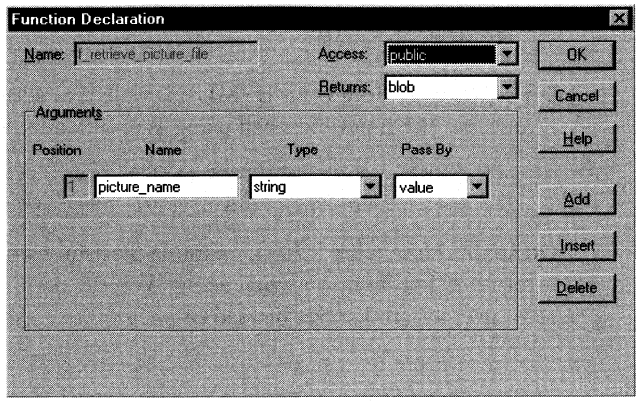
FOR INFO For a complete explanation of the HTML syntax you use to invoke remote object methods, see "Designing the HTML pages" on page 150.

Example 2: returning a picture file

Description

This example shows how to return a binary image that is stored in a file. The nonvisual user object uo_pictures has a function f_retrieve_picture_file. F_retrieve_picture_file retrieves binary data from a file. The function constructs a string that includes an HTML header and the binary data. The HTML header specifies the content type and length of the binary data.

Function declaration



Script

Here is the script for the f_retrieve_picture_file function. Because this script returns a blob, the Web.PB program will call the function repeatedly until the function returns a NULL blob value (or a blob value that has a length of 0). Each time it is executed, f_retrieve_picture_file uses the FileRead function to read the entire file or the next 32,765 bytes of the file, whichever is shorter. The final time it is executed, it returns a NULL blob value to notify Web.PB that processing has finished.

```
//Instance variables:
//Boolean TerminateProcessing
//Integer fnum

blob bitmap
//Set this to where you keep your images
string image_directory
//Don't allow the remote user to specify paths
image_directory = "c:\webpb\htdocs\"

IF TerminateProcessing = TRUE THEN
   setNull(bitmap)
   RETURN bitmap
END IF

IF fnum = 0 THEN // First time?

   long BadFileName
   //look for path specifier
   BadFileName = Pos(picture_name,"\")
   IF BadFileName = 0 THEN
```

```
//look for path specifier
BadFileName = Pos(picture_name,"/")
END IF

IF BadFileName = 0 THEN
  //look for path specifier
  BadFileName = Pos(picture_name,":")
END IF

IF NOT BadFileName = 0 THEN
  //report the error
  bitmap =
    blob("content-type: text/plain~r~n~r~n
    Illegal Path or Drive information used "
    + picture_name+ " ~r~n")
  TerminateProcessing = TRUE
  RETURN bitmap
ELSE
  picture_name = image_directory + picture_name
END IF

TerminateProcessing = FALSE

boolean file_exists
file_exists = FileExists(picture_name)
IF NOT file_exists THEN
  bitmap =
    blob("content-type: text/plain~r~n~r~n
    picture '" + picture_name + "' does not
    exist~r~n")
  TerminateProcessing = TRUE
  RETURN bitmap
END IF

long flen
// Get the file length, and open the file
flen = FileLength(picture_name)
fnum = FileOpen(picture_name, &
  StreamMode!, Read!)

IF fnum = -1 THEN
  bitmap = blob("content-type: text/plain~r~n~r~n
    picture '" + picture_name +
```

```
            "' could not be opened for read~r~n")
        TerminateProcessing = TRUE
        fnum = 0
        RETURN bitmap
    END IF

    string image_extent
    //Take the last 3 characters.
    //They should be something like "gif" or "bmp".
    image_extent = right(picture_name,3)
    // output the content type
    bitmap =
      blob("content-type: image/" + image_extent
      + "~r~n")
    // output the picture length
    bitmap += blob("content-length: ")
    bitmap += blob(string(flen))
    bitmap += blob("~r~n~r~n")
    RETURN bitmap
END IF

// Read the file
long bytes_read

bytes_read = FileRead(fnum, bitmap)

//no more data to return
IF bytes_read <= 0 THEN
    FileClose(fnum)
    //Set return value to NULL to signal end of data
    setNull(bitmap)
    fnum = 0
END IF

// output the picture data
RETURN bitmap
```

HTML

On Windows Here is an HTML example that shows how to invoke the f_retrieve_picture_file function when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.exe/PBWEB/uo_pictures
```

```
/f_retrieve_picture_file">
Enter Picture Name: <INPUT NAME="picture_name"
value="pwrslogo.gif">
<INPUT TYPE="SUBMIT" VALUE="Display picture">
</FORM>
<P>
```

On UNIX Here is an HTML example that shows how to invoke the f_retrieve_picture_file function when using Web.PB on a UNIX Web server. You must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.script/PBWEB/uo_pictures
/f_retrieve_picture_file">
Enter Picture Name: <INPUT NAME="picture_name"
value="pwrslogo.gif">
<INPUT TYPE="SUBMIT" VALUE="Display picture">
</FORM>
<P>
```

On Macintosh Here is an HTML example that shows how to invoke the f_retrieve_picture_file function when using Web.PB on a Macintosh Web Server. You must use webpb.pbw$ as the Web.PB argument to invoke the WebPB shared library in your Web server's Plug-Ins folder.

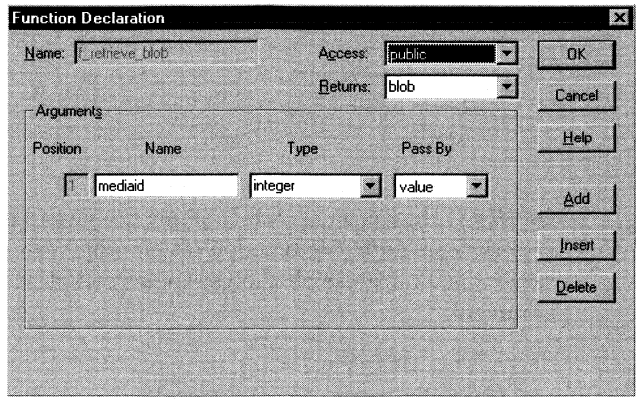
```
<FORM METHOD="GET"
ACTION="/webpb.pbw$/PBWEB/uo_pictures
/f_retrieve_picture_file">
Enter Picture Name: <INPUT NAME="picture_name"
value="pwrslogo.gif">
<INPUT TYPE="SUBMIT" VALUE="Display picture">
</FORM>
<P>
```

Example 3: returning a database blob column

Description

This example shows how to return a binary image that is stored in a database. The nonvisual user object uo_pictures has a function f_retrieve_blob that retrieves an image from the database based on a media ID value entered by the user. F_retrieve_blob uses embedded SQL statements to retrieve the blob data from the database into a local blob variable. The function then constructs a string that includes an HTML header and the blob data. The HTML header specifies the content type and length of the blob data.

Function declaration



Script

Here is the script for the f_retrieve_blob function. Note that because this script returns a blob, the Web.PB program will call the function repeatedly until the function returns a NULL blob value (or a blob value that has a length of 0). The first time it is executed, f_retrieve_blob generates the HTTP header and retrieves the data. The second time it is executed, it returns a NULL blob value to notify Web.PB that processing has finished.

```
//Instance variable:
//Boolean TerminateProcessing

blob bitmap, bmpdata
long datalength
string media_type

// First time ?
// Generate the HTTP header.
IF TerminateProcessing = FALSE THEN

   SELECTBLOB picture
   INTO :bmpdata
   FROM media
   WHERE media.id = :mediaid
   USING SQLCA;

   IF sqlca.sqlcode <> 0 THEN
     bitmap =
       blob("content-type: text/plain~r~n~r~n
       Retrieve blob failed!~r~n")
     TerminateProcessing = TRUE
```

```
    RETURN bitmap
  END IF

  SELECT medtype
  INTO :media_type
  FROM media
  WHERE media.id = :mediaid
  USING SQLCA;

  IF sqlca.sqlcode <> 0 THEN
    bitmap =
      blob("content-type: text/plain~r~n~r~n
      Retrieve type failed!~r~n")
    TerminateProcessing = TRUE
    RETURN bitmap
  END IF

  IF media_type = "jpg" THEN media_type = "jpeg"
  datalength = len (bmpdata)

  bitmap = blob("content-type: IMAGE/"
    + media_type + "~r~n")
  bitmap += blob("content-length: ")
  bitmap += blob(string(datalength))
  bitmap += blob("~r~n~r~n")
  bitmap += bmpdata
  TerminateProcessing = TRUE
  RETURN bitmap
ELSE
  SetNull (bitmap)
  RETURN bitmap
END IF
```

HTML

On Windows Here is an HTML example that shows how to invoke the f_retrieve_blob function when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.exe/PBWEB/uo_pictures
/f_retrieve_blob">
Enter Picture Number: <INPUT NAME="mediaid"
VALUE="906">
<INPUT TYPE="SUBMIT" VALUE="Display picture">
</FORM>
```

```
<P>
```

On UNIX Here is an HTML example that shows how to invoke the f_retrieve_blob function when using Web.PB on a UNIX Web server. You must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.script/PBWEB/uo_pictures
/f_retrieve_blob">
Enter Picture Number: <INPUT NAME="mediaid"
VALUE="906">
<INPUT TYPE="SUBMIT" VALUE="Display picture">
</FORM>
<P>
```

On Macintosh Here is an HTML example that shows how to invoke the f_retrieve_blob function when using Web.PB on a Macintosh Web Server. You must use webpb.pbw$ as the Web.PB argument to invoke the WebPB shared library in your Web server's Plug-Ins folder.

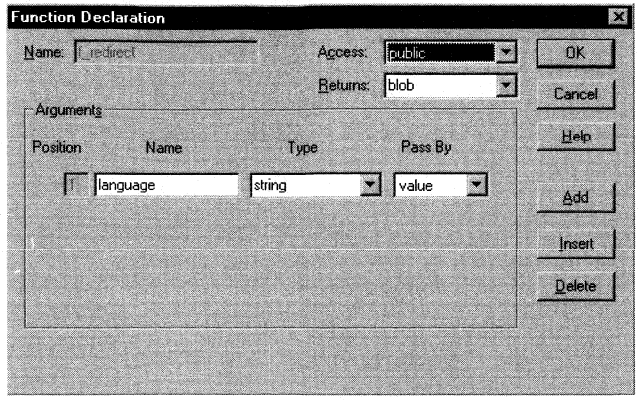
```
<FORM METHOD="GET"
ACTION="/webpb.pbw$/PBWEB/uo_pictures
/f_retrieve_blob">
Enter Picture Number: <INPUT NAME="mediaid"
VALUE="906">
<INPUT TYPE="SUBMIT" VALUE="Display picture">
</FORM>
<P>
```

Example 4: performing a URL redirection

Description

This example shows how to redirect the user to a particular URL based on conditions you set. The nonvisual user object uo_webtest has a function f_redirect. F_redirect routes the user to one of several Sybase Web pages, depending on which language the user specifies in a form entry field.

Function declaration



Script

Here is the script for the f_redirect function. To redirect the browser to another Web location, f_redirect constructs an HTTP header that uses the LOCATION: *url* syntax to specify the destination. The function then returns this string as a blob value.

```
//Instance variable:
//long callcount

String rc
blob rb

IF callcount = 0 THEN

   CHOOSE CASE language
     CASE "English"
       rc = "location: "
          + "http://www.sybase.com/" + "~r~n~r~n"
       rb = blob(rc)
       callcount = callcount + 1

     CASE "Japanese"
       rc = "location: "
          + "http://www.sybase.co.jp/" + "~r~n~r~n"
       rb = blob(rc)
       callcount = callcount + 1

     CASE "Norwegian"
       rc = "location: "
          + "http://www.sybase.no/" + "~r~n~r~n"
```

```
            rb = blob(rc)
            callcount = callcount + 1

        CASE ELSE
            rc = "content-type: text/plain~r~n~r~n"
               + language + " is not supported!~r~n"
            rb = blob(rc)
            callcount = callcount + 1

        END CHOOSE
    ELSE
       SetNull(rb)
    END IF

    RETURN rb
```

HTML

On Windows Here is an HTML example that shows how to invoke the f_redirect function when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.exe/PBWEB/uo_webtest
/f_redirect">
My language is: <INPUT NAME="language"
VALUE="English">
<INPUT TYPE="SUBMIT" VALUE="Please tell me more about
Sybase!">
</FORM>
```

On UNIX Here is an HTML example that shows how to invoke the f_redirect function when using Web.PB on a UNIX Web server. You must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.script/PBWEB/uo_webtest
/f_redirect">
My language is: <INPUT NAME="language"
VALUE="English">
<INPUT TYPE="SUBMIT" VALUE="Please tell me more about
Sybase!">
</FORM>
```

On Macintosh Here is an HTML example that shows how to invoke the f_redirect function when using Web.PB on a Macintosh Web Server. You must use webpb.pbw$ as the Web.PB argument to invoke the WebPB shared library in your Web server's Plug-Ins folder.

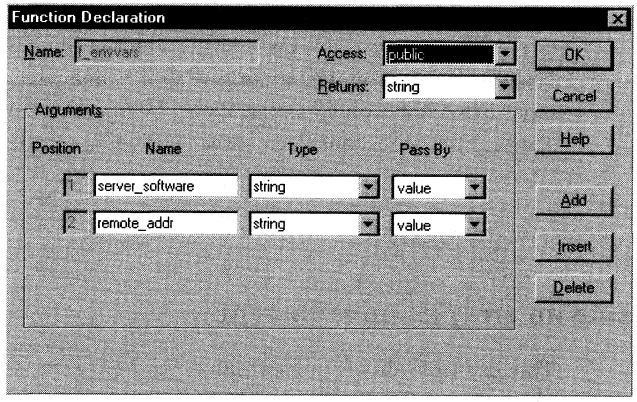
```
<FORM METHOD="GET"
ACTION="/webpb.pbw$/PBWEB/uo_webtest/f_redirect">
My language is: <INPUT NAME="language"
VALUE="English">
<INPUT TYPE="SUBMIT" VALUE="Please tell me more about
Sybase!">
</FORM>
```

Example 5: accessing the server environment variables

Description

To access the server environment variables, you need to create methods that take arguments *with the same names as the server variables*. This example shows how to do this. The nonvisual user object uo_webtest has a function f_envvars. To retrieve the name of the server software and the IP address of the remote host making the request, f_envvars takes SERVER_SOFTWARE and REMOTE_ADDR as its arguments.

Function declaration



Script

Here is the script for the f_envvars function.

```
string rc

rc = "<!DOCTYPE HTML PUBLIC ~"-//IETF//DTD HTML
3.0//EN~" ~"html.dtd~">"
rc += "<HTML>"
rc += "<HEAD>"
rc += "<TITLE> f_envvars test </TITLE>"
rc += "</HEAD>"
rc += "<BODY>"
```

```
rc += "<br>SERVER_SOFTWARE = " + SERVER_SOFTWARE
rc += "<br>REMOTE_ADDR = " + REMOTE_ADDR

rc += "</BODY>"
rc += "</HTML>"

RETURN rc
```

HTML

On Windows Here is an HTML example that shows how to invoke the f_envvars function when you use Web.PB on a Windows Web server.

```
<A HREF="/scripts/pbcgi60.exe/SERVER01/uo_webtest
/f_envvars">Return CGI Environment Variables</A>
```

On UNIX Here is an HTML example that shows how to invoke the f_envvars function when you use Web.PB on a UNIX Web server. You must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<A HREF="/scripts/pbcgi60.script/SERVER01/uo_webtest
/f_envvars">Return CGI Environment Variables</A>
```

On Macintosh Here is an HTML example that shows how to invoke the f_envvars function when using Web.PB on a Macintosh Web Server. You must use webpb.pbw$ as the Web.PB argument to invoke the WebPB shared library in your Web server's Plug-Ins folder.

```
<A HREF="/webpb.pbw$/SERVER01/uo_webtest
/f_envvars">Return CGI Environment Variables</A>
```

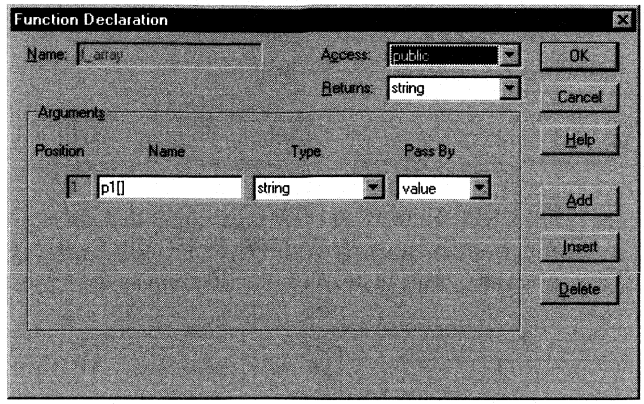
Example 6: passing an array as an argument

Description

This example shows the definition for the f_array function of a nonvisual user object uo_webtest. The declaration of f_array shows that it takes an array named p1 as an argument and returns a string.

The f_array function is used in a simple shopping cart application where a user can select multiple items from a list and add them to a shopping cart for possible purchase. The items in the shopping cart become the elements in the array passed to the f_array function.

Function declaration



Script

Here is the script for the f_array function. After determining the size of the array (p1) and moving through the array elements one at a time, the script returns an HTML string that lists the items selected for the shopping cart.

```
long ll_size
string rc = "<HTML> "
integer li_index

ll_size = UpperBound(p1)

rc += "You selected: "

for li_index = 1 to ll_size
   rc += "<br>" + p1[li_index]

next

rc += "</html>"
return rc
```

HTML

On Windows Here is an HTML example that shows how to invoke the f_array function when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="/cgi-shl/pbcgi60.exe/SERVER01/uo_webtest
/f_array">
<SELECT Name="p1" SIZE=3 ALIGN=top Multiple>
<OPTION>Tee Shirt
<OPTION>Baseball Cap
<OPTION>Visor
```

```
<OPTION>Sweatshirt
<OPTION>Shorts
</SELECT>
<P>
<INPUT TYPE="SUBMIT" VALUE="Add to shopping cart">
</FORM>
```

On UNIX Here is an HTML example that shows how to invoke the f_array function when using Web.PB on a UNIX Web server. You must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<FORM METHOD="GET"
ACTION="/cgi-shl/pbcgi60.script/SERVER01/uo_webtest
/f_array">
<SELECT Name="p1" SIZE=3 ALIGN=top Multiple>
<OPTION>Tee Shirt
<OPTION>Baseball Cap
<OPTION>Visor
<OPTION>Sweatshirt
<OPTION>Shorts
</SELECT>
<P>
<INPUT TYPE="SUBMIT" VALUE="Add to shopping cart">
</FORM>
```

On Macintosh Here is an HTML example that shows how to invoke the f_array function when using Web.PB on a Macintosh Web Server. You must use webpb.pbw$ as the Web.PB argument to invoke the WebPB shared library in your Web server's Plug-Ins folder.

```
<FORM METHOD="GET"
ACTION="/webpb.pbw$/SERVER01/uo_webtest/f_array">
<SELECT Name="p1" SIZE=3 ALIGN=top Multiple>
<OPTION>Tee Shirt
<OPTION>Baseball Cap
<OPTION>Visor
<OPTION>Sweatshirt
<OPTION>Shorts
</SELECT>
<P>
<INPUT TYPE="SUBMIT" VALUE="Add to shopping cart">
</FORM>
```

Editing the Web.PB initialization file

What is the Web.PB initialization file?

To establish a connection to the distributed PowerBuilder server application, Web.PB uses connection properties that are maintained in the Web.PB initialization file. When an HTML document makes a program request, Web.PB reads the Web.PB initialization file to determine how to establish the connection.

The Web.PB initialization file specifies various connection parameters that the Web.PB program can use, including the server location (or IP address), the service name (or port number) for the application, and the communications driver to use for the connection.

Name and location on different platforms

The name and location of the Web.PB initialization file depend on the operating system platform on the Web server machine, as follows:

Platform	Filename	Location
Windows	PBWEB.INI	Main Windows directory on Web server machine
UNIX	.pbweb.ini (hidden file)	Web.PB installation directory on Web server machine
Power Macintosh	WebPB Preferences	System Folder:Preferences

Sample Web.PB initialization file

The content of the Web.PB initialization file is similar on all platforms. The only difference is that on each platform, the [Web.PB] section of the file contains server environment variable keywords for the program interfaces (CGI, ISAPI, NSAPI, or WSAPI) supported on that platform.

Here is a sample Web.PB initialization file on Windows.

```
[Web.PB]
errormessage='<br>E-Mail: <A HREF=
"mailto:YOURwebmaster@YOURcompany.com"> Let me know
about it</A>'
ISAPIKeywords="SERVER_SOFTWARE,SERVER_NAME,
GATEWAY_INTERFACE,SERVER_PROTOCOL,SERVER_PORT,
REQUEST_METHOD,HTTP_ACCEPT,PATH_INFO,PATH_TRANSLATED,
SCRIPT_NAME,QUERY_STRING,REMOTE_HOST,REMOTE_ADDR,
REMOTE_USER,CONTENT_TYPE,CONTENT_LENGTH,
HTTP_USER_AGENT"
CGIKeywords="SERVER_SOFTWARE,SERVER_NAME,
GATEWAY_INTERFACE,SERVER_PROTOCOL,SERVER_PORT,
REQUEST_METHOD,HTTP_ACCEPT,PATH_INFO,PATH_TRANSLATED,
```

```
SCRIPT_NAME,QUERY_STRING,REMOTE_HOST,REMOTE_ADDR,
REMOTE_USER,CONTENT_TYPE,CONTENT_LENGTH,
HTTP_USER_AGENT"
NSAPIKeywords="SERVER_SOFTWARE,SERVER_NAME,
GATEWAY_INTERFACE,SERVER_PROTOCOL,SERVER_PORT,
REQUEST_METHOD,HTTP_ACCEPT,PATH_INFO,PATH_TRANSLATED,
SCRIPT_NAME,QUERY_STRING,REMOTE_HOST,REMOTE_ADDR,
REMOTE_USER,CONTENT_TYPE,CONTENT_LENGTH,
HTTP_USER_AGENT"

[Default]
serveralias=idkdemo
serverobject=u_web_examples

[idkdemo]
application=pb_net_examples
location=localhost
driver=winsock

[PowerBank]
application=pbanksvr
location=bank_01
driver=winsock
options=rawdata=1
```

Sections

The Web.PB initialization file has three kinds of sections:

- [Web.PB] section
- [Default] section
- Server section

Web.PB section

Contents

The [Web.PB] section of the Web.PB initialization file contains keys that specify system-wide settings for Web.PB.

ErrorMessage key

The [Web.PB] section contains a key called ErrorMessage that provides a way for users to send mail to their site administrator when an error occurs during processing. Whenever an error occurs, Web.PB passes the HTML specified in the ErrorMessage key to the browser. The HTML defines a link that allows the user to invoke the browser's mail facility.

Environment variable keywords

The [Web.PB] section also contains definitions for the CGI, ISAPI, NSAPI, and WSAPI keywords required for Web.PB to access the server environment variables. (CGI is supported on Windows and UNIX; ISAPI and NSAPI are supported on Windows; WSAPI is supported on Macintosh.) The server environment variables provide a way for user object methods to obtain information about the Web server configuration, the browser, the user who makes a program request, and the request method (GET or POST).

To access the server environment variables, you need to create methods that take arguments *with the same names as the server variables*. For example, to generate HTML that returns the IP address of the remote host making a request, you would need to define a method that takes REMOTE_ADDR as one of its arguments.

Editing

You may need to edit the ErrorMessage key in the [Web.PB] section. For example, you might want to change the mail address or the text for the link that invokes the mail facility.

```
errormessage='<br>E-Mail: <A
HREF="mailto:joea@widgets.com"> Please send me a
message</A>'
```

In most cases, you should *not* need to edit the environment variable keywords in the [Web.PB] section. However, if you want to prevent server applications from accessing one or more server environment variables, you can remove these keywords from the [Web.PB] section.

Default section

Contents

If a program request does not specify a server alias or server object, Web.PB uses the defaults specified in the [Default] section of the Web.PB initialization file to determine which server alias and object to use.

Keys

The [Default] section has the following keys:

Key	Description
Serveralias	The name of the server alias you want Web.PB to use when a URL reference does not specify a server alias The server alias specified must have a separate server section in the Web.PB initialization file that identifies the application, driver, and location for the server application
Serverobject	The name of the server object you want Web.PB to use when a URL reference does not specify a user object

Server section

Contents

Each distributed server application that will be accessed from a Web server must have a separate section in the Web.PB initialization file. The server section specifies the connection parameters the Web.PB program uses to connect to the server application. These parameters include the server location (or IP address), the service name (or port number) for the application, and the communications driver to use for the connection.

Section name

The name of each server section is an alias that you give to the server application. The server alias provides a way for the program request to refer to the server application without having to specify the required connection parameters.

Communications drivers

The values you supply for keys in each server section depend on the communications driver you are using with Web.PB. Web.PB can work with different drivers depending on the operating system running on the Web server machine where Web.PB is installed:

Driver	Provides communication through	Platforms
WinSock	Sockets facility for a TCP/IP network	Windows UNIX Power Macintosh
NamedPipes	Named Pipes facility	Windows

Keys

The server section has the following keys:

Connection property	Data type	Description
Application	String	Specifies the server application that a client application wants to connect to **For the WinSock driver** Specifies either of the following: ♦ The port number for the application (for example, 10099). Each server application requires a unique port number on the server machine. If you specify a port number, select a number that is greater than 4096 ♦ The service name for the application. The service name is an indirect reference to the application's port number. The mapping of the service name to the port number is specified in the TCP/IP services file **For the NamedPipes driver** Specifies the application portion of the pipe name. The combination of the Location and Application values forms the pipe name. The pipe name is constructed as follows: *location*\PIPE*application* The Application must be unique for the Location you specify
Driver	String	The communications driver that will be used for the connection. Values are: WinSock NamedPipes

Connection property	Data type	Description
Location	String	Specifies the location of the server application **For the WinSock driver** Specifies one of the following values: ♦ The host name for the server (as specified in the TCP/IP hosts file) ♦ The IP address (for example, 199.99.99.91) ♦ LocalHost (which tells PowerBuilder that the server application resides on the local machine) **For the NamedPipes driver** Specifies the location portion of the pipe name. The combination of the Location and Application values forms the pipe name. The pipe name is constructed as follows: *\\location*\PIPE*application* If no location is specified, a local pipe name is constructed using a dot (.) in the machine name portion.
Options	String	Specifies one or more communications options. If you specify more than one option, you need to separate the options with commas FOR INFO For a description of each of the options, see "Options" next

Options

The Options key in the server section can specify one or more of the following communications options for particular drivers:

Option	Description
BufSize=*n*	Sets the connection buffer size to the value specified
MaxRetry=*n*	Specifies how many times the client application will try to connect when the server application's listening port is busy Applies to the WinSock driver only
NoDelay=1	Specifies that each packet be sent without delay. Corresponds to the TCP_NODELAY option Setting this option may degrade performance significantly. Do not use this option unless you thoroughly understand its implications Applies to the WinSock driver only

Option	Description
RawData=1	Specifies that raw data be passed over the network. By default, the WinSock driver obscures the data that is passed over the network. Setting this option to 1 overrides the default behavior. Both the client and server applications must have the same setting. If there is a discrepancy between the client and server settings, the communication fails Setting this option to 1 may improve performance slightly Applies to the WinSock driver only
TimeOut=*n*	Specifies the amount of time PowerBuilder will wait for a reply from a synchronous request. After the specified time elapses, an error is returned

Designing the HTML pages

A Web page can specify a reference to a method of a remote object in a server application by using either of two HTML elements:

- Form element (<FORM>)
- Anchor element (<A>)

You can create the HTML syntax required to invoke a remote object method in two ways:

- Typing the HTML statements in a text editor
- Using a commercially available HTML editor

Using the Form element to reference an object

The Form element allows you to create a data input form. You can use the Form element to reference a remote object method, passing one or more values specified by the user as arguments to the method.

Syntax

To specify a reference to a remote object method in the Form element, use this syntax:

```
<FORM METHOD="GET/POST"
ACTION="URLpath/Web.PB/serveralias/object/method>
text<INPUT NAME="parametername" TYPE="type"
VALUE="parametervalue">
<INPUT TYPE="SUBMIT" VALUE="buttontext">
</FORM>
```

Argument	Description
URLpath	Specifies the URL path that maps to the directory where CGI programs reside (for example, /SCRIPTS/ or /CGI-BIN/)

Argument	Description
Web.PB	**Standard CGI (on Windows)** PBCGI60.EXE **Standard CGI (on UNIX)** pbcgi60.script **ISAPI (on Windows only)** PBISA60.DLL **NSAPI (on Windows only)** The name of a file on the server machine that has the extension PBW (such as WEBAGENT.PBW). When you specify a URL reference to this file, the server invokes either PBNS160.DLL or PBNS260.DLL to service the request **WSAPI (on Macintosh only)** Use webpb.pbw$ to invoke the WebPB shared library in your Web server's Plug-Ins folder
serveralias (optional)	The name of the server alias that you want Web.PB to use. The server alias specified must have a separate server section in the Web.PB initialization file that identifies the application, driver, and location for the server application If you do not specify *serveralias*, Web.PB uses the server alias specified in the [Default] section of the Web.PB initialization file
object (optional)	The name of the remote object in the server application If you do not specify *object*, Web.PB uses the object specified in the [Default] section of the Web.PB initialization file
method	The name of the remote object method to invoke
text	Text that prompts the user to enter a value
parametername	The name of an argument to the remote object method. The value you specify must match the name given to the argument in the method declaration
type	The type of control. Values are: ♦ CHECKBOX ♦ HIDDEN ♦ PASSWORD ♦ RADIO ♦ TEXT (Default)
parametervalue	A default value for the parameter. This value is used if the user does not specify a value in the input field
buttontext	Text that appears on the button

Usage

You can specify multiple <INPUT> elements in any order as long as the parameter names match the names given to the corresponding arguments in the method declaration. When the user inputs a value in the input field, Web.PB reads the data as an ASCII string and then converts it to the data type of the target argument.

You can use either GET or POST as the submission method for a form. When you use GET, the argument values are appended to the URL path sent to the server. When you use POST, the argument values are added to the message body. Use the POST method for forms that send large quantities of data to the server. Use the GET method if you want the user to be able to create a bookmark to call the remote object method with the argument values entered in the form.

Examples

Using CGI with a server alias

This example shows how you might use the Form element to invoke the f_name_search method of the uo_pbdemo user object when using Web.PB on a Windows Web server. The form references the PBWEBTEST server alias in the Web.PB initialization file and uses standard CGI to make the request.

CGI is supported when using Web.PB on Windows and UNIX, but not on Macintosh.

On Windows Here is the HTML code for invoking f_name_search when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="/cgi-bin/pbcgi60.exe/PBWEBTEST/uo_pbdemo
/f_name_search">
Type a letter: <INPUT NAME="search_letter" VALUE="">
<INPUT TYPE="SUBMIT" VALUE="Search by name">
</FORM>
```

On UNIX Here is the HTML code for invoking f_name_search when using Web.PB on a UNIX Web server. The only difference from the HTML code on Windows is that on UNIX, you must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<FORM METHOD="GET"
ACTION="/cgi-bin/pbcgi60.script/PBWEBTEST
/uo_pbdemo/f_name_search">
Type a letter: <INPUT NAME="search_letter" VALUE="">
```

```
<INPUT TYPE="SUBMIT" VALUE="Search by name">
</FORM>
```

Using CGI without a server alias

This example shows how you might use the Form element to invoke the f_string method. The method call does not specify a server alias or an object. Instead, it uses the server alias and object specified in the [Default] section of the Web.PB initialization file. The request uses standard CGI.

CGI is supported when using Web.PB on Windows and UNIX, but not on Macintosh.

On Windows Here is the HTML code for invoking f_string when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.exe/f_string">
Enter value: <INPUT NAME="p1" VALUE="hello internet">
<INPUT TYPE="SUBMIT" VALUE="Display string">
</FORM>
```

On UNIX Here is the HTML code for invoking f_string when using Web.PB on a UNIX Web server. The only difference from the HTML code on Windows is that on UNIX, you must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<FORM METHOD="GET"
ACTION="/scripts/pbcgi60.script/f_string">
Enter value: <INPUT NAME="p1" VALUE="hello internet">
<INPUT TYPE="SUBMIT" VALUE="Display string">
</FORM>
```

Using NSAPI without a server alias

This example uses NSAPI to invoke the f_get_product method of the uo_webtest object. The method call does not specify a server alias. Instead, it uses the server alias specified in the [Default] section of the Web.PB initialization file.

NSAPI is supported when using Web.PB on Windows, but not on UNIX or Macintosh.

On Windows Here is the HTML code for invoking f_get_product when using Web.PB on a Windows Web server.

```
<FORM METHOD="GET"
ACTION="webagent.pbw/uo_webtest/f_get_product">
Enter value: <INPUT NAME="id">
</FORM>
<INPUT TYPE="SUBMIT" VALUE="Display data">
```

Using WSAPI with a server alias

This example shows how you might use the Form element to invoke the f_char method of the uo_webtest user object when using Web.PB on a Macintosh Web server. The form references the PBWEBTEST server alias in the Web.PB initialization file and uses the WebSTAR API (WSAPI) to make the request.

WSAPI is supported when using Web.PB on Macintosh, but not on Windows or UNIX.

On Macintosh Here is the HTML code for invoking f_string when using Web.PB on a Macintosh Web server.

```
<FORM METHOD="GET"
ACTION="/webpb.pbw$/PBWEBTEST/uo_webtest/f_char">
Enter value: <INPUT NAME="p1">
<INPUT TYPE="SUBMIT" VALUE="Display character">
</FORM>
```

Using the Anchor element to reference an object

The Anchor element allows you to mark text that will be used as a hypertext link. You can use the Anchor element to reference a remote object method, passing one or more values as arguments to the method. If you pass arguments in an Anchor element, the arguments must be hardcoded in the HTML.

Syntax

To specify a reference to a remote object method in the Anchor element, use this syntax:

<A HREF="*URLpath*/*Web.PB*/*serveralias*/*object*/*method*?*argumentvalues*">*text*</A>

Argument	Description
URLpath	Specifies the URL path that maps to the directory where CGI programs reside (for example, /SCRIPTS/ or /CGI-BIN/)

Argument	Description
Web.PB	**For standard CGI (on Windows)** PBCGI60.EXE **For standard CGI (on UNIX)** pbcgi60.script **For ISAPI (on Windows only)** PBISA60.DLL **For NSAPI (on Windows only)** The name of a file on the server machine that has the extension PBW (such as WEBAGENT.PBW). When you specify a URL reference to this file, the server invokes either PBNS160.DLL or PBNS260.DLL to service the request **WSAPI (on Macintosh only)** Use webpb.pbw$ to invoke the WebPB shared library in your Web server's Plug-Ins folder
serveralias (optional)	The name of the server alias that you want Web.PB to use. The server alias specified must have a separate server section in the Web.PB initialization file that identifies the application, driver, and location for the server application If you do not specify *serveralias*, Web.PB uses the server alias specified in the [Default] section of the Web.PB initialization file
object (optional)	The name of the remote object in the server application If you do not specify *object*, Web.PB uses the object specified in the [Default] section of the Web.PB initialization file
method	The name of the remote object method to invoke If *method* takes one or more *argumentvalues*, follow *method* with a question mark (?) to separate it from its *argumentvalues*
argumentvalues	One or more argument values separated by ampersands (&). The argument values must be hardcoded in the HTML
text	Text that the user can click on to invoke the method

You can specify multiple argument values, but the order of the values needs to match the order of arguments in the method declaration. Web.PB reads each argument value as an ASCII string and then converts it to the data type of the target argument.

Examples

Using CGI without a server alias

This example shows how you might use the Anchor element to invoke the f_crosstab method of the uo_pbdemo user object. The method call does not specify a server alias or an object but instead uses the server alias and object specified in the [Default] section of the Web.PB initialization file. The request uses standard CGI.

CGI is supported when using Web.PB on Windows and UNIX, but not on Macintosh.

On Windows Here is the HTML code for invoking f_crosstab when using Web.PB on a Windows Web server.

```
<A HREF="/cgi-shl/pbcgi60.exe/f_crosstab?">Retrieve
sales data for printers</A>
```

On UNIX Here is the HTML code for invoking f_crosstab when using Web.PB on a UNIX Web server. The only difference from the HTML code on Windows is that on UNIX, you must use pbcgi60.script instead of pbcgi60.exe as the Web.PB argument.

```
<A HREF="/cgi-shl/pbcgi60.script/f_crosstab?">Retrieve
sales data for printers</A>
```

Using ISAPI with a server alias

This example shows how you might use the Anchor element to invoke the f_retrieve_cust_list method of the uo_pbdemo user object. The anchor references the PBWEBTEST server alias in the Web.PB initialization file and uses ISAPI to make the request.

ISAPI is supported when using Web.PB on Windows, but not on UNIX or Macintosh.

On Windows Here is the HTML code for invoking f_retrieve_cust_list when using Web.PB on a Windows Web server.

```
<A HREF="/scripts/pbisa60.dll/PBWEBTEST/uo_pbdemo
/f_retrieve_cust_list">Retrieve customer list</A>
```

Using NSAPI with a server alias

This example shows how you might use the Anchor element to invoke the f_emp_query method of the uo_pbdemo user object. The Anchor element passes two parameters to the method; the first specifies a department ID and the second specifies a city where employees work. The anchor references the PBWEBTEST serveralias in the Web.PB initialization file and uses NSAPI to make the request.

NSAPI is supported when using Web.PB on Windows, but not on UNIX or Macintosh.

On Windows Here is the HTML code for invoking f_emp_query when using Web.PB on a Windows Web server.

```
<A HREF="webagent.pbw/PBWEBTEST/uo_pbdemo
/f_emp_query?200&Boston">Retrieve customer list</A>
```

Using WSAPI with a server alias

This example shows how you might use the Anchor element to invoke the f_redirloop method of the uo_webtest user object when using Web.PB on a Macintosh Web server. The form references the PBWEBTEST server alias in the Web.PB initialization file and uses the WebSTAR API (WSAPI) to make the request.

WSAPI is supported when using Web.PB on Macintosh, but not on Windows or UNIX.

On Macintosh Here is the HTML code for invoking f_redirloop when using Web.PB on a Macintosh Web server.

```
<A HREF="/webpb.pbw$/PBWEBTEST/uo_webtest
/f_redirloop?">Start forever redirection loop</A>
```

Editing the HOSTS and SERVICES files (TCP/IP only)

On the Web server machine

When an HTML document makes a program request, Web.PB reads the Web.PB initialization file to determine how to establish a connection to the server application. If you use the WinSock driver to communicate through TCP/IP, Web.PB may also read the HOSTS and SERVICES files as well.

If the Location key in the Web.PB initialization file specifies a host name for the server machine, Web.PB checks the HOSTS file to get the IP address for the host. If the Application key in the Web.PB initialization file specifies a service name, Web.PB also checks the SERVICES file to get the port number for the server application. Depending on how you have set up your Web.PB initialization file, you may need to edit these files on the Web server machine where Web.PB resides.

On the application server machine

If the distributed server application specifies a service name in the Application property of its Transport object, PowerBuilder checks the SERVICES file to get the port number for the server application. In this case you need to edit the SERVICES file on the machine where the server application resides.

Editing the HOSTS file

For each Location key in the Web.PB initialization file that specifies a host name for the server machine, you need to add an entry in the HOSTS file on the Web server machine. For example, if you define a Location key that specifies server01 as the host name, you need to add an entry similar to the following to identify server01.

```
128.66.12.1     server01
```

Editing the SERVICES file

For each Application key in the Web.PB initialization file that specifies a service name for the server application, you need to add an entry in the SERVICES file on the Web server machine. For example, if you define an Application key that specifies dpbserv as the service name, you need to add an entry similar to the following to identify dpbserv.

```
dpbserv     10015/tcp
```

If the distributed server application specifies a service name in the Application property of its Transport object, you also to need add an entry to the SERVICES file on the application server machine to ensure that Web.PB and the server application are communicating through the same port number.

Running the application

Once you've performed the steps required to configure a Web.PB application, you can run the application.

❖ **To run a Web.PB application:**

1 Start the Web server software.

2 Start the distributed PowerBuilder server application and begin listening for client connections.

3 Start the Web browser and open the document that provides the HTML to invoke the remote object method.

4 Invoke the remote object method.

FOR INFO For complete information about designing and deploying a PowerBuilder application, see *Application Techniques*.

CHAPTER 5

Using the Web.PB Class Library

About this chapter

This chapter explains how to use the objects in the Web.PB class library.

Contents

About the examples

The code examples in this chapter show high-level concepts related to class library usage. They do not contain detailed error checking.

Overview of the Web.PB class library

Custom class user objects

The Internet Tools include the Web.PB class library. This library contains custom class user objects that enable you to:

- Save time
- Minimize HTML coding errors
- Implement a consistent look and feel
- Develop powerful distributed applications

Learn HTML

Just as you must learn PowerBuilder to get the most out of a PowerBuilder class library, you must learn HTML before you can get the most out of the Web.PB class library. The Web.PB class library produces basic HTML as well as extensions implemented by Netscape 3.0 and Internet Explorer 3.0 and above.

Two types of objects

The Web.PB class library includes:

- **HTML generation objects** These objects contain functions that you can use to generate HTML syntax:

  u_html_form
  u_html_format
  u_html_template

- **Session management objects** These objects contain functions that you can use to control session and transaction persistence across a distributed application:

  u_session
  u_transaction

What you do

You typically define these objects as instance variables in the user objects that run on your application server. In some cases you may want to define standalone descendants, as explained in "Using customized u_html_form descendants" on page 164.

Creating HTML forms

U_html_form functions

The best way to establish an interactive Web.PB session is through the use of HTML forms. The **u_html_form** custom class user object contains functions that you can call from your user object to generate elements in an HTML form.

These are the u_html_form functions you call to create selected HTML form items:

To create this item	Call this function
Form element and ACTION attribute	f_BeginForm
/Form element	f_EndForm
Singleline edit	f_MakeSLE
Multiline edit	f_MakeMLE
Hidden field	f_MakeHidden
Radio button	f_MakeRadio
Checkbox	f_MakeCheckbox
Listbox	f_MakeLB
Dropdown listbox	f_MakeDDLB
Submit button	f_MakeSubmit
Reset button	f_MakeReset

U_html_form also provides the f_RedirectClient function, which allows you to return the user to a predefined page.

FOR INFO For more on redirection, see "Programming with u_html_format" on page 172.

Implementing u_html_form

When using u_html_form to create HTML syntax, you can implement it either as a service object or by creating customized u_html_form descendants.

Using u_html_form as a service object

When using u_html_form as a service object, you associate it with the user object containing the functions called by Web.PB. This enables your user object to call u_html_form functions.

❖ **To use u_html_form as a service object:**

1 In your user object, declare an instance variable of type u_html_form:

```
u_html_form  inv_html_form
```

This user object uses PowerBuilder's autoinstantiate feature, so there is no need to code a CREATE statement.

2 Code user object functions that call u_html_form functions to create HTML syntax.

For example, a function might call the u_html_form f_MakeSLE function to create an entry field for the user ID:

```
...
ls_html += inv_html_form.f_MakeSLE &
   ("as_userid", 30, 30)
...
```

Using customized u_html_form descendants

When you use customized u_html_form descendants, you create additional functions in the descendant to display and process the form. This enables you to create a structured application in which each object handles one Web page.

❖ **To use u_html_form as a customized descendant:**

1 Create a descendant of u_html_form. This procedure creates a u_displaylogon descendant to display and process a logon form.

2 Create two additional functions in the descendant: one to display the form and one to process the form upon submission. This procedure creates an f_CreateForm function and an f_ProcessForm function.

3 Create an HTML page that calls the descendant's f_CreateForm function, passing arguments as necessary:

```
<FORM METHOD="get"
   ACTION="/cgi-bin/pbcgi60.exe/f_CreateForm">
<INPUT TYPE="Submit" VALUE="Go to Logon">
</FORM>
```

4 Code the f_CreateForm function, calling u_html_form ancestor functions as necessary to create HTML syntax. This example displays a logon form. Note that the f_MakeSLE functions create fields that contain the arguments to the f_ProcessForm function, specified in the f_BeginForm function's first argument:

```
String   ls_html = "<HTML>"

ls_html += this.f_BeginForm &
   ("/cgi-bin/pbcgi60.exe/myapp/" &
      + "u_displaylogon/f_ProcessForm", 1)
ls_html += &
   "<P>Type your ID and password. Then click OK."
ls_html += "<P>User ID: "
ls_html += this.f_MakeSLE("as_userid", 30, 30)
ls_html += "<P>Password: "
ls_html += this.f_MakeSLE &
   ("as_userpass", 30, 30, "", TRUE)
ls_html += "<P>"
ls_html += this.f_MakeSubmit("OK")
ls_html += this.f_EndForm()
ls_html += "</BODY></HTML>"
RETURN ls_html
```

5 Code the f_ProcessForm function, responding to the passed arguments and calling u_html_form ancestor functions as necessary to create HTML syntax. This example, which assumes an ib_firsttime boolean instance variable, processes a logon form and displays a selection form:

```
Blob     lblb_return

IF ib_firsttime THEN
   ib_firsttime = FALSE
   // Add logic here to validate ID and password.
   ...
   // U_displayinfo is another customized
   // u_html_form descendant.
   RETURN this.f_RedirectClient &
      ("/cgi-bin/pbcgi60.exe/myapp/" + &
         "u_displayinfo/f_CreateForm")
ELSE
   SetNull(lblb_return)
   RETURN(lblb_return)
END IF
```

6 Continue creating descendent objects to handle HTML forms. For example, the previous step specifies the f_CreateForm function in the u_displayinfo user object.

Programming with u_html_form

Call u_html_form functions to create the HTML form syntax. You typically create this syntax in either of two ways:

- Building the syntax as you call u_html_form functions
- Returning a concatenated value in the function's return statement

Building syntax as you go

One way to create HTML syntax is by creating a string variable and building HTML from u_html_form functions.

❖ **To build HTML syntax as u_html_form functions are called:**

1 Declare variables in your user object function, including a string variable to contain the HTML syntax:

```
String  ls_html = "<HTML>"
String  ls_errortext = "<P>Retrieval Error"
DataStore  lds_data
Integer  li_return
Boolean  lb_checked
```

2 Access data. This example retrieves information for a specified employee ID:

```
lds_data = CREATE DataStore
lds_data.DataObject = "d_empbyid"
lds_data.SetTransObject(SQLCA)
li_return = lds_data.Retrieve(Long(as_empid))
IF li_return < 1 THEN
   RETURN ls_errortext + &
      "<P>Return code: " + String(li_return)
END IF
```

3 Create the HTML, calling u_html_form functions as needed. This example appends to the ls_html local variable:

```
ls_html += "<BODY><H1>Employee Information</H1>"
ls_html += inv_html_form.f_BeginForm &
   ("/cgi-bin/pbcgi60.exe/myapp/" + &
      "uo_webtest/f_EmpInfo", 0)
// Set up employee information.
// They can input a new emp ID, so this field
// is an SLE.
ls_html += "<P>Employee ID: "
ls_html += &
   inv_html_form.f_MakeSLE("as_empid", 5, 5, &
```

```
      String(lds_data.Object.emp_id[1]), FALSE)
ls_html += "<P>Department ID: "
ls_html += String(lds_data.Object.dept_id[1])
ls_html += "<P>Last Name: "
ls_html += lds_data.Object.emp_lname[1]
ls_html += "<P>First Name: "
ls_html += lds_data.Object.emp_fname[1]
// Set up RadioButtons for status.
ls_html += "<P>Status:<P>"
IF lds_data.Object.status[1] = "A" THEN
   lb_checked = TRUE
ELSE
   lb_checked = FALSE
END IF
ls_html += inv_html_form.f_MakeRadio &
   ("as_status", "Active", lb_checked)
ls_html += "Active"
IF lds_data.Object.status[1] = "L" THEN
   lb_checked = TRUE
ELSE
   lb_checked = FALSE
END IF
ls_html += "<P>"
ls_html += inv_html_form.f_MakeRadio &
   ("as_status", "On Leave", lb_checked)
ls_html += "On Leave"
IF lds_data.Object.status[1] = "T" THEN
   lb_checked = TRUE
ELSE
   lb_checked = FALSE
END IF
ls_html += "<P>"
ls_html += inv_html_form.f_MakeRadio &
   ("as_status", "Terminated", lb_checked)
ls_html += "Terminated"
ls_html += "<P>Type a new Emp ID and click OK."
ls_html += "<P>" + inv_html_form.f_MakeSubmit &
   ("OK")
ls_html += inv_html_form.f_EndForm()
ls_html += "</BODY></HTML>"
DESTROY lds_data
```

4 Return the string variable:

```
RETURN ls_html
```

Returning a concatenated value

Another way to create HTML syntax is by creating separate string variables for each HTML element to be returned. This style of HTML creation makes it easier for you to perform error checking.

❖ **To return a concatenated value in the function's return statement:**

1 Declare variables in your user object function, including string variables for all the elements in your form:

```
String  ls_html = "<HTML>"
String  ls_body, ls_begin, ls_empid, ls_deptid
String  ls_last, ls_first, ls_status
String  ls_submit, ls_endform, ls_endhtml
String  ls_errortext = "<P>Retrieval Error"
DataStore  lds_data
Integer  li_return
Boolean  lb_checked
```

2 Access data. This example retrieves information for a specified employee ID:

```
lds_data = CREATE DataStore
lds_data.DataObject = "d_empbyid"
lds_data.SetTransObject(SQLCA)
li_return = lds_data.Retrieve(Long(as_empid))
IF li_return < 1 THEN
   RETURN ls_errortext + &
      "<P>Return code: " + String(li_return)
END IF
```

3 Create the HTML, calling u_html_form functions as needed:

```
ls_body = "<BODY><H1>Employee Information</H1>"
ls_begin = inv_html_form.f_BeginForm &
   ("/cgi-bin/pbcgi60.exe/myapp/" &
      + "uo_webtest/f_EmpInfo", 1)
// Set up employee information. They can input
// a newemp ID, so this field is an SLE.
ls_empid = "<P>Employee ID: "
ls_empid += inv_html_form.f_MakeSLE &
   ("as_empid", 5, 5, &
      String(lds_data.Object.emp_id[1]), FALSE)
ls_deptid = "<P>Department ID: "
ls_deptid += String(lds_data.Object.dept_id[1])
ls_last = "<P>Last Name: "
```

```
ls_last += lds_data.Object.emp_lname[1]
ls_first = "<P>First Name: "
ls_first += lds_data.Object.emp_fname[1]
// Set up radiobuttons for status
ls_status = "<P>Status:<P>"
IF lds_data.Object.status[1] = "A" THEN
   lb_checked = TRUE
ELSE
   lb_checked = FALSE
END IF
ls_status += inv_html_form.f_MakeRadio &
   ("as_status", "Active", lb_checked)
ls_status += "Active"
IF lds_data.Object.status[1] = "L" THEN
   lb_checked = TRUE
ELSE
   lb_checked = FALSE
END IF
ls_status += "<P>"
ls_status += inv_html_form.f_MakeRadio &
   ("as_status", "On Leave", lb_checked)
ls_status += "On Leave"
IF lds_data.Object.status[1] = "T" THEN
   lb_checked = TRUE
ELSE
   lb_checked = FALSE
END IF
ls_status += "<P>"
ls_status += inv_html_form.f_MakeRadio &
   ("as_status", "Terminated", lb_checked)
ls_status += "Terminated"
ls_status += "<P>Type a new Emp ID and click OK."
ls_submit += "<P>"
ls_submit += inv_html_form.f_MakeSubmit("OK")
ls_endform += inv_html_form.f_EndForm()
ls_endhtml += "</BODY></HTML>"
DESTROY lds_data
```

4 Return a concatenation of all the form elements:

```
RETURN ls_html + ls_body + ls_begin + &
   ls_empid + ls_deptid + ls_last + ls_first + &
      ls_status + ls_submit + ls_endform + &
        ls_endhtml
```

Creating HTML pages

The u_html_format functions

The vast majority of HTML pages displayed on the World Wide Web are not interactive. They display read-only text with optional hypertext links, Java applets, plug-ins, and ActiveX controls. The **u_html_format** custom class user object contains functions that you can call from your user object to generate HTML elements for use on Web pages.

These are the u_html_format functions you call to create selected HTML items:

To create this item	Call this function
HTML element	f_BeginPage
/HTML element	f_EndPage
Head element	f_BeginPageHeading
/Head element	f_EndPageHeading
Body element	f_BeginPageBody
/Body element	f_EndPageBody
Title element	f_SetPageTitle
Heading element	f_MakeHeading
Basefont element	f_SetBaseFont
Base element	f_SetBaseURL
BGSound	f_SetBGSound
Begin a table	f_BeginTable
Begin a table heading	f_BeginTableHeading
Create a cell in a table heading	f_MakeTableHeadingCell
End a table heading	f_EndTableHeading
Begin a table row	f_BeginTableRow
Create a cell in a table row	f_MakeTableRowCell
End a table row	f_EndTableRow
End a table	f_EndTable
Begin any type of list	f_BeginList
List items	f_MakeListItem

To create this item	Call this function
End any type of list	f_EndList
HREF element (link)	f_MakeLink
HR element (horizontal rule)	f_InsertHRule
Banner element	f_MakeBanner
Block quote element	f_MakeBlockQuote
Paragraph element	f_InsertParagraph
Line Break element	f_InsertLineBreak
Map element	f_BeginMapDef
Map AREA attribute	f_MakeMapArea
/Map element	f_EndMapDef
Set font attributes	f_FormatText
IMG element (image)	f_MakeImage
Embed element	f_MakeEmbed
Marquee element	f_MakeMarquee
Object element (ActiveX object)	f_MakeObject
Applet element	f_MakeApplet
Video clip	f_MakeVideoClip
Display a page at a specified location	f_RedirectClient
Display plain text	f_ReturnPlainText

Implementing u_html_format

When using u_html_format to create HTML syntax, you can implement it either as a service object or by creating customized u_html_format descendants.

FOR INFO For more information on service objects and customized descendants, see "Creating HTML forms" on page 163.

Programming with u_html_format

Creating a basic page

A typical HTML page has a heading area and a body area. It's best to learn HTML so you know which elements can only exist in the heading area, which elements can only exist in the body area, and which can exist in both.

You can optionally call functions to define defaults for the page.

Define u_html_format as an instance variable
These examples assume that u_html_format is defined as an instance variable.

❖ **To create a basic page:**

1 Call functions to create the heading area:

```
String  ls_html
String  ls_temp
ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle &
   ("Base page for testing u_html_format")
ls_html += inv_format.f_EndPageHeading()
```

2 Call functions to begin the body of the page and define page defaults:

```
ls_html += inv_format.f_BeginPageBody &
   (-12, -1, -9, -16)
ls_html += &
   inv_format.f_SetBaseFont(2, "Arial", -1)
```

3 Call functions to create the page content:

```
ls_html += inv_format.f_MakeHeading &
   ("Test u_html_format", 1, 1)
ls_html += inv_format.f_InsertHRule()
ls_html += inv_format.f_BeginList(1)
ls_html += inv_format.f_MakeListItem &
   ("Java", 1)
ls_html += inv_format.f_MakeListItem &
   ("JavaScript", 1)
ls_html += inv_format.f_MakeListItem &
   ("ActiveX", 1)
ls_html += inv_format.f_EndList(1)
```

4 Call functions to end the page:

```
ls_html += inv_format.f_EndPageBody()
ls_html += inv_format.f_EndPage()
```

5 Return the HTML:

```
RETURN ls_html
```

Creating hypertext links

The Anchor element defines an HTML hypertext link.

❖ **To create a hypertext link:**

1 Call the f_MakeLink function to create the link and the link text:

```
String  ls_html
String  ls_temp

ls_temp = inv_format.f_MakeLink &
   ("http://www.powersoft.com", &
      "Powersoft web site")
```

2 Add the link to the HTML:

```
ls_html += "For more information, see the "
ls_html += ls_temp
```

Creating tables

HTML tables feature a heading, one or more rows, and one or more columns. U_html_format includes overloaded functions to provide support for:

- Generic HTML
- Netscape table definition extensions
- Internet Explorer table definition extensions
- Table definition extensions that are supported by both Netscape and Internet Explorer

The example in the procedure that follows creates a table using generic HTML.

❖ **To create a table:**

1 Call the f_BeginTable function:

```
ls_html += inv_format.f_BeginTable &
   (0, 10, 3, 3, 75, -12, TRUE, TRUE, 0)
```

2 Create the table heading:

```
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
```

```
    ("Writer", 0, 4, 1, 1, TRUE)
ls_html += inv_format.f_MakeTableHeadCell &
    ("Book", 0, 1, 1, 1, TRUE)
```

3 Create the table body:

```
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
    ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertLineBreak(0)
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
    (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
```

4 Call the f_EndTable function:

```
ls_html += inv_format.f_EndTable()
```

Formatting text

HTML supports a many types of character formatting. U_html_format supports character formatting through the f_FormatText function.

❖ **To format text:**

1 Initialize the u_html_format s_font structure (defined as the ist_font instance variable) with the text formatting you want:

```
String  ls_html
String  ls_temp
...
inv_format.ist_font.Bold = TRUE
inv_format.ist_font.Italic = TRUE
```

You may need to reinitialize this structure
If you have previously used the s_font structure within the function, you should reinitialize all modified values.

2 Call the f_FormatText function, passing the text to be formatted and the s_font structure:

```
ls_temp = inv_format.f_FormatText &
    ("The Elements of Style", &
        inv_format.ist_font)
```

3 Add the formatted text to the HTML returned to the browser:

```
ls_html += ls_temp
ls_html += " by Strunk and White."
```

Embedding plug-ins

Plug-ins allow you to provide advanced functionality in an HTML page. For example, you can embed the PowerBuilder window plug-in in a page to provide an interactive application with all features available in a child window.

❖ **To embed a plug-in in an HTML page:**

1 Perform the steps required to define the plug-in to the Web server and the browser.

2 Call the f_MakeEmbed function:

```
...
ls_temp = inv_format.f_MakeEmbed &
   ("WinPlug", "/plgintst.pbd", &
      "WINDOW=w_empbydept", 0, 550, 80)
```

Displaying a URL

The Web.PB class library provides the f_RedirectClient function, which allows you to return the end user to a specified URL.

❖ **To redirect the end user to a specified URL:**

1 Create a user object instance variable to help the function control processing:

```
Boolean  ib_firsttime = TRUE
```

2 Create a function that returns a blob.

3 Call the f_RedirectClient function, passing the location of the URL to be displayed:

```
Blob   lblb_return

IF ib_firsttime THEN
   ib_firsttime = FALSE
   RETURN(inv_format.f_RedirectClient &
      ("http://www.powersoft.com/"))
ELSE
   SetNull(lblb_return)
   RETURN(lblb_return)
END IF
```

Returning plain text

The Web.PB class library provides the f_ReturnPlainText function, which allows you to return unformatted text to the Web browser. This capability is useful when returning simple text, such as error messages.

❖ **To redirect the end user to a specified URL:**

1 Create a user object instance variable to help the function control processing:

```
Boolean   ib_firsttime = TRUE
```

2 Create a function that returns a blob.

3 Call the f_ReturnPlainText function, passing the text to be displayed:

```
Blob   lblb_return
String  ls_text

IF ib_firsttime THEN
   ib_firsttime = FALSE
   ls_text = "This is a sample of plain text"
   RETURN(inv_format.f_ReturnPlainText(ls_text))
ELSE
   SetNull(lblb_return)
   RETURN(lblb_return)
END IF
```

Using HTML templates

Eliminate repetitive coding

Every HTML form has a standard set of elements that define the beginning and end of the page. In addition, your server application may implement a standardized look and feel that features the same heading and layout on every page.

To implement a standardized look and feel, you can define a template HTML file and use u_html_template functions to customize variable elements within the HTML. This saves you the coding required to create the standard elements on each page. It also helps to enforce consistency and reduces the possibility of coding errors.

These are the u_html_template functions:

To perform this action	Call this function
Establish defaults for directory paths	f_SetEnvironment
Open a template file	f_OpenTemplate
Replace the first occurrence of a substitution parameter	f_Replace
Replace all occurrences of a substitution parameter	f_ReplaceAll

Implementing u_html_template

You can implement u_html_template either as a service object or by creating customized u_html_template descendants.

FOR INFO For more information on service objects and customized descendants, see "Creating HTML forms" on page 163.

Programming with u_html_template

Create one or more template files

To use u_html_template, you first define one or more template HTML files. These files contain:

- HTML for the page heading and body
- Substitution parameters to be replaced by u_html_template functions

Define your own convention

Define your own convention for substitution parameters. These examples prefix substitution parameters with two ampersands (&&).

❖ **To create a template file:**

♦ Create a text file containing HTML, text, and substitution parameters:

```
<HTML>
<HEAD>
<TITLE>Acme Corporation</TITLE>
<BASEFONT SIZE=3 COLOR=BLACK FACE="ARIAL">
</HEAD>

<BODY BGCOLOR=TEAL>
<H3>AcmeWeb:Enterprise</H3>
<P>Records &&Start to &&End of &&Total
<P>&&Data
</HTML>
```

Accessing and customizing the template file

The template file contains substitution parameters that you change using u_html_template functions.

❖ **To access and customize the template file:**

1 (Optional) Call the f_SetEnvironment function to define the default path to the template file:

```
String  ls_html

inv_template.f_SetEnvironment &
   ("c:\Program Files\WebSite\htdocs", &
      "templates")
```

2 Load the template file into a string variable:

```
ls_html = inv_template.f_OpenTemplate &
   ("formtemp.htm")
```

3 Replace substitution parameters as necessary:

```
ls_html = inv_template.f_Replace &
   (ls_html, "&&Start", "1")
ls_html = inv_template.f_Replace &
   (ls_html, "&&End", "10")
ls_html = inv_template.f_Replace &
   (ls_html, "&&Total", "87")
ls_html = inv_template.f_Replace &
   (ls_html, "&&Data", &
     lds_page.Object.DataWindow.Data.HtmlTable)
```

Managing distributed sessions

Why you need to do this

In Web.PB, software running on the Web server acts as the client application and uses PowerBuilder's distributed capabilities to connect to your server application. Each Web.PB method (user object function) is a discrete unit of work to the server application. For example, after a Web.PB method returns HTML (or a URL) to the user, PowerBuilder triggers the ConnectionEnd event, disconnecting all open database connections and destroying all user objects and their instance variables.

This means that each time the Web server invokes a Web.PB method, the distributed application instantiates a new copy of the user object containing the method. There is no inherent knowledge of the state of previous instantiations.

What you do

When you design an interactive Web.PB application, you must have your server application implement a system to maintain state information for each active user. An active user in this case is any browser user whose current page was created by Web.PB and has the ability to respond to the server application.

Objects to use

The Web.PB class library includes two user objects (u_session and u_transaction) and a set of database tables that you can use to maintain high-level and detailed state management information.

State management concepts

Sessions and transactions

There are two main concepts related to state management as implemented by the Web.PB class library:

- **Session** High-level information related to a user's interaction with the server application. Session information might include a logon ID and password. A session can exist on its own or can own one or more transactions.

  You save session information using the u_session user object.

- **Transaction** Detailed information related to a user request. Transaction information might include cached pages of data that the server application displays as the user requests each page. A transaction can exist on its own or can be associated with a session.

This is not a database transaction
In this context, a transaction is an application-specific unit of work, not a database transaction.

You save transaction information using the u_transaction user object.

Role of the webpb database

The Web.PB class library saves session and transaction information in the webpb database, distributed with the Internet Tools.

Flexible state management

Every Web.PB application has unique state management needs. The webpb database and the u_session and u_transaction user objects together provide the flexibility needed to maintain whatever persistent information is important to your server application.

The state management database

Both u_session and u_transaction use tables in the webpb database to hold session and transaction information. This database is distributed with the Internet Tools.

Your application maintains a separate connection to the webpb database. Alternatively, you can pipe the tables to your application database and use its connection to access the tables.

These are the tables in the webpb database:

Table	Contents	Row usage
www_session	High-level session information	One row for each session
www_session_argument	Session argument information	One row for each session argument
www_new_session	Next sequential session ID	Only one row
www_transaction	High-level transaction information	One row for each transaction
www_transaction_argument	Transaction argument information	One row for each transaction argument
www_transaction_page	Transaction pages	One row for each transaction page

Table	Contents	Row usage
www_new_transaction	Next sequential transaction ID	Only one row

FOR INFO For complete information on these tables and their contents, use the Database painter to view column definitions.

Managing a session

Starting a session

You use a session to hold high-level information related to a browser user. You generate a unique ID for each session and associate it with the browser user by passing it back as a hidden field in the HTML. Subsequent user object functions take the session ID as an argument and use it to access session information in the webpb database.

The Web.PB class library uses the webpb database to store session information in the www_session and www_session_argument tables. In some cases it also uses the www_new_session_id table to generate session IDs.

❖ **To start a new session:**

1 Declare an instance variable in your user object of type u_session:

```
u_session  inv_session
```

This user object uses PowerBuilder's autoinstantiate feature so there is no need to code a CREATE statement.

2 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

3 Create a session ID by calling the f_GenerateID function:

```
Long   ll_sessionid
String  ls_html

ll_sessionid = inv_session.f_GenerateID(SQLCA)
```

Generating session IDs

U_session provides two versions of f_GenerateID. If you specify the Transaction object as an argument, it accesses the www_new_session_id table, increments it, and returns the next sequential session ID. If you call the f_GenerateID function with no arguments, it generates an ID based on the number of seconds since a fixed date.

4 Add the session to the database by calling the f_NewSession function, passing a Transaction object, the user's IP address, and the session ID:

```
inv_session.f_NewSession(SQLCA, remote_addr, &
   ll_sessionid)
```

This creates a row for the session in the www_session table.

Accessing the IP address

Access the user's IP address by passing the remote_addr CGI environment variable as a function argument.

5 Check the ii_errorcode instance variable to see if the function succeeded:

```
IF inv_session.ii_errorcode <> 0 THEN
   RETURN(inv_session.is_errormsg)
END IF
```

6 Return the session ID as a hidden field in the HTML:

```
ls_html += inv_html_form.f_MakeHidden &
   ("as_sessionid", String(ll_sessionid))
```

Session ID argument

To access the session ID in subsequent user object functions, define it as a function argument. In this example, you define as_sessionid as an argument to the user object function invoked when the user clicks the form's submit button.

Saving and accessing session arguments

When you create a session, you call Web.PB class library functions to generate and store a session ID, which exists as a row in the www_session table. Columns in this table include the session ID, the browser user's IP address, initial start date and time, and last access date and time.

You then define one or more session arguments for any user information that later functions will require to process successfully. Session arguments are application dependent. For example, one application may store the user's ID and password as session arguments; another application may store the Web server's interface type (CGI, ISAPI, or NSAPI).

❖ **To save session arguments:**

1 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

2 Call the f_SetArgumentValue function to save session arguments. This example saves the user ID and password (accessed in this example through the as_userid and as_userpass function arguments):

```
inv_session.f_SetArgumentValue &
   (SQLCA, ll_sessionid, "userid", as_userid)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN(inv_session.is_errormsg)
END IF
inv_session.f_SetArgumentValue &
   (SQLCA, ll_sessionid, "password", as_userpass)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN(inv_session.is_errormsg)
END IF
```

❖ **To access session arguments:**

1 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

2 Call the f_GetArgumentValue function to access session arguments. Note that this example accesses the session ID through the as_sessionid function argument:

```
String   ls_userid, ls_userpass

ls_userid = inv_session.f_GetArgumentValue &
   (SQLCA, Long(as_sessionid), "userid")
IF ls_userid = "" THEN
   RETURN(inv_session.is_errormsg)
END IF
ls_userpass = inv_session.f_GetArgumentValue &
   (SQLCA, Long(as_sessionid), "password")
IF ls_userid = "" THEN
```

```
        RETURN(inv_session.is_errormsg)
    END IF
```

Removing old sessions

Because of the World Wide Web's stateless nature, your server application has no way of knowing when a browser user stops work on an application, leaving open session information. What constitutes an old session is entirely application dependent. For some applications, five minutes of inactivity signals an old session; other applications might allow users 60 minutes of inactivity.

To periodically remove old sessions from the webpb database, you use the f_CleanUpSessions function in the Web.PB class library.

Call this function from the console application
Because this function can potentially remove all session information, it should not be accessible via Web.PB. You typically add this functionality to the console application (perhaps using the Timer event).

❖ **To remove old sessions from the state management database:**

1 Connect to the state management database in the console application's ConnectionBegin event or in the function itself.

2 Call the f_CleanUpSessions function:

```
inv_session.f_CleanUpSessions &
   (SQLCA, DateTime(Today(), Now()) )
CHOOSE CASE inv_session.ii_errorcode
   CASE 0
      RETURN("Old sessions removed")
   CASE IS <0
      RETURN(inv_session.is_errormsg)
END CHOOSE
```

Verifying and updating the session ID

Your application passes the session ID to the browser user as a hidden field in the HTML. If your server application deletes the session ID before the user responds to the HTML form, an error occurs. Before processing user requests, you should check to ensure that the passed session ID is valid:

- If the session ID is *valid*, you will need to call the f_UpdateLastAccess function to update the session ID in the webpb database. This prevents it from being deleted by the f_CleanUpSessions function.
- If the session ID is *invalid*, you will need to return the browser user to the server application's initial form.

❖ **To verify a session ID:**

- Call the f_VerifySessionID function before calling other session management functions. Return the user to a logon form if the session is invalid; update the session ID if the session is valid:

```
String   ls_html

IF NOT inv_session.f_VerifySessionID &
   (SQLCA, Long(as_sessionid)) THEN
   ls_html += inv_html_form.f_BeginForm &
      ("/cgi-bin/pbcgi60.exe/f_displaylogin", 0)
   ls_html += &
      "<P>Session has expired. Please Logon"
   ls_html += "<P>" + &
      inv_html_form.f_MakeSubmit("Begin Logon")
   ls_html += inv_html_form.f_EndForm()
   RETURN(ls_html)
ELSE
   inv_session.f_UpdateLastAccess &
      (SQLCA, Integer(as_sessionid), &
         DateTime(Today(), Now()) )
END IF
...
```

Managing transactions

Starting a transaction

You use a transaction to hold detailed information related to a browser user's task. You generate a unique ID for each transaction and associate it with the browser user by passing it back as a hidden field in the HTML, along with the session ID. Subsequent user object functions take the session and transaction IDs as arguments and use them to access session and transaction information in the webpb database.

The Web.PB class library uses the webpb database to store transaction information in the www_transaction and www_transaction_argument tables. In some cases it also uses the www_new_transaction_id table to generate transaction IDs.

❖ **To start a new transaction:**

1 Declare instance variables in your user object of type u_session and u_transaction:

```
u_session   inv_session
u_transaction   inv_transaction
```

These user objects use the PowerBuilder autoinstantiate feature, so there is no need to code CREATE statements.

2 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

3 Create a session ID by calling the inv_session f_GenerateID function:

```
Long   ll_sessionid, ll_transid

ll_sessionid = inv_session.f_GenerateID(SQLCA)
```

4 Create a transaction ID by calling the inv_transaction f_GenerateID function:

```
ll_transid = inv_transaction.f_GenerateID(SQLCA)
```

5 Add the transaction to the database by calling the f_NewTransaction function and passing a Transaction object, the session ID, and the transaction ID:

```
inv_transaction.f_NewTransaction &
   (SQLCA, ll_sessionid, ll_transid)
```

This creates a row for the transaction in the www_transaction table.

6 Check the ii_errorcode instance variable to see if the function succeeded:

```
IF inv_transaction.ii_errorcode <> 0 THEN
   RETURN(inv_transaction.is_errormsg)
END IF
```

7 Return the session ID and the transaction ID as hidden fields in the HTML:

```
ls_html += inv_html_form.f_MakeHidden &
   ("as_sessionid", String(ll_sessionid))
ls_html += inv_html_form.f_MakeHidden &
   ("as_transactionid", String(ll_transid))
```

Transaction ID argument

To access the transaction ID in subsequent user object functions, define it as a function argument. In this example, you define as_sessionid and as_transactionid as arguments to the user object function invoked when the user clicks the form's submit button.

Saving and accessing transaction arguments

After creating a transaction, you can define one or more arguments for any transaction information that later functions will require to process successfully. For example, you might save the displayed values to determine whether the browser user changed them before submitting a form.

❖ **To save transaction arguments:**

1 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

2 Call the f_SetArgumentValue function to save transaction arguments. This example saves the transaction type (in this example, the user requested page-by-page display of an employee list):

```
...
inv_transaction.f_SetArgumentValue &
   (SQLCA, ll_sessionid, ll_transid, &
      "transtype", "emplist")
IF inv_transaction.ii_errorcode <> 0 THEN
   RETURN(inv_transaction.is_errormsg)
END IF
```

❖ **To access transaction arguments:**

1 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

2 Call the f_GetArgumentValue function to access transaction arguments. Note that this example accesses the session and transaction IDs through function arguments:

```
String   ls_transtype

ls_transtype = &
   inv_transaction.f_GetArgumentValue(SQLCA, &
      Long(as_sessionid), &
         Long(as_transactionid), "transtype")
IF ls_transtype = "" THEN
   RETURN(inv_transaction.is_errormsg)
END IF
CHOOSE CASE ls_transtype
CASE "emplist"
...
```

Saving transaction pages

Many World Wide Web applications use an interface that displays one page of data at a time, accessing the server for each new page as requested by the browser user. This helps to reduce network traffic and gives the user more control over the data that is displayed.

U_transaction provides functions that you can use to access rows, segment and cache the result set, and return rows one page at a time.

❖ **To save transaction pages:**

1 Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.

2 Access the data to be stored in transaction pages:

```
DataStore   lds_data
Integer   li_return, li_numpages
Long   ll_transid

lds_data = CREATE DataStore
lds_data.DataObject = "d_employee"
lds_data.SetTransObject(gtr_trans)
li_return = lds_data.Retrieve()
```

3 Generate a transaction ID (this example assumes that a session ID already exists):

```
ll_transid = inv_transaction.f_GenerateID(SQLCA)
inv_transaction.f_NewTransaction &
   (SQLCA, Long(as_sessionid), ll_transid)
```

4 Create the transaction pages by calling the f_SetTransactionPage function:

```
li_numpages = &
   inv_transaction.f_SetTransactionPage(SQLCA, &
      Long(as_sessionid), ll_transid, 10, &
         lds_data)
```

5 Display an error message or the first page:

```
IF li_numpages < 1 THEN
   RETURN("No Pages Cached" &
      + inv_transaction.is_errormsg)
ELSE
   RETURN(this.f_newpage(as_sessionid, &
      String(ll_transid), "1", &
         String(li_numpages)))
```

```
END IF
```

The f_newpage function

In this example the f_newpage function is a user-defined function that displays the requested transaction page. The code for f_newpage is outlined in the next procedure.

Accessing transaction pages

This example function (f_newpage) takes as arguments the session ID, transaction ID, the requested page number, and the highest page.

❖ **To access a transaction page:**

1. Connect to the state management database in the application's ConnectionBegin event, in the user object's Constructor event, or in the function itself.
2. Create the DataStore to contain the retrieved page:

```
Integer  li_return, li_page
DataStore  lds_page
String  ls_html, ls_page

lds_page = CREATE DataStore
lds_page.DataObject = "d_employee"
```

3. Retrieve the transaction page by calling the f_GetTransactionPage function:

```
li_return = &
   inv_transaction.f_GetTransactionPage(SQLCA, &
      Long(as_sessionid), &
         Long(as_transactionid), &
            Integer(as_page), lds_page)
```

4. Create a return string containing the HTML to be returned:

```
ls_html += &
   lds_page.Object.DataWindow.Data.HtmlTable
ls_html += &
   "<P>Page " + as_page + " of " + as_numpages
```

5. (Optional) Append navigation links to the HTML to be returned. This example (which uses the default server application and user object from the pbweb.ini file) creates first page, previous page, next page, and last page links:

```
ls_html += "<P>"
```

```
// Make First page jump
IF Integer(as_page) <> 1 THEN
   ls_html += "<a HREF=" &
      + "/cgi-bin/pbcgi60.exe/f_newpage"
   ls_html += "?as_page=1"
   ls_html += "&as_numpages=" + as_numpages
   ls_html += "&as_sessionid=" + as_sessionid
   ls_html += &
      "&as_transactionid=" + as_transactionid
   ls_html += ">First Page</A> "
END IF
// Make Previous page jump
IF Integer(as_page) > 1 THEN
   ls_html += &
      "<a HREF=/cgi-bin/pbcgi60.exe/f_newpage"
   li_page = Integer(as_page) - 1
   ls_page = String(li_page)
   ls_html += "?as_page=" + ls_page
   ls_html += "&as_numpages=" + as_numpages
   ls_html += "&as_sessionid=" + as_sessionid
   ls_html += &
      "&as_transactionid=" + as_transactionid
   ls_html += ">Previous Page</A> "
END IF
// Make Next page jump
IF Integer(as_page) < Integer(as_numpages) THEN
   ls_html += &
      "<a HREF=/cgi-bin/pbcgi60.exe/f_newpage"
   li_page = Integer(as_page) + 1
   ls_page = String(li_page)
   ls_html += "?as_page=" + ls_page
   ls_html += "&as_numpages=" + as_numpages
   ls_html += "&as_sessionid=" + as_sessionid
   ls_html += &
      "&as_transactionid=" + as_transactionid
   ls_html += ">Next Page</A> "
END IF
// Make Last Page jump
IF as_page <> as_numpages THEN
   ls_html += &
      "<a HREF=/cgi-bin/pbcgi60.exe/f_newpage"
   ls_html += "?as_page=" + as_numpages
   ls_html += "&as_numpages=" + as_numpages
```

```
        ls_html += "&as_sessionid=" + as_sessionid
        ls_html += &
            "&as_transactionid=" + as_transactionid
        ls_html += ">Last Page</A>"
    END IF
```

6 Destroy the DataStore and return the HTML string:

```
DESTROY lds_page
RETURN ls_html
```

Removing old transactions

Because of the World Wide Web's stateless nature, your server application has no way of knowing when a browser user stops work on an application, leaving open session and transaction information. What constitutes an old transaction is entirely application dependent. For some applications, five minutes of inactivity signals an old transaction; other applications might allow users 20 minutes of inactivity.

To periodically remove old transactions from the webpb database, you use the f_CleanUpTransactions function in the Web.PB class library.

Call this function from the console application
Because this function can potentially remove all transaction information, it should not be accessible via Web.PB. You typically add this functionality to the console application (perhaps using the Timer event).

❖ **To remove old transactions from the state management database:**

1 Connect to the state management database in the console application's ConnectionBegin event or in the function itself.

2 Call the f_CleanUpTransactions function:

```
inv_transaction.f_CleanUpTransactions &
   (SQLCA, DateTime(Today(), Now()) )
CHOOSE CASE inv_transaction.ii_errorcode
   CASE 0
      RETURN("Old transactions removed")
   CASE IS <0
      RETURN(inv_transaction.is_errormsg)
END CHOOSE
```

Verifying and updating the transaction ID

Your application passes the transaction ID to the browser user as a hidden field in the HTML. If your server application deletes the transaction ID before the user responds to the HTML form, an error occurs. Before processing user requests, you should check to ensure that the passed transaction ID is valid:

- If the transaction ID is *valid*, you will need to call the f_UpdateLastAccess function to update the transaction ID in the webpb database. This prevents it from being deleted by the f_CleanUpTransactions function.
- If the transaction ID is *invalid*, return the browser user to a logical place in the server application.

❖ **To verify a transaction ID:**

- Call the f_VerifyTransactionID function before calling other transaction management functions. Return the user to a query form if the transaction is invalid; update the transaction ID if the transaction is valid:

```
String   ls_html

IF NOT inv_transaction.f_VerifyTransactionID &
   (SQLCA, Long(as_sessionid), &
      Long(as_transactionid)) THEN
   ls_html += inv_html_form.f_BeginForm &
      ("/cgi-bin/pbcgi60.exe/f_getinfo", 0)
   ls_html += &
      "<P>Transaction has expired. Please return"
   ls_html += "<P>" + &
      inv_html_form.f_MakeSubmit("Return")
   ls_html += inv_html_form.f_EndForm()
   RETURN(ls_html)
END IF
...
```

CHAPTER 6

Web.PB Class Library Object Reference

About this chapter

This chapter describes the user objects in webpb.pbl, the Web.PB class library.

Contents

Before you begin

Examples show how to implement functions and their arguments. They don't always show a complete scenario, and they don't include error checking.

u_html_form

Description

HTML form processing service. This object contains functions you call to generate syntax used in HTML forms—including buttons, edit fields, list boxes, dropdown listboxes, and radio buttons.

Ancestry

Inherits from NonVisualObject.

Library

Webpb.pbl

Usage

There are two ways to use u_html_form:

- **Define u_html_form as a service object:**

  1 Declare an instance variable in your user object of type u_html_form:

  ```
  u_html_form    inv_html_form
  ```

  This user object uses PowerBuilder's autoinstantiate feature so there is no need to code a CREATE statement.

  2 Call u_html_form functions from your user object functions as needed to create HTML syntax.

- **Create a standalone object using a customized u_html_form descendant:**

  1 Create a user object that descends from u_html_form.

  2 Create an f_CreateForm function that calls u_html_form functions to create HTML syntax. The ACTION statement returned in the HTML syntax should call f_ProcessForm, passing all necessary arguments.

  3 Create an f_ProcessForm function that accepts the arguments and responds to the user's input.

  4 Continue creating customized u_html_form descendants—one for each Web page to be displayed.

FOR INFO For complete information on using u_html_form, see "Creating HTML forms" on page 163.

See also

u_html_format
u_html_template

Instance variables

U_html_form includes instance variables:

Instance variable	Description	Data type	Access	Usage
ii_errorcode	Contains an error code for the last u_html_form function	Integer	Public	If a u_html_form function returns an empty string, use this instance variable to access high-level error information
ii_errormsg	Contains an error description for the last u_html_form function	String	Public	If a u_html_form function returns an empty string, use this instance variable to access detailed error information

Functions

U_html_form includes precoded object functions:

f_BeginForm	f_MakeMLE
f_EndForm	f_MakeRadio
f_MakeCheckBox	f_MakeReset
f_MakeDDLB	f_MakeSLE
f_MakeHidden	f_MakeSubmit
f_MakeLB	f_RedirectClient

f_BeginForm

Description

Creates syntax for the Form element.

Access

Public

Syntax

instancename.**f_BeginForm** (*action, method*)

Argument	Description
instancename	Instance name of u_html_form
action	String specifying the action the browser will take when the user clicks the form's submit button. This can specify either a URL or a user object function

Argument	Description
method	String specifying the submission method: ♦ 0 Get ♦ 1 Post

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Usage

Call this function to create syntax for an HTML FORM tag.

Examples

This example calls the f_BeginForm function:

```
...
ls_inputid = inv_html_form.f_MakeSLE &
   ("as_userid", 30, 30)
ls_inputpassword = inv_html_form.f_MakeSLE &
   ("as_userpass", 30, 30, "", TRUE)
ls_formmethodaction = inv_html_form.f_BeginForm&
   ("/cgi-bin/pbcgi60.exe/myapp/uo_test/f_login", 0)
...
```

f_EndForm

Description

Creates syntax to end a form (</FORM>).

Access

Public

Syntax

instancename.**f_EndForm ( )**

Argument	Description
instancename	Instance name of u_html_form

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Usage

Call this function to create syntax to end a form.

Examples

This example calls the f_EndForm function:

```
...
ls_inputtypevalue = inv_html_form.f_MakeSubmit &
   ("Begin Login")
ls_endform = inv_html_form.f_EndForm()
...
```

f_MakeCheckBox

Description

Creates syntax for a checkbox.

Access

Public

Syntax

instancename.**f_MakeCheckBox** (*name, returnvalue, checked*)

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the checkbox. If this form's action calls a user object function, this value must match one of the function's argument names
returnvalue	String specifying the value submitted if the checkbox is selected
checked	Boolean specifying whether the checkbox is selected initially

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *returnvalue* was NULL or an empty string
- -3 *checked* was NULL

Usage

Call this function to create syntax for a checkbox.

Examples

This example calls the f_MakeCheckBox function:

```
...
ls_cbx1 = inv_html_form.f_MakeCheckBox&
    ("benefits", "TRUE", FALSE)
...
```

f_MakeDDLB

Creates syntax for a dropdown listbox. There are two syntaxes:

To	Use
Create a dropdown listbox, specifying a DataStore column by name	Syntax 1
Create a dropdown listbox, specifying a DataStore column by number	Syntax 2

Syntax 1 **Create a dropdown listbox, specifying a DataStore column by name**

Description

Creates syntax for a dropdown listbox, using the specified column names from a passed DataStore.

Access

Public

Syntax

instancename.**f_MakeDDLB** (*name*, *datastore*, *datacolumn*, *displaycolumn* {, *initialrow* {, *beginrow*, *endrow* } })

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the dropdown listbox. If this form's action calls a user object function, this value must match one of the function's argument names
datastore	DataStore containing the data and display values displayed in the dropdown listbox
datacolumn	String specifying the name of the column that contains data values associated with *displaycolumn* values
displaycolumn	String specifying the name of the column whose values are to be displayed in the dropdown listbox
initialrow (optional)	Long specifying the row to be selected when the dropdown listbox displays
beginrow (optional)	Long specifying the first row to be included in the dropdown listbox. If you specify this value, you must also specify *endrow*
endrow (optional)	Long specifying the last row to be included in the dropdown listbox. If you specify this value, you must also specify *beginrow*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *datastore* was not a valid DataStore (probably not instantiated)
- -3 *datacolumn* is not a column in *datastore*
- -4 *displaycolumn* is not a column in *datastore*
- -5 *datacolumn* is of type TimeStamp
- -6 *displaycolumn* is of type TimeStamp
- -7 *beginrow* is NULL, 0, or greater than the number of rows in *datastore*
- -8 *endrow* is NULL, 0, or greater than the number of rows in *datastore*
- -9 *beginrow* is greater than *endrow*
- -10 *initialrow* is not between *beginrow* and *endrow*

Examples

This example calls the f_MakeDDLB function:

```
...
lds_dept = CREATE DataStore
lds_dept.dataobject = "d_department"
lds_dept.SetTransObject(gtr_trans)
IF lds_dept.Retrieve() < 1 THEN
   RETURN("Retrieval Error")
END IF

ls_dept = inv_html_form.f_MakeDDLB &
   ("as_dept", lds_dept, "dept_id", "dept_name")
...
```

Syntax 2

Create a dropdown listbox, specifying a DataStore column by number

Description

Creates syntax for a dropdown listbox, using the specified column numbers from a passed DataStore.

Access

Public

Syntax

instancename.**f_MakeDDLB** (*name, datastore, datacolumn, displaycolumn* {, *initialrow* {, *beginrow, endrow* } })

Argument	Description
instancename	Instance name of u_html_form

Argument	Description
name	String specifying a name for the dropdown listbox. If this form's action calls a user object function, this value must match one of the function's argument names
datastore	DataStore containing the data and display values displayed in the dropdown listbox
datacolumn	Integer specifying the number of the column that contains data values associated with *displaycolumn* values
displaycolumn	Integer specifying the number of the column whose values are to be displayed in the dropdown listbox
initialrow (optional)	Long specifying the row to be selected when the dropdown listbox displays
beginrow (optional)	Long specifying the first row to be included in the dropdown listbox. If you specify this value, you must also specify *endrow*
endrow (optional)	Long specifying the last row to be included in the dropdown listbox. If you specify this value, you must also specify *beginrow*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, check ii_errorcode and is_errormsg for more information. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *datastore* was not a valid DataStore (probably not instantiated)
- -3 *datacolumn* is not a column in *datastore*
- -4 *displaycolumn* is not a column in *datastore*
- -5 *datacolumn* is of type TimeStamp
- -6 *displaycolumn* is of type TimeStamp
- -7 *beginrow* is NULL, 0, or greater than the number of rows in *datastore*
- -8 *endrow* is NULL, 0, or greater than the number of rows in *datastore*
- -9 *beginrow* is greater than *endrow*
- -10 *initialrow* is not between *beginrow* and *endrow*

Examples

This example calls the f_MakeDDLB function:

```
...
lds_dept = CREATE DataStore
```

```
lds_dept.dataobject = "d_department"
lds_dept.SetTransObject(gtr_trans)
IF lds_dept.Retrieve() < 1 THEN
   RETURN("Retrieval Error")
END IF

ls_dept = inv_html_form.f_MakeDDLB&
   ("as_dept", lds_dept, 1, 2)
...
```

f_MakeHidden

Creates syntax for a hidden field. There are four syntaxes:

To	Use
Create a hidden field, specifying a string	Syntax 1
Create a hidden field, specifying an integer, long, or double	Syntax 2
Create a hidden field, specifying a decimal	Syntax 3
Create a hidden field, specifying a datetime	Syntax 4

Syntax 1

Create a hidden field, specifying a string

Description

Creates syntax for a hidden field containing a string.

Access

Public

Syntax

instancename.**f_MakeHidden** (*name, value*)

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the hidden field. If this form's action calls a user object function, this value must match one of the function's argument names
value	String containing the value to be stored in the hidden field

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *value* was NULL or an empty string

Examples

This example calls the f_MakeHidden function:

```
...
ll_sessionid = inv_session.f_GenerateID(SQLCA)
inv_session.f_NewSession &
   (SQLCA, remote_addr, ll_sessionid)
ls_sessionid = &
   inv_html_form.f_MakeHidden("as_sessionid", &
      String(ll_sessionid))
...
```

Syntax 2

Create a hidden field, specifying an integer, long, or double

Description

Creates syntax for a hidden field containing an integer, long, or double.

Access

Public

Syntax

instancename.**f_MakeHidden** (*name*, *value*)

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the hidden field. If this form's action calls a user object function, this value must match one of the function's argument names
value	Integer, long, or double containing the value to be stored in the hidden field

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *value* was NULL

Examples

This example calls the f_MakeHidden function:

```
Long   ll_empid

ll_empid = lds_data.Object.emp_id[1]
ls_html += &
   inv_html_form.f_MakeHidden("as_empid", ll_empid)
...
```

Syntax 3

Create a hidden field, specifying a decimal

Description

Creates syntax for a hidden field containing a decimal.

Access

Public

Syntax

instancename.**f_MakeHidden** (*name*, *value*)

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the hidden field. If this form's action calls a user object function, this value must match one of the function's argument names
value	Decimal containing the value to be stored in the hidden field

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -3 *value* was NULL

Examples

This example calls the f_MakeHidden function:

```
Decimal   ld_creditlimit
...
ls_html += &
   inv_html_form.f_MakeHidden("as_credit", &
      ld_creditlimit)
...
```

Syntax 4

Create a hidden field, specifying a datetime

Description

Creates syntax for a hidden field containing a datetime.

Access

Public

Syntax

instancename.**f_MakeHidden** (*name*, *value*)

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the hidden field. If this form's action calls a user object function, this value must match one of the function's argument names
value	Datetime containing the value to be stored in the hidden field

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -4 *value* was NULL

Examples

This example calls the f_MakeHidden function:

```
Datetime   ldtm_datetime
...
ldtm_datetime = DateTime(Today(), Now())
ls_html += &
    inv_html_form.f_MakeHidden("as_start", &
        ldtm_datetime)
...
```

f_MakeLB

Creates syntax for a listbox. There are two syntaxes:

To	Use
Create a dropdown listbox, specifying a DataStore column by name	Syntax 1
Create a dropdown listbox, specifying a DataStore column by number	Syntax 2

Syntax 1

Create a listbox, specifying a DataStore column by name

Description

Creates syntax for a listbox, using the specified column names from a passed DataStore.

Access

Public

Syntax

instancename.**f_MakeLB** (*name, datastore, datacolumn, displaycolumn* {, *numberoflines* {, *initialrow* {, *beginrow, endrow* } } })

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the listbox. If this form's action calls a user object function, this value must match one of the function's argument names
datastore	DataStore containing the data and display values displayed in the listbox

Argument	Description
datacolumn	String specifying the name of the column that contains data values associated with *displaycolumn* values
displaycolumn	String specifying the name of the column whose values are to be displayed in the listbox
numberoflines (optional)	Integer specifying the number of lines to display at one time
initialrow (optional)	Long specifying the row to be selected when the listbox displays
beginrow (optional)	Long specifying the first row to be included in the listbox. If you specify this value, you must also specify *endrow*
endrow (optional)	Long specifying the last row to be included in the listbox. If you specify this value, you must also specify *beginrow*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *datastore* was not a valid DataStore (probably not instantiated)
- -3 *datacolumn* is not a column in *datastore*
- -4 *displaycolumn* is not a column in *datastore*
- -5 *datacolumn* is of type TimeStamp
- -6 *displaycolumn* is of type TimeStamp
- -7 *beginrow* is NULL, 0, or greater than the number of rows in *datastore*
- -8 *endrow* is NULL, 0, or greater than the number of rows in *datastore*
- -9 *beginrow* is greater than *endrow*
- -10 *initialrow* is not between *beginrow* and *endrow*
- -11 *numberoflines* was NULL

Examples

This example calls the f_MakeLB function:

```
...
lds_dept = CREATE DataStore
lds_dept.dataobject = "d_department"
lds_dept.SetTransObject(gtr_trans)
IF lds_dept.Retrieve() < 1 THEN
```

```
    RETURN("Retrieval Error")
END IF

ls_dept = inv_html_form.f_MakeLB&
    ("as_dept", lds_dept, "dept_id", "dept_name")
...
```

Syntax 2 **Create a listbox, specifying a DataStore column by number**

Description

Creates syntax for a listbox, using the specified column numbers from a passed DataStore.

Access

Public

Syntax

instancename.**f_MakeLB** (*name, datastore, datacolumn, displaycolumn* {, *numberoflines* {, *initialrow* {, *beginrow, endrow* } })

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the listbox. If this form's action calls a user object function, this value must match one of the function's argument names
datastore	DataStore containing the data and display values displayed in the listbox
datacolumn	Integer specifying the number of the column that contains data values associated with *displaycolumn* values
displaycolumn	Integer specifying the number of the column whose values are to be displayed in the listbox
numberoflines (optional)	Integer specifying the number of lines to display at one time
initialrow (optional)	Long specifying the row to be selected when the listbox displays
beginrow (optional)	Long specifying the first row to be included in the listbox. If you specify this value, you must also specify *endrow*
endrow (optional)	Long specifying the last row to be included in the listbox. If you specify this value, you must also specify *beginrow*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *datastore* was not a valid DataStore (probably not instantiated)
- -3 *datacolumn* is not a column in *datastore*
- -4 *displaycolumn* is not a column in *datastore*
- -5 *datacolumn* is of type TimeStamp
- -6 *displaycolumn* is of type TimeStamp
- -7 *beginrow* is NULL, 0, or greater than the number of rows in *datastore*
- -8 *endrow* is NULL, 0, or greater than the number of rows in *datastore*
- -9 *beginrow* is greater than *endrow*
- -10 *initialrow* is not between *beginrow* and *endrow*
- -11 *numberoflines* was NULL

Examples

This example calls the f_MakeLB function:

```
...
lds_dept = CREATE DataStore
lds_dept.dataobject = "d_department"
lds_dept.SetTransObject(gtr_trans)
IF lds_dept.Retrieve() < 1 THEN
   RETURN("Retrieval Error")
END IF

ls_dept = inv_html_form.f_MakeLB&
   ("as_dept", lds_dept, 1, 2)
...
```

f_MakeMLE

Description

Creates syntax for a multiline edit.

Access

Public

Syntax

instancename.**f_MakeMLE** (*name, height, width, wordwrap* {, *initialvalue* })

Argument	Description
instancename	Instance name of u_html_form

Argument	Description
name	String specifying a name for the multiline edit. If this form's action calls a user object function, this value must match one of the function's argument names
height	Integer specifying the height of the multiline edit (in lines)
width	Integer specifying the width of the multiline edit (in characters)
wordwrap	Integer indicating whether all text lines are returned or only the first line is returned: ♦ 0 Wordwrap is disabled. Users must press ENTER between lines ♦ 1 Wordwrap is enabled. The form returns all lines in the multiline edit ♦ 2 Wordwrap is enabled. The form returns the first line only
initialvalue (optional)	String specifying an initial value for the multiline edit

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *width* is NULL
- -3 *height* is NULL
- -4 *wordwrap* is NULL
- -5 *initialvalue* is NULL or an empty string

Usage

Call this function to create syntax for a multiline edit.

Wordwrap
Not all browsers support wordwrap. Before implementing this feature, make sure your users run a browser that supports wordwrap.

Examples

This example calls the f_MakeMLE function:

```
...
ls_comment = inv_html_form.f_MakeMLE &
    ("as_comment", 10, 40, 0)
...
```

f_MakeRadio

Description

Creates syntax for a radio button.

Access

Public

Syntax

instancename.**f_MakeRadio** (*name*, *returnvalue*, *selected*)

Argument	Description
instancename	Instance name of u_html_form
name	String specifying a name for the radio button. If this form's action calls a user object function, this value must match one of the function's argument names
returnvalue	String specifying the value submitted if this radio button is selected
selected	Boolean specifying whether the radio button is selected initially

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *returnvalue* was NULL or an empty string
- -3 *selected* was NULL

Usage

Make sure you specify the text that displays next to the radio button.

Examples

This example calls the f_MakeRadio function:

```
...
ls_rb1 = "<P>" + &
    inv_html_form.f_MakeRadio&
        ("as_choice", "yes", TRUE) + &
            "Yes"
ls_rb2 = "<P>" + &
    inv_html_form.f_MakeRadio&
        ("as_choice", "no", FALSE) + &
            "No"
...
```

f_MakeReset

Description

Creates syntax for a command button that clears all form fields.

Access

Public

Syntax

instancename.**f_MakeReset** (*label* {, *name* {, *height*, *width* } })

Argument	Description
instancename	Instance name of u_html_form
label	String specifying the text that appears on the button
name (optional)	String specifying a name for the button
height (optional)	Integer specifying button height. If you specify this value, you must also specify *width*
width (optional)	Integer specifying button width. If you specify this value, you must also specify *height*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *label* was NULL or an empty string
- -2 *name* was NULL or an empty string
- -3 *height* is NULL or 0
- -4 *width* is NULL or 0

Usage

Call this function to create a reset button on a form.

Examples

This example calls the f_MakeReset function:

```
...
ls_reset += inv_html_form.f_MakeReset("Reset")
...
```

f_MakeSLE

Description

Creates syntax for a single line edit.

Access

Public

Syntax

instancename.**f_MakeSLE** (*name*, *length* {, *width* {, *initialvalue* {, *password* } } })

Argument	Description
instancename	Instance name of u_html_form

Argument	Description
name	String specifying a name for the single line edit. If this form's action calls a user object function, this value must match one of the function's argument names
length	Integer specifying the maximum number of characters in the single line edit
width (optional)	Integer specifying the visible width of the field (in characters)
initialvalue (optional)	String specifying an initial value for the single line edit
password (optional)	Boolean indicating whether the single line edit is a password field. Password fields display all characters as asterisks

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *name* was NULL or an empty string
- -2 *length* is NULL
- -3 *width* is NULL
- -4 *initialvalue* is NULL
- -5 *password* is NULL

Usage

Call this function to create syntax for a single line edit.

Examples

This example calls the f_MakeSLE function:

```
...
ls_inputid = inv_html_form.f_MakeSLE &
    ("as_userid", 30, 30)
ls_inputpassword = inv_html_form.f_MakeSLE&
    ("as_userpass", 30, 30, "", TRUE)
...
```

f_MakeSubmit

Description

Creates syntax for a command button that submits form contents to the Web server.

Access

Public

Syntax

instancename.**f_MakeSubmit** (*label* {, *name* {, *height*, *width* } })

Argument	Description
instancename	Instance name of u_html_form
label	String specifying the text that appears on the button
name (optional)	String specifying a name for the button
height (optional)	Integer specifying button height. If you specify this value, you must also specify *width*
width (optional)	Integer specifying button width. If you specify this value, you must also specify *height*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *label* was NULL or an empty string
- -2 *name* was NULL or an empty string
- -3 *height* is NULL or 0
- -4 *width* is NULL or 0

Usage

Call this function to create a button that submits form contents to the Web server.

Examples

This example calls the f_MakeSubmit function:

```
...
ls_submit = inv_html_form.f_MakeSubmit("OK")
...
```

f_RedirectClient

Description

Returns the end user to a specified URL.

Access

Public

Syntax

instancename.**f_RedIrectClient** (*redirecturl*)

Argument	Description
instancename	Instance name of u_html_form
redirecturl	String containing the URL to be accessed and displayed on the client's browser

Return value

Blob. Returns HTML syntax prefixed by the text/html content type if the function succeeds and a blob containing an empty string if *redirecturl* is null or empty.

Usage

Call this function to send the end user to a specified URL.

Examples

This example calls the f_RedirectClient function. It assumes an ib_firsttime boolean instance variable:

```
Blob   lblb_return

IF ib_firsttime THEN
   ib_firsttime = FALSE
   RETURN(inv_form.f_RedirectClient &
      ("http://www.powersoft.com/"))
ELSE
   SetNull(lblb_return)
   Return(lblb_return)
END IF
```

u_html_format

Description

HTML generation service. This object contains functions that you call to generate syntax for HTML elements, including lists, tables, images, banners, applets, and embedded objects.

FOR INFO For information on generating syntax for HTML forms, see u_html_form on page 194.

Ancestry

Inherits from NonVisualObject.

Library

Webpb.pbl

Usage

To use u_html_format:

1 Declare an instance variable in your user object of type u_html_format:

```
u_html_format    inv_format
```

This user object uses PowerBuilder's autoinstantiate feature, so there is no need to code a CREATE statement.

2 Call u_html_format functions from your user object functions as needed to create HTML syntax.

FOR INFO For complete usage information on u_html_format, see "Creating HTML pages" on page 170.

Structures

Structure	Field	Data type
S_column	alignment	Integer
	bgcolor	String
	bgimageurl	String
	bordercolor	Long
	bordercolordark	Long
	bordercolorlight	Long
	borderwidth	Integer
	columnid	Integer
	columnname	String
	columntype	String
	fontspec	S_font
	format	String

Structure	Field	Data type
	heading	String
	textcolor	Long
	units	Integer
	valignment	Integer
	width	Integer
	wordwrap	Boolean
S_font	bold	Boolean
	italic	Boolean
	underline	Boolean
	strikethrough	Boolean
	superscript	Boolean
	subscript	Boolean
	fixedwidth	Boolean
	blink	Boolean
	fontsize	Integer
	fontname	String
	fontcolor	Long
S_table	alignment	Integer
	borderwidth	Integer
	cellpadding	Integer
	cellspacing	Integer
	width	Integer
	backcolor	Long
	clearimage	Integer
	cleartext	Boolean
	wordwrap	Boolean
	numcols	Integer

Structure	Field	Data type
	bgimageurl	String
	bordercolor	Long
	bordercolorlight	Long
	bordercolordark	Long
	outsideborder	Integer
	insideborder	Integer

See also

u_html_form
u_html_template

Instance variables

U_html_format includes instance variables:

Instance variable	Description	Data type	Access	Usage
ii_errorcode	Contains an error code for the last u_html_format function	Integer	Public	If a u_html_format function returns an empty string, use this instance variable to access high-level error information
is_errormsg	Contains an error description for the last u_html_format function	String	Public	If a u_html_format function returns an empty string, use this instance variable to access detailed error information
ist_column and ist_columns[ ]	Contains information on a DataWindow column	S_column	Public	Populate this structure with the f_GetColSpecs, f_GetCompSpecs, or f_GetTextSpecs function, for use as input to the f_MakeTableBodyCell or f_MakeTableHeadCell function. Use ist_columns[ ] to store data on multiple columns

Instance variable	Description	Data type	Access	Usage
ist_font and ist_fonts[ ]	Contains information on font formatting	S_font	Public	Populate this structure with formatting information before using it as an argument to the f_FormatText function. Use ist_fonts[ ] to store multiple formats
ist_table and ist_tables[ ]	Contains information on table formatting	S_table	Public	Populate this structure with table information before using it as an argument to the f_BeginTable function. Use ist_tables[ ] to store data on multiple table formats

Functions

U_html_format includes precoded object functions:

f_BeginList	f_MakeApplet
f_BeginMapDef	f_MakeBanner
f_BeginPage	f_MakeBlockQuote
f_BeginPageBody	f_MakeEmbed
f_BeginPageHeading	f_MakeHeading
f_BeginTable	f_MakeImage
f_BeginTableBody	f_MakeLink
f_BeginTableHead	f_MakeListItem
f_BeginTableRow	f_MakeMapArea
f_EndList	f_MakeMarquee
f_EndMapDef	f_MakeObject
f_EndPage	f_MakeTableBodyCell
f_EndPageBody	f_MakeTableHeadCell
f_EndPageHeading	f_MakeVideoClip
f_EndTable	f_RedirectClient
f_EndTableRow	f_ReturnHTMLText
f_FormatText	f_ReturnNonText
f_GetColSpecs	f_ReturnPlainText
f_GetCompSpecs	f_SetBaseFont
f_GetTextSpecs	f_SetBaseURL
f_InsertHRule	f_SetBGSound
f_InsertLineBreak	f_SetPageTitle
f_InsertParagraph	

f_BeginList

Description

Creates HTML syntax to begin the specified type of list.

Access

Public

Syntax

instancename.**f_BeginList** (*listtype*)

Argument	Description
instancename	Instance name of u_html_format
listtype	Integer specifying the type of list to begin: ♦ 1 Bulleted list (<UL>) ♦ 2 Numbered list (<OL>) ♦ 3 Definition list (<DL>) ♦ 4 Directory list (<DIR>) ♦ 5 Menu list (<MENU>)

Return value

String. Returns HTML syntax if the function succeeds an empty string if *listtype* is invalid.

Examples

This example calls the f_BeginList function:

```
String    ls_html
String    ls_temp
...
ls_html += inv_format.f_BeginList(1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_common", &
      "Test common elements")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_netscape", &
      "Test Netscape HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_explorer", &
      "Test Internet Explorer HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_html += inv_format.f_EndList(1)
...
```

f_BeginMapDef

Description

Creates HTML syntax to create an image map.

Access

Public

Syntax

instancename.**f_BeginMapDef** (*mapname*)

Argument	Description
instancename	Instance name of u_html_format
mapname	String specifying the name of the map

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *mapname* is invalid (an empty string).

Usage

You need to call the f_MakeMapArea function to define hypertext links within the map.

Examples

This example calls the f_BeginMapDef function:

```
ls_html += inv_format.f_BeginMapDef("map1")
ls_html += inv_format.f_MakeMapArea &
   ("localhost/", "10,10,30,30", 0, 0, TRUE)
ls_html += inv_format.f_MakeMapArea &
   ("localhost/webpb/", "10,30,30,60", 0, 0, FALSE)
ls_html += inv_format.f_EndMapDef()
ls_html += inv_format.f_makeimage ("/world.gif", &
   TRUE, 1, 200, 164, 2, 3, 3, "World Map", &
      TRUE, "#map1")
```

f_BeginPage

Description

Creates the HTML element, which begins the HTML portion of a page.

Access

Public

Syntax

instancename.**f_BeginPage** ( )

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_BeginPage function:

```
String    ls_html
String    ls_temp

ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle &
   ("Base page for testing u_html_format")
ls_html += inv_format.f_EndPageHeading()
...
```

f_BeginPageBody

Description

Creates the Body element for the page's body text.

Access

Public

Syntax

instancename.**f_BeginPageBody** ({*bgcolor* {, *textcolor*, *linktextcolor*, *vlinktextcolor* {, *leftmargin*, *topmargin* {, *bgurl*, *bgimageposition* } } } })

Argument	Description
instancename	Instance name of u_html_format

Argument	Description
bgcolor (optional)	Long specifying the background color. Specify values as follows: ♦ 0 Default color ♦ -1 Black ♦ -2 White ♦ -3 Silver ♦ -4 Gray ♦ -5 Maroon ♦ -6 Red ♦ -7 Yellow ♦ -8 Lime ♦ -9 Green ♦ -10 Olive ♦ -11 Teal ♦ -12 Aqua ♦ -13 Blue ♦ -14 Navy ♦ -15 Purple ♦ -16 Fuchsia ♦ *other values* Long value that the function converts into hexadecimal number representing the color Specify -999 to use the default background color
textcolor (optional)	Long specifying the text color. If you specify this argument you must also specify *linktextcolor* and *vlinktextcolor*. Specify -999 to use the default text color FOR INFO For a list of color specifications, see *bgcolor*
linktextcolor (optional)	Long specifying the color for hypertext links. If you specify this argument you must also specify *textcolor* and *vlinktextcolor*. Specify -999 to use the default link text color FOR INFO For a list of color specifications, see *bgcolor*
vlinktextcolor (optional)	Long specifying the color for previously accessed hypertext links. If you specify this argument you must also specify *textcolor* and *linktextcolor*. Specify -999 to use the default color for previously accessed hypertext links FOR INFO For a list of color specifications, see *bgcolor*

Argument	Description
leftmargin (optional)	Integer specifying the left margin for the body of the page. If you specify this argument you must also specify *topmargin*. Specify 0 to use the default left margin
topmargin (optional)	Integer specifying the top margin for the body of the page. If you specify this argument you must also specify *leftmargin*. Specify 0 to use the default top margin
bgurl (optional)	String specifying a URL to be used as the page's background image. Specify an empty string ("") to omit a background image
bgimageposition (optional)	Boolean specifying whether the background image scrolls with the page (TRUE) or is fixed (FALSE). If you specify this argument you must also specify *bgurl*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_BeginPageBody function:

```
...
ls_html += inv_format.f_BeginPageBody &
    (-12, -1, -9, -16)
...
```

f_BeginPageHeading

Description

Creates HTML syntax for the Head element (to begin a page heading).

Access

Public

Syntax

instancename.**f_BeginPageHeading** ( )

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_BeginPageHeading function:

```
String    ls_html
String    ls_temp
```

```
ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle &
   ("Base page for testing u_html_format")
ls_html += inv_format.f_EndPageHeading()
...
```

f_BeginTable

Creates HTML syntax to begin a table There are four syntaxes:.

To	Use
Create a table for all browsers	Syntax 1
Create a table using Netscape extensions	Syntax 2
Create a table using Internet Explorer extensions	Syntax 3
Create a table using both Netscape and Internet Explorer extensions	Syntax 4

Syntax 1

Create a table for all browsers

Description

Creates HTML syntax to begin a table. This version of the function generates generic syntax for all browsers.

Access

Public

Syntax

instancename.**f_BeginTable** (*alignment*, *borderwidth*, *cellpadding*, *cellspacing*, *width*, *bgcolor*)

Argument	Description
instancename	Instance name of u_html_format
alignment	Integer specifying table alignment. Values are: ♦ 0 Left ♦ 1 Center (Netscape only) ♦ 2 Right ♦ 3 Full justify (Netscape only) ♦ 4 Bleed left (Netscape only) ♦ 5 Bleed right (Netscape only)
borderwidth	Integer specifying the width of the table border. Specify 0 to use the default border width

Argument	Description
cellpadding	Integer specifying the amount of space between a cell border and its data. Specify 0 to use the default cell padding
cellspacing	Integer specifying the amount of space between table cells. Specify 0 to use the default cell spacing
width	Integer specifying the screen width percentage occupied by the table. Specify 0 to omit the WIDTH attribute from the returned HTML
bgcolor	Long specifying the table background color. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *alignment* is invalid.

Usage

Use Syntax 2 to create a table using Netscape extensions; use Syntax 3 to create a table using Internet Explorer extensions; use Syntax 4 to create a table using Netscape and Internet Explorer extensions.

Examples

This example calls the f_BeginTable function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE)
...
```

Syntax 2

Create a table using Netscape extensions

Description

Creates HTML syntax to begin a table using Netscape extensions. This version of the functions generates Netscape table extensions.

Access

Public

Syntax

instancename.**f_BeginTable** (*alignment, borderwidth, cellpadding, cellspacing, width, bgcolor, wordwrap, cleartext, clearimage*)

Argument	Description
instancename	Instance name of u_html_format
alignment	Integer specifying table alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right ♦ 3 Full justify ♦ 4 Bleed left ♦ 5 Bleed right
borderwidth	Integer specifying the width of the table border. Specify 0 to use the default border width
cellpadding	Integer specifying the amount of space between a cell border and its data. Specify 0 to use the default cell padding
cellspacing	Integer specifying the amount of space between table cells. Specify 0 to use the default cell spacing
width	Integer specifying the screen width percentage occupied by the table. Specify 0 to omit the WIDTH attribute from the returned HTML
bgcolor	Long specifying the table background color. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
wordwrap	Boolean specifying whether text wraps within the table cells
cleartext	Boolean specifying whether a preceding text table wraps around this table
clearimage	Integer specifying how the table wraps around an image. Values are: ♦ 0 Clear all ♦ 1 Clear left ♦ 2 Clear right

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *alignment* was NULL or an invalid value
- -2 *clearimage* was NULL or an invalid value

Usage

Use Syntax 1 to create a table using generic HTML; use Syntax 3 to create a table using Internet Explorer extensions; use Syntax 4 to create a table using Netscape and Internet Explorer extensions.

Examples

This example calls the f_BeginTable function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12, TRUE, TRUE, 0)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE, 90)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE, 90)
...
```

Syntax 3

Create a table using Internet Explorer extensions

Description

Creates HTML syntax to begin a table using Internet Explorer extensions.

Access

Public

Syntax

instancename.**f_BeginTable** (*alignment, borderwidth, cellpadding, cellspacing, width, bgcolor, numcolumns, bgimageurl, bordercolor, bordercolorlight, bordercolordark, outsideborder, insideborder*)

Argument	Description
instancename	Instance name of u_html_format
alignment	Integer specifying table alignment. Values are: ♦ 0 Left ♦ 2 Right
borderwidth	Integer specifying the width of the table border. Specify 0 to use the default border width
cellpadding	Integer specifying the amount of space between a cell border and its data. Specify 0 to use the default cell padding
cellspacing	Integer specifying the amount of space between table cells. Specify 0 to use the default cell spacing
width	Integer specifying the screen width percentage occupied by the table. Specify 0 to omit the WIDTH attribute from the returned HTML

Argument	Description
bgcolor	Long specifying the table background color. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
numcolumns	Integer specifying the number of columns in the table. Specify 0 to omit the COLS attribute from the returned HTML
bgimageurl	String specifying the URL of an image to use as the table's background. Specify an empty string to omit the BACKGROUND attribute from the returned HTML
bordercolor	Long specifying a border color for the table. Specify -999 to use the default border color
bordercolorlight	Long specifying the border's highlight color; used to make a 3D table border. Specify -999 to omit the BORDERCOLORLIGHT attribute from the returned HTML
bordercolordark	Long specifying the border's shadow color; used to make a 3D table border. Specify -999 to omit the BORDERCOLORDARK attribute from the returned HTML
outsideborder	Integer specifying the type of border for the outside edge of the table. Values are: ♦ 0 None ♦ 1 All ♦ 2 Top and bottom ♦ 3 Right and left ♦ 4 Top ♦ 5 Bottom ♦ 6 Left ♦ 7 Right
insideborder	Integer specifying the type of border for inside cell edges of the table. Values are: ♦ 0 None ♦ 1 All ♦ 2 Rows ♦ 3 Columns ♦ 4 Groups (groups are defined by <THEAD><TBODY><TFOOT> elements)

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *alignment* is invalid.

Usage

Use Syntax 1 to create a table using generic HTML; use Syntax 2 to create a table using Netscape extensions; use Syntax 4 to create a table using Netscape and Internet Explorer extensions.

Examples

This example calls the f_BeginTable function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12, 2, "", -1, -12, -11, 2, 2)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE, "", -7, -14, &
      -12, -11)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE, "", -7, -14, -12, -11)
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE, "", -3, &
      -14, -12, -11)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE, "", -3, -14, -12, &
      -11)
ls_html += inv_format.f_EndTableRow()
...
```

Syntax 4

Create a table using Netscape and Internet Explorer extensions

Description

Creates HTML syntax to begin a table.

Access

Public

Syntax

instancename.**f_BeginTable** (*tablespec*)

Argument	Description
instancename	Instance name of u_html_format

Argument	Description
tablespec	S_table structure containing table specifications. This argument is passed by reference

Return value

String. Returns HTML syntax if the function succeeds and an empty string if ist_table.*alignment* is invalid.

Usage

This version of the function generates generic syntax for all browsers. Initialize the s_table structure with table creation specifications before calling this function. For information on populating the elements in s_table, see the other syntaxes for this function.

Use Syntax 1 to create a table using generic HTML; use Syntax 2 to create a table using Netscape extensions; use Syntax 3 to create a table using Internet Explorer extensions.

Examples

This example calls the f_BeginTable function:

```
...
inv_format.ist_table.Alignment = 0
inv_format.ist_table.BackColor = -7
inv_format.ist_table.BorderColor = -1
inv_format.ist_table.BorderWidth = 5
inv_format.ist_table.CellPadding = 3
inv_format.ist_table.CellSpacing = 3
inv_format.ist_table.NumCols = 2
inv_format.ist_table.Width = 75
inv_format.ist_table.WordWrap = TRUE

ls_html += &
   inv_format.f_BeginTable(inv_format.ist_table)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
...
```

f_BeginTableBody

Description

Creates HTML syntax to begin the table body.

Access

Public

Syntax

instancename.**f_BeginTableBody** ( )

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_BeginTableBody function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
...
```

f_BeginTableHead

Description

Creates HTML syntax to begin a table heading.

Access

Public

Syntax

instancename.**f_BeginTableHead** ( )

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_BeginTableHead function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12, TRUE, TRUE, 0)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
```

```
    ("Writer", 0, 4, 1, 1, TRUE)
ls_html += inv_format.f_MakeTableHeadCell &
    ("Book", 0, 1, 1, 1, TRUE)
...
```

f_BeginTableRow

Description

Creates HTML syntax to begin a table row.

Access

Public

Syntax

instancename.**f_BeginTableRow** (*halign*, *valign* {, *bgimageurl*, *bgcolor*, *bordercolor*, *bordercolorlight*, *bordercolordark* }))

Argument	Description
instancename	Instance name of u_html_format
halign	Integer specifying horizontal alignment. Values are: ♦ 0 Left ♦ 1 Center (Internet Explorer only) ♦ 2 Right ♦ 7 Middle (Netscape only)
valign	Integer specifying vertical alignment. Values are: ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
bgimageurl (optional)	String specifying the URL of an image to use as the row's background. Specify an empty string to omit the BACKGROUND attribute from the returned HTML
bgcolor (optional)	Long specifying a background color for the row. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
bordercolor (optional)	Long specifying a border color for the row. Specify -999 to use the default border color
bordercolorlight (optional)	Long specifying the border's highlight color; used to make a 3D row border. Specify -999 to omit the BORDERCOLORLIGHT attribute from the returned HTML

Argument	Description
bordercolordark (optional)	Long specifying the border's shadow color; used to make a 3D row border. Specify -999 to omit the BORDERCOLORDARK attribute from the returned HTML

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halign* or *valign* are invalid. If an error occurs, check ii_errorcode and is_errormsg for more information.

Usage

The optional arguments for this function are for Internet Explorer.

Examples

This example calls the f_BeginTableRow function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
...
```

f_EndList

Description

Creates HTML syntax to end the specified type of list.

Access

Public

Syntax

instancename.**f_EndList** (*listtype*)

Argument	Description
instancename	Instance name of u_html_format
listtype	Integer specifying the type of list to end. Values are: ♦ 1 Bulleted list (</UL>) ♦ 2 Numbered list (</OL>) ♦ 3 Definition list (</DL>) ♦ 4 Directory list (</DIR>) ♦ 5 Menu list (</MENU>)

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *listtype* is invalid.

Examples

This example calls the f_EndList function:

```
String    ls_html
String    ls_temp
...
ls_html += inv_format.f_BeginList(1)
ls_temp = inv_format.f_MakeLink &
    ("/cgi-bin/pbcgi60.exe/f_common", &
"Test common elements")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
    ("/cgi-bin/pbcgi60.exe/f_netscape", &
       "Test Netscape HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
    ("/cgi-bin/pbcgi60.exe/f_explorer", &
       "Test Internet Explorer HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_html += inv_format.f_EndList(1)
...
```

f_EndMapDef

Description

Creates HTML syntax to end a client-side map definition.

Access

Public

Syntax

instancename.**f_EndMapDef ( )**

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_EndMapDef function:

```
...
ls_html += inv_format.f_BeginMapDef("map1")
ls_html += inv_format.f_MakeMapArea &
    ("localhost/", "10,10,30,30", 0, 0, TRUE)
```

```
ls_html += inv_format.f_MakeMapArea &
   ("localhost/webpb/", "10,30,30,60", 0, 0, FALSE)
ls_html += inv_format.f_EndMapDef()
ls_html += inv_format.f_makeimage ("/world.gif", &
   TRUE, 1, 200, 164, 2, 3, 3, "World Map", &
      TRUE, "#map1")
...
```

f_EndPage

Description

Creates HTML syntax to end the HTML portion of a page.

Access

Public

Syntax

instancename.**f_EndPage ( )**

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_EndPage function:

```
...
ls_html += inv_format.f_EndPage()

RETURN ls_html
```

f_EndPageBody

Description

Creates HTML syntax to end the body of a page.

Access

Public

Syntax

instancename.**f_EndPageBody ( )**

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_EndPageBody function:

```
...
ls_html += inv_format.f_EndPageBody()
ls_html += inv_format.f_EndPage()

RETURN ls_html
```

f_EndPageHeading

Description

Creates HTML syntax to end a page heading.

Access

Public

Syntax

instancename.**f_EndPageHeading ( )**

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_EndPageHeading function:

```
String    ls_html
String    ls_temp

ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle &
   ("Base page for testing u_html_format")
ls_html += inv_format.f_EndPageHeading()
...
```

f_EndTable

Description

Creates HTML syntax to end a table.

Access

Public

Syntax

instancename.**f_EndTable ( )**

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_EndTable function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
ls_html += inv_format.f_EndTable()
...
```

f_EndTableRow

Description

Creates HTML syntax to end a table row.

Access

Public

Syntax

instancename.**f_EndTableRow ( )**

Argument	Description
instancename	Instance name of u_html_format

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_EndTableRow function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
```

```
    (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
...
```

f_FormatText

Description

Applies HTML font formatting to the passed string.

Access

Public

Syntax

instancename.**f_FormatText** (*text, fontspec*)

Argument	Description
instancename	Instance name of u_html_format
text	String containing the text to be formatted
fontspec	S_font structure specifying the HTML font formatting to be applied to *text*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Usage

Before calling this function, initialize the ist_font structure variable with the formatting you want.

Examples

This example calls the f_FormatText function:

```
String    ls_html
String    ls_temp
...
inv_format.ist_font.Bold = TRUE
inv_format.ist_font.FontSize = 5
inv_format.ist_font.Underline = TRUE
ls_temp = inv_format.f_FormatText&
   ("Test u_html_format", inv_format.ist_font)
ls_html += ls_temp
...
```

f_GetColSpecs

Description

Returns detailed information for the specified column name in the specified DataStore.

Access

Public

Syntax

instancename.**f_GetColSpecs** (*datastore, columnname, columninfo*)

Argument	Description
instancename	Instance name of u_html_format
datastore	DataStore containing the column for which specifications are returned (passed by reference)
columnname	String specifying the name of the column within *datastore*
columninfo	S_column structure into which the function places column specifications (passed by reference)

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- <0 An error occurred

Usage

Use the output from this function as input to the f_MakeTableHeadCell and f_MakeTableBodyCell functions.

Examples

This example calls the f_GetColSpecs function:

```
...
inv_format.f_GetColSpecs(lds_datastore, &
    "emp_id", inv_format.ist_column)
...
```

f_GetCompSpecs

Description

Returns detailed information for the specified computed column in the specified DataStore.

Access

Public

Syntax

instancename.**f_GetCompSpecs** (*datastore, columnname, columninfo*)

Argument	Description
instancename	Instance name of u_html_format
datastore	DataStore containing the column for which specifications are returned (passed by reference)

Argument	Description
columnname	String specifying the name of the computed column within *datastore*
columninfo	S_column structure into which the function places computed column specifications (passed by reference)

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- <0 An error occurred

Usage

Use the output from this function as input to the f_MakeTableHeadCell and f_MakeTableBodyCell functions.

Examples

This example calls the f_GetCompSpecs function:

```
...
inv_format.f_GetCompSpecs(lds_datastore, &
    "emp_fullname", inv_format.ist_column)
...
```

f_GetTextSpecs

Description

Returns detailed information for the specified text field in the specified DataStore.

Access

Public

Syntax

instancename.**f_GetTextSpecs** (*datastore*, *columnname*, *columninfo*)

Argument	Description
instancename	Instance name of u_html_format
datastore	DataStore containing the column for which specifications are returned (passed by reference)
columnname	String specifying the name of the text field within *datastore*
columninfo	S_column structure into which the function places text field specifications (passed by reference)

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- <0 An error occurred

Usage

Use the output from this function as input to the f_MakeTableHeadCell and f_MakeTableBodyCell functions.

Examples

This example calls the f_GetTextSpecs function:

```
...
inv_format.ist_column = &
   inv_format.f_GetTextSpecs(lds_datastore, &
      "emp_id_t")
...
```

f_InsertHRule

Description

Creates HTML syntax for a horizontal rule across the page.

Access

Public

Syntax

instancename.**f_InsertHRule** ({ *height, color, shadow* {, *width, alignment, clearimage* } })

Argument	Description
instancename	Instance name of u_html_format
height (optional)	Integer specifying line height, in pixels. Specify 0 to take the default line height. If you specify this argument you must also specify *color* and *shadow*
color (optional)	Integer specifying line color. This argument may not work on all browsers. If you specify this argument you must also specify *height* and *shadow*. Specify -999 to use the default line color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
shadow (optional)	Boolean specifying whether the line has a shadow. If you specify this argument you must also specify *height* and *color*
width (optional)	Integer specifying line width, in pixels. Specify 0 to use the default width

Argument	Description
alignment (optional)	Integer specifying alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right
clearimage (optional)	Integer specifying how the table wraps around the horizontal rule. Values are: ♦ 0 Clear all ♦ 1 Clear left ♦ 2 Clear right

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *alignment* was NULL or an invalid value
- -2 *clearimage* was NULL or an invalid value

Usage

Not all arguments work for all browsers.

Examples

This example calls the f_InsertHRule function:

```
...
ls_html += inv_format.f_InsertHRule &
    (0, -5, FALSE)
...
```

f_InsertLineBreak

Description

Creates HTML syntax for a line break.

Access

Public

Syntax

instancename.**f_InsertLineBreak** ({ *clearimage* })

Argument	Description
instancename	Instance name of u_html_format
clearimage (optional)	Integer specifying how text that follows the line break clears an image. Values are: ♦ 0 Clear all ♦ 1 Clear left ♦ 2 Clear right

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *clearimage* is invalid.

Usage

The *clearimage* argument applies to Netscape only.

Examples

This example calls the f_InsertLineBreak function:

```
...
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertLineBreak(0)
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
...
```

f_InsertParagraph

Description

Creates HTML syntax for a Paragraph element.

Access

Public

Syntax

instancename.**f_InsertParagraph** ({ *wordwrap* {, *alignment* {, *clearimage* } } })

Argument	Description
instancename	Instance name of u_html_format
wordwrap (optional)	Boolean specifying whether wordwrap is enabled
alignment (optional)	Integer specifying alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right ♦ 3 Justify

Argument	Description
clearimage (optional)	Integer specifying how text that follows the line break clears an image. Values are: ♦ 0 Clear all ♦ 1 Clear left ♦ 2 Clear right

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *alignment* was NULL or an invalid value
- -2 *clearimage* was NULL or an invalid value

Usage

The arguments apply to Netscape only.

Examples

This example calls the f_InsertParagraph function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph(TRUE, 0, 0)
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
...
```

f_MakeApplet

Description

Creates HTML syntax to embed a Java applet in an HTML page.

Access

Public

Syntax

instancename.**f_MakeApplet** (*object, codeurl, codebaseurl, alternatetext, arguments, alignment, height, width, vmargin, hmargin*)

Argument	Description
instancename	Instance name of u_html_format

Argument	Description
object	String specifying the name of the applet object in the HTML page
codeurl	String specifying the URL for the applet's code file
codebaseurl	String specifying the URL for the applet's working directory
alternatetext	String specifying text to display if the browser doesn't support applets
arguments	Comma-separated string specifying arguments for the applet. Use the format *arg1=val1,arg2=val2*, and so on
alignment	Integer specifying applet alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right
height	Integer specifying the applet height, in pixels
width	Integer specifying the applet width, in pixels
vmargin	Integer specifying the number of blank pixels above and below the applet
hmargin	Integer specifying the number of blank pixels to the left and right of the applet

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *object* was NULL or an empty string
- -2 *codeurl* was NULL or an empty string
- -3 *codebaseurl* was NULL or an empty string
- -4 *alignment* was NULL or an invalid value

Examples

This example calls the f_MakeApplet function:

```
...
ls_html += inv_format.f_MakeApplet&
    ("applettest", "rknTest.class", "/", &
       "Applet won't work here", "", 1, &
          100, 50, 5, 5)
...
```

f_MakeBanner

Description

Creates HTML syntax for a banner. Banners provide a nonscrolling region at the top of a page.

Access

Public

Syntax

instancename.**f_MakeBanner** (*bannertext*)

Argument	Description
instancename	Instance name of u_html_format
bannertext	String specifying the text to display in the nonscrolling region at the top of the page

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_MakeBanner function:

```
String    ls_html
String    ls_temp

ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle &
   ("Test generic u_html_format functions")
ls_html += inv_format.f_EndPageHeading()
ls_html += inv_format.f_MakeBanner &
   ("U_HTML_FORMAT Testing")
...
```

f_MakeBlockQuote

Description

Creates HTML syntax for a BlockQuote element.

Access

Public

Syntax

instancename.**f_MakeBlockQuote** (*quotetext*)

Argument	Description
instancename	Instance name of u_html_format
quotetext	String specifying the text to display in the block quote

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_MakeBlockQuote function:

```
...
inv_format.ist_font.Bold = TRUE
inv_format.ist_font.FontSize = 4
inv_format.ist_font.Italic = TRUE
ls_temp = inv_format.f_FormatText &
   ("The future is wide open.", &
      inv_format.ist_font)
ls_html += inv_format.f_MakeBlockQuote &
   (ls_temp)
...
```

f_MakeEmbed

Description

Creates HTML syntax to for an Embed element. Embedded objects are supported by application-specific plug-ins.

Access

Public

Syntax

instancename.**f_MakeEmbed** (*objectname, objectdata, arguments, palette, height, width*)

Argument	Description
instancename	Instance name of u_html_format
objectname	String specifying the name of the object within the HTML page
objectdata	String specifying the data file for the embedded object
arguments	Comma-separated string specifying arguments for the object. Use the format *arg1=val1,arg2=val2*, and so on
palette	Integer specifying the palette. Values are: ♦ 0 Foreground palette ♦ 1 Background palette
height	Integer specifying the height, in pixels. Specify 0 to omit the HEIGHT attribute from the returned HTML
width	Integer specifying the width, in pixels. Specify 0 to omit the WIDTH attribute from the returned HTML

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *objectname* was NULL or an empty string
- -2 *objectdata* was NULL or an empty string

Examples

This example calls the f_MakeEmbed function:

```
...
ls_html += inv_format.f_MakeEmbed &
    ("WinPlug", "/plgintst.pbd", &
       "WINDOW=w_empbydept", 0, 550, 80)
...
```

f_MakeHeading

Description

Creates HTML syntax for a Heading element, using the specified level.

Access

Public

Syntax

instancename.**f_MakeHeading** (*headingtext*, *level*, *alignment* {, *clearimage*, *wordwrap*, *imageurl*, *verifyimage*, *bulletname*, *sequencenum*, *skipnum* })

Argument	Description
instancename	Instance name of u_html_format
headingtext	String specifying the heading text
level	Integer between 1 and 6 specifying the heading level
alignment	Integer specifying heading alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right ♦ 3 Justify (Netscape only)
clearimage (optional)	Integer specifying how the table wraps around the heading. Values are: ♦ 0 Clear all ♦ 1 Clear left ♦ 2 Clear right
wordwrap (optional)	Boolean specifying whether text wraps in the heading

Argument	Description
imageurl (optional)	String specifying the URL of an image to use as the heading's background
verifyimage (optional)	Boolean indicating whether to verify *imageurl* for accuracy and changed status
bulletname (optional)	String specifying a dingbat bullet to appear before the heading. Specify an empty string to omit the bullet
sequencenum (optional)	Integer specifying the initial number if you are using a numbered or sequenced bullet. Specify 0 to omit this specification
skipnum (optional)	Integer specifying the number of bullets to skip. Specify 0 to omit this specification

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, check ii_errorcode and is_errormsg for more information.

Usage

The value specified for *level* controls the heading's font. The optional arguments are for Netscape only.

Examples

This example calls the f_MakeHeading function:

```
String    ls_html
String    ls_temp

ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle &
   ("Base page for testing u_html_format")
ls_html += inv_format.f_EndPageHeading()
ls_html += inv_format.f_BeginPageBody()
ls_html += inv_format.f_MakeHeading &
   ("Test u_html_format", 1, 1)
...
```

f_MakeImage

Description

Creates HTML syntax to add an image to the page.

Access

Public

Syntax

instancename.**f_MakeImage** (*imageurl, imagecheck* {, *alignment* {, *height, width, borderwidth* {, *vmargin, hmargin* {, *alternatetext* {, *imagemap, mapfileurl* } } } } })

Argument	Description
instancename	Instance name of u_html_format
imageurl	String specifying the URL of the image file
imagecheck	Boolean specifying whether the browser checks to see if the image has changed before downloading it again
alignment	Integer specifying image alignment. Values are: ♦ 0 Left ♦ 1 Center (Internet Explorer only) ♦ 2 Right (Netscape only) ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom
height (optional)	Integer specifying the image height. If you specify this argument you must also specify *width* and *borderwidth*. Specify 0 to omit the HEIGHT attribute from the returned HTML
width (optional)	Integer specifying the image width. If you specify this argument you must also specify *height* and *borderwidth*. Specify 0 to omit the WIDTH attribute from the returned HTML
borderwidth (optional)	Integer specifying the border width, in pixels. If you specify this argument you must also specify *height* and *width*. Specify 0 to omit the BORDER attribute from the returned HTML
vmargin (optional)	Integer specifying the number of blank pixels above and below the image. If you specify this argument you must also specify *hmargin*. Specify 0 to omit the default vertical margin
hmargin (optional)	Integer specifying the number of blank pixels to the left and right of the image. If you specify this argument you must also specify *vmargin*. Specify 0 to omit the default horizontal margin
alternatetext (optional)	String specifying text to display if the browser doesn't support images
imagemap (optional)	Boolean specifying whether the image represents an image map. If you specify this argument you must also specify *mapfileurl*

Argument	Description
mapfileurl (optional)	String specifying the URL of the map file. If you specify this argument you must also specify *imagemap*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *imageurl* or *alignment* are invalid.

Usage

Call this function to create HTML syntax to add an image to the page.

Examples

This example calls the f_MakeImage function:

```
...
ls_html += inv_format.f_MakeImage&
    ("/world.gif", FALSE, 0, 200, 154, 3)
...
```

f_MakeLink

Description

Creates HTML syntax for a hypertext link.

Access

Public

Syntax

instancename.**f_MakeLink** (*linkurl*, *linktext*)

Argument	Description
instancename	Instance name of u_html_format
linkurl	String specifying the URL to jump to
linktext	String specifying the text that users click to activate the link

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs If an error occurs, ii_errorcode contains values as follows:

- -1 *linkurl* was NULL or an empty string
- -2 *linktext* was NULL or an empty string

Usage

The class library also includes an f_MakeLink global function. You use the f_MakeLink global function in DataWindow computed fields to provide hypertext links in the HTML returned by the Object.DataWindow.Data.HTMLTable property.

Examples

This example calls the f_MakeLink function:

```
String    ls_html
```

```
String    ls_temp
...
ls_html += inv_format.f_BeginList(1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_common", &
      "Test common elements")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_netscape", &
      "Test Netscape HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
   ("http://www.powersoft.com", &
      "Go to Powersoft Home Page")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_html += inv_format.f_EndList(1)
...
```

f_MakeListItem

Description

Creates HTML syntax for a list item.

Access

Public

Syntax

instancename.**f_MakeListItem** (*itemtext*, *listtype*)

Argument	Description
instancename	Instance name of u_html_format
itemtext	String specifying the list item's text
listtype	Integer specifying the list type. Values are: ♦ 1 Bulleted list (<LI>) ♦ 2 Numbered list (<LI>) ♦ 3 Definition list (<DD>) ♦ 4 Directory list (<LI>) ♦ 5 Menu list (<LI>)

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *listtype* is invalid.

Examples

This example calls the f_MakeListItem function:

```
String    ls_html
String    ls_temp
...
ls_html += inv_format.f_BeginList(1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_common", &
      "Test common elements")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_netscape", &
      "Test Netscape HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_temp = inv_format.f_MakeLink &
   ("/cgi-bin/pbcgi60.exe/f_explorer", &
      "Test Internet Explorer HTML")
ls_html += inv_format.f_MakeListItem(ls_temp, 1)
ls_html += inv_format.f_EndList(1)
...
```

f_MakeMapArea

Description

Creates HTML syntax for a Map's Area element.

Access

Public

Syntax

instancename.**f_MakeMapArea** (*linkurl*, *coordinates*, *shapetype*, *window*, *ignorelink*)

Argument	Description
instancename	Instance name of u_html_format
linkurl	String specifying the URL to jump to
coordinates	String specifying map coordinates. This specification varies based on the value specified for *shapetype*: ♦ If *shapetype* is 0 (rectangle), specify the x1, y1, x2, and y2 coordinates ♦ If *shapetype* is 1(circle), specify the x and y coordinates of the circle's midpoint, followed by the radius ♦ If *shapetype* is 2 (polygon), specify the x1, y1, x2, y2, x3, and y3 coordinates (that is, specify the successive x, y vertices of the polygon)

Argument	Description
shapetype	Integer specifying the shape of the map area. Values are: ♦ 0 Rectangle ♦ 1 Circle ♦ 2 Polygon
window	Integer specifying the window in which the browser opens the page specified in *linkurl*
ignorelink	Boolean specifying whether the map area should generate links (FALSE) or no links (TRUE)

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *linkurl* was not specified and *ignorelink* was FALSE
- -2 *coordinates* was NULL or an empty string
- -3 *shapetype* was an invalid value

Examples

This example calls the f_MakeMapArea function:

```
...
ls_html += inv_format.f_BeginMapDef("map1")
ls_html += inv_format.f_MakeMapArea &
   ("localhost/", "10,10,30,30", 0, 0, TRUE)
ls_html += inv_format.f_MakeMapArea &
   ("localhost/webpb/", "10,30,30,60", 0, 0, FALSE)
ls_html += inv_format.f_EndMapDef()
ls_html += inv_format.f_makeimage ("/world.gif", &
   TRUE, 1, 200, 164, 2, 3, 3, "World Map", &
      TRUE, "#map1")
...
```

f_MakeMarquee

Description

Creates HTML syntax for a Marquee element.

A Marquee element features text that scrolls across the screen. It provides functionality similar to that found in many simple Java applets. This element is an Internet Explorer extension.

Access

Public

Syntax

instancename.**f_MakeMarquee** (*marqueetext, type, direction, loop scrollamount, scrolldelay, alignment, bgcolor, height, width, vmargin, hmargin*)

Argument	Description
instancename	Instance name of u_html_format
marqueetext	String specifying the text that scrolls across the screen
type	Integer specifying scrolling behavior. Values are: ♦ 0 Scroll across and start over ♦ 1 Scroll across and stop ♦ 2 Bounce back and forth
direction	Integer specifying scrolling direction. Values are: ♦ 0 Right-to-left ♦ 1 Left-to-right
loop	Integer specifying the number of times *marqueetext* scrolls across the screen. Specify -1 for an infinite loop
scrollamount	Integer specifying the number of pixels *marqueetext* moves with each scrolling action. Specify 0 to use the default scroll amount
scrolldelay	Integer specifying the number of milliseconds to wait before scrolling. Specify 0 to use the default scroll delay
alignment	Integer specifying text alignment within the marquee. Values are: ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom
bgcolor	Long specifying the background color. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
height	Integer specifying marquee height, in pixels. Specify 0 to omit the HEIGHT attribute from the returned HTML
width	Integer specifying marquee width, as a percentage of screen width. Specify 0 to omit the WIDTH attribute from the returned HTML
vmargin	Integer specifying the number of blank pixels above and below the marquee. Specify 0 to use the default vertical margin

Argument	Description
hmargin	Integer specifying the number of blank pixels to the left and right of the marquee. Specify 0 to use the default horizontal margin

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *alignment* is invalid.

Examples

This example calls the f_MakeMarquee function:

```
...
ls_html += inv_format.f_MakeMarquee &
    ("Welcome to the Internet Explorer Test page", &
       0, 0, 100, 5, 5, 3, -12, 30, 500, 5, 5)
...
```

f_MakeObject

Description

Creates HTML syntax for an Object element, allowing you to embed ActiveX controls in a page.

Access

Public

Syntax

instancename.**f_MakeObject** (*object*, *classid*, *arguments*, *waitingmessage*, *alignment*, *borderwidth*, *height*, *width*, *vmargin*, *hmargin* {, *codebaseurl*, *codetype*, *dataurl*, *datatype*, *declareonly* })

Argument	Description
instancename	Instance name of u_html_format
object	String specifying the name of the object in the HTML page
classid	String specifying the class ID for the ActiveX control or Java class
arguments	String specifying arguments and values to be included in the Object element
waitingmessage	String specifying the message to display while the object downloads

Argument	Description
alignment	Integer specifying object alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right ♦ 9 Baseline ♦ 12 TextTop ♦ 13 TextMiddle ♦ 14 TextBottom
borderwidth	Integer specifying the border width, in pixels
height	Integer specifying the object height, in pixels
width	Integer specifying the object width, in pixels
vmargin	Integer specifying the number of blank pixels above and below the object
hmargin	Integer specifying the number of blank pixels to the left and right of the object
codebaseurl (optional)	String specifying the URL for the object's working directory
codetype (optional)	String specifying the ActiveX class or Java Class
dataurl (optional)	String specifying the URL for the object's data file
datatype (optional)	String specifying the Internet media type
declareonly (optional)	Boolean indicating whether you are declaring this object for another object's use (TRUE) or it can stand alone (FALSE)

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *object* was NULL or an empty string
- -2 *classid* was NULL or an empty string
- -3 *codebaseurl* was NULL or an empty string
- -4 *codetype* was NULL or an empty string
- -5 *dataurl* was NULL or an empty string
- -6 *datatype* was NULL or an empty string

Usage

Call this function to create the HTML syntax for an Object element, using the Internet Explorer 3.0 HTML extension.

Examples

This example calls the f_MakeObject function:

```
STRING ls_html
...
ls_html += &
    "<PARAM NAME=~"defaultfrstart~" value=~"0~">"
ls_html += &
    "<PARAM NAME=~"defaultfrend~" value=~"7~">"
ls_html += &
    "<PARAM NAME=~"mouseoverfrstart~" value=~"8~">"
ls_html += &
    "<PARAM NAME=~"mouseoverfrend~" value=~"15~">"
ls_html += &
    "<PARAM NAME=~"focusfrstart~" value=~"16~">"
ls_html += &
    "<PARAM NAME=~"focusfrend~" value=~"23~">"
ls_html += &
    "<PARAM NAME=~"downfrstart~" value=~"24~">"
ls_html += &
    "<PARAM NAME=~"downfrend~" value=~"34~">"
ls_html += &
    "<PARAM NAME=~"URL~"" &
       + "value=~"http://www.mycompany.com/my.avi~">"

mle_html2.text += inv_format.f_MakeObject &
    ("abutton", &
       "clsid:0482B100-739C-11CF-A3A9-00A0C9034920", &
          ls_html, "Waiting for ActiveX to Load", &
```

```
1, 2, 200, 300, 0, 0, "", "", "", &
   "", FALSE )
```

f_MakeTableBodyCell

Creates HTML syntax for a cell within the body of a table. There are four syntaxes:

To	Use
Create a table body cell for all browsers	Syntax 1
Create a table body cell using Netscape extensions	Syntax 2
Create a table body cell using Internet Explorer extensions	Syntax 3
Create a table body cell using Netscape and Internet Explorer extensions	Syntax 4

Syntax 1 — Create a table body cell for all browsers

Description: Creates HTML syntax for a cell within the body of a table.

Access: Public

Syntax: *instancename*.**f_MakeTableBodyCell** (*celltext*, *halignment*, *valignment*, *columnspan*, *rowspan*, *wrap*)

Argument	Description
instancename	Instance name of u_html_format
celltext	String specifying the text displayed in the table cell
halignment	Integer specifying horizontal cell alignment. Values are: ♦ 0 Left ♦ 1 Center (Internet Explorer only) ♦ 2 Right ♦ 7 Middle (Netscape only)
valignment	Integer specifying vertical cell alignment. Values are: ♦ 7 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
columnspan	Integer specifying the number of columns occupied by the cell

Argument	Description
rowspan	Integer specifying the number of rows occupied by the cell
wrap	Boolean specifying whether word wrap is enabled for the cell

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halignment* or *valignment* are invalid.

Usage

Call this function to create the HTML syntax for a cell in the body of a table. This version of the function generates generic syntax for all browsers.

Use Syntax 2 to create a table cell using Netscape extensions; use Syntax 3 to create a table cell using Internet Explorer extensions; use Syntax 4 to create a table cell using Netscape and Internet Explorer extensions.

Examples

This example calls the f_MakeTableBodyCell function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE)
ls_html += inv_format.f_EndTableRow()
ls_html += inv_format.f_EndTable()
...
```

Syntax 2

Create a cell using Netscape extensions

Description

Creates HTML syntax for a cell in the body of a table using Netscape extensions.

Access

Public

Syntax

instancename.**f_MakeTableBodyCell** (*celltext*, *halignment*, *valignment*, *columnspan*, *rowspan*, *wrap*, *width*)

Argument	Description
instancename	Instance name of u_html_format
celltext	String specifying the text displayed in the table cell

Argument	Description
halignment	Integer specifying horizontal cell alignment. Values are: ♦ 0 Left ♦ 2 Right ♦ 7 Middle
valignment	Integer specifying vertical cell alignment. Values are: ♦ 7 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
columnspan	Integer specifying the number of columns occupied by the cell
rowspan	Integer specifying the number of rows occupied by the cell
wrap	Boolean specifying whether word wrap is enabled for the cell
width	Integer specifying table width, as a percentage of screen width

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halignment* or *valignment* are invalid.

Usage

Call this function to create the HTML syntax for a table body cell when you want to use Netscape table extensions.

Use Syntax 1 to create a table cell using generic HTML; use Syntax 3 to create a table cell using Internet Explorer extensions; use Syntax 4 to create a table cell using Netscape and Internet Explorer extensions.

Examples

This example calls the f_MakeTableBodyCell function:

```
...
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE, 90)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE, 90)
ls_html += inv_format.f_EndTableRow()
ls_html += inv_format.f_EndTable()
...
```

Syntax 3 **Create a cell using Internet Explorer extensions**

Description

Creates HTML syntax for a cell in the body of a table using Internet Explorer extensions

Access

Public

Syntax

instancename.**f_MakeTableBodyCell** (*celltext, halignment, valignment, columnspan, rowspan, wrap, bgimageurl, bgcolor, bordercolor, bordercolorlight, bordercolordark*)

Argument	Description
instancename	Instance name of u_html_format
celltext	String specifying the text displayed in the table cell
halignment	Integer specifying horizontal cell alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right
valignment	Integer specifying vertical cell alignment. Values are: ♦ 7 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
columnspan	Integer specifying the number of columns occupied by the cell
rowspan	Integer specifying the number of rows occupied by the cell
wrap	Boolean specifying whether word wrap is enabled for the cell
bgimageurl	String specifying the URL of an image to use as the cell's background
bgcolor	Long specifying the cell background color. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
bordercolor	Long specifying a border color for the cell. Specify -999 to use the default border color

Argument	Description
bordercolorlight	Long specifying the cell border's highlight color; used to make a 3D table border. Specify -999 to omit the BORDERCOLORLIGHT attribute from the returned HTML
bordercolordark	Long specifying the cell border's shadow color; used to make a 3D table border. Specify -999 to omit the BORDERCOLORDARK attribute from the returned HTML

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halignment* or *valignment* are invalid.

Usage

Call this function to create the HTML syntax for a table body cell when you want to use Internet Explorer table extensions.

Use Syntax 1 to create a table cell using generic HTML; use Syntax 2 to create a table cell using Netscape extensions; use Syntax 4 to create a table cell using Netscape and Internet Explorer extensions.

Examples

This example calls the f_MakeTableBodyCell function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12, 2, "", -1, -12, -11, 2, 2)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE, "", -7, -14, &
      -12, -11)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE, "", -7, -14, -12, -11)
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE, "", -3, &
      -14, -12, -11)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE, "", -3, -14, -12, &
      -11)
ls_html += inv_format.f_EndTableRow()
...
```

Syntax 4 **Create a cell for Netscape and Internet Explorer**

Description

Creates HTML syntax for a table cell using a set of extensions supported by both Netscape and Internet Explorer.

Access

Public

Syntax

instancename.**f_MakeTableBodyCell** (*columnspec*, *celltext*)

Argument	Description
instancename	Instance name of u_html_format
columnspec	S_column structure containing all column attributes
celltext	String specifying the text displayed in the cell

Return value

String. Returns HTML syntax if the function succeeds and an empty string if ist_column.alignment or ist_column.valignment are invalid.

Usage

Call this function to create the HTML syntax for a cell in a table row. This version of the function generates syntax for Netscape and Internet Explorer.

Initialize the s_column structure with column specifications before calling this function. To initialize the s_column structure, call f_GetColSpecs, f_GetCompSpecs, or f_GetTextSpecs, as appropriate.

Use Syntax 1 to create a table cell using generic HTML; use Syntax 2 to create a table cell using Netscape extensions; use Syntax 3 to create a table cell using Internet Explorer extensions.

Examples

This example calls the f_MakeTableBodyCell function:

```
...
inv_format.f_GetColSpecs(lds_data, "emp_id", &
   inv_format.ist_column)
ls_html += inv_format.f_MakeTableBodyCell &
   (inv_format.ist_column, "Bill Jones")
...
```

f_MakeTableHeadCell

Creates HTML syntax for a cell within the heading of a table. There are four syntaxes:

To	Use
Create a table head cell for all browsers	Syntax 1

To	Use
Create a table head cell using Netscape extensions	Syntax 2
Create a table head cell using Internet Explorer extensions	Syntax 3
Create a table head cell using Netscape and Internet Explorer extensions	Syntax 4

Syntax 1 **Create a table head cell for all browsers**

Description

Creates HTML syntax for a cell within the heading of a table.

Access

Public

Syntax

instancename.**f_MakeTableHeadCell** (*celltext*, *halignment*, *valignment*, *columnspan*, *rowspan*, *wrap*)

Argument	Description
instancename	Instance name of u_html_format
celltext	String specifying the text displayed in the table cell
halignment	Integer specifying horizontal cell alignment. Values are: ♦ 0 Left ♦ 1 Center (Internet Explorer only) ♦ 2 Right ♦ 7 Middle (Netscape only)
valignment	Integer specifying vertical cell alignment. Values are: ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
columnspan	Integer specifying the number of columns occupied by the cell
rowspan	Integer specifying the number of rows occupied by the cell
wrap	Boolean specifying whether word wrap is enabled for the cell

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halignment* or *valignment* are invalid.

Usage

Call this function to create the HTML syntax for a cell in the heading of a table. This version of the function generates generic syntax for all browsers.

Use Syntax 2 to create a table cell using Netscape extensions; use Syntax 3 to create a table cell using Internet Explorer extensions; use Syntax 4 to create a table cell using Netscape and Internet Explorer extensions.

Examples

This example calls the f_MakeTableHeadCell function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE)
...
```

Syntax 2

Create a table using Netscape extensions

Description

Creates HTML syntax for a cell in the heading of a table using Netscape extensions.

Access

Public

Syntax

instancename.**f_MakeTableHeadCell** (*celltext*, *halignment*, *valignment*, *columnspan*, *rowspan*, *wrap*, *width*)

Argument	Description
instancename	Instance name of u_html_format
celltext	String specifying the text displayed in the table cell
halignment	Integer specifying horizontal cell alignment. Values are: ♦ 0 Left ♦ 2 Right ♦ 7 Middle
valignment	Integer specifying vertical cell alignment. Values are: ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
columnspan	Integer specifying the number of columns occupied by the cell
rowspan	Integer specifying the number of rows occupied by the cell

Argument	Description
wrap	Boolean specifying whether word wrap is enabled for the cell
width	Integer specifying the table width, as a percentage of the screen width

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halignment* or *valignment* are invalid.

Usage

Call this function to create the HTML syntax to create a table heading cell when you want to use Netscape table extensions.

Use Syntax 1 to create a table cell using generic HTML; use Syntax 3 to create a table cell using Internet Explorer extensions; use Syntax 4 to create a table cell using Netscape and Internet Explorer extensions.

Examples

This example calls the f_MakeTableHeadCell function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12, TRUE, TRUE, 0)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE, 90)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE, 90)
...
```

Syntax 3

Create a table using Internet Explorer extensions

Description

Creates HTML syntax for a cell in the heading of a table using Internet Explorer extensions.

Access

Public

Syntax

instancename.**f_MakeTableHeadCell** (*celltext, halignment, valignment, columnspan, rowspan, wrap, bgimageurl, bgcolor, bordercolor, bordercolorlight, bordercolordark*)

Argument	Description
instancename	Instance name of u_html_format
celltext	String specifying the text displayed in the table cell

Argument	Description
halignment	Integer specifying horizontal cell alignment. Values are: ♦ 0 Left ♦ 1 Center ♦ 2 Right
valignment	Integer specifying vertical cell alignment. Values are: ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom ♦ 9 Baseline
columnspan	Integer specifying the number of columns occupied by the cell
rowspan	Integer specifying the number of rows occupied by the cell
wrap	Boolean specifying whether word wrap is enabled for the cell
bgimageurl	String specifying the URL of an image to use as the cell's background
bgcolor	Long specifying the cell background color. Specify -999 to use the default background color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220
bordercolor	Long specifying a border color for the cell. Specify -999 to use the default border color
bordercolorlight	Long specifying the cell border's highlight color; used to make a 3D table border. Specify -999 to omit the BORDERCOLORLIGHT attribute from the returned HTML
bordercolordark	Long specifying the cell border's shadow color; used to make a 3D table border. Specify -999 to omit the BORDERCOLORDARK attribute from the returned HTML

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *halignment* or *valignment* are invalid.

Usage

Call this function to create the HTML syntax for a table heading cell when you want to use Internet Explorer table extensions.

Use Syntax 1 to create a table cell using generic HTML; use Syntax 2 to create a table cell using Netscape extensions; use Syntax 4 to create a table cell using Netscape and Internet Explorer extensions.

Examples

This example calls the f_MakeTableHeadCell function:

```
...
ls_html += inv_format.f_BeginTable(0, 10, 3, 3, &
   75, -12, 2, "", -1, -12, -11, 2, 2)
ls_html += inv_format.f_BeginTableHead()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Writer", 0, 4, 1, 1, TRUE, "", -7, -14, &
      -12, -11)
ls_html += inv_format.f_MakeTableHeadCell &
   ("Book", 0, 1, 1, 1, TRUE, "", -7, -14, -12, -11)
ls_html += inv_format.f_BeginTableBody()
ls_html += inv_format.f_BeginTableRow(0, 1)
ls_html += inv_format.f_MakeTableBodyCell &
   ("Bill Jones", 0, 5, 1, 2, TRUE, "", -3, &
      -14, -12, -11)
ls_temp = "Object Reference"
ls_temp += inv_format.f_InsertParagraph()
ls_temp += "User's Guide"
ls_html += inv_format.f_MakeTableBodyCell &
   (ls_temp, 0, 5, 1, 2, TRUE, "", -3, -14, -12, &
      -11)
ls_html += inv_format.f_EndTableRow()
...
```

Syntax 4

Create a cell for Netscape and Internet Explorer

Description

Creates HTML syntax for a table cell using a set of extensions supported by both Netscape and Internet Explorer.

Access

Public

Syntax

instancename.**f_MakeTableHeadCell** (*columnspec, celltext*)

Argument	Description
instancename	Instance name of u_html_format
columnspec	S_column structure containing all column attributes
celltext	String specifying the text displayed in the cell

Return value

String. Returns HTML syntax if the function succeeds and an empty string if ist_column.alignment or ist_column.valignment are invalid.

Usage

Call this function to create the HTML syntax for a cell in a table heading. This version of the function generates syntax for Netscape and Internet Explorer.

Initialize the s_column structure with column specifications before calling this function. To initialize the s_column structure, call f_GetColSpecs, f_GetCompSpecs, or f_GetTextSpecs, as appropriate.

Use Syntax 1 to create a table cell using generic HTML; use Syntax 2 to create a table cell using Netscape extensions; use Syntax 3 to create a table cell using Internet Explorer extensions.

Examples

This example calls the f_MakeTableHeadCell function:

```
...
inv_format.f_GetColSpecs(lds_data, "emp_id_t", &
    inv_format.ist_column)
ls_html += inv_format.f_MakeTableHeadCell &
    (inv_format.ist_column, "Emp ID")
...
```

f_MakeVideoClip

Description

Creates HTML for a video clip in Internet Explorer.

Access

Public

Syntax

instancename.**f_MakeVideoClip** (*imageurl, alternatetext, displaycontrols, loops, startevent, imagemap, mapfileurl, alignment, borderwidth, height, width, vmargin, hmargin*)

Argument	Description
instancename	Instance name of u_html_format
imageurl	String specifying the URL of the image file
alternatetext	String specifying text to display if the browser doesn't support images
displaycontrols	Boolean indicating whether to display controls (start, stop, pause) below the image
loops	Integer specifying the number of times to loop through the animation. Specify -1 for an infinite loop

Argument	Description
startevent	Integer specifying the event that causes the animation to start. Values are: ♦ 0 FileOpen ♦ 1 MouseOver
imagemap	Boolean specifying whether the image represents an image map
mapfileurl	String specifying the URL of the map file
alignment	Integer specifying image alignment. Values are: ♦ 0 Left ♦ 1 Right ♦ 6 Top ♦ 7 Middle ♦ 8 Bottom
borderwidth	Integer specifying the border width, in pixels. Specify 0 to use the default border width
height	Integer specifying the image height, in pixels. Specify 0 to use the default height
width	Integer specifying the image width, in pixels. Specify 0 to use the default width
vmargin	Integer specifying the number of blank pixels above and below the image. Specify 0 to use the default vertical margin
hmargin	Integer specifying the number of blank pixels to the left and right of the image. Specify 0 to use the default horizontal margin

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *imageurl* or *alignment* are invalid.

Examples

This example calls the f_MakeVideoClip function:

```
String ls_html
...
ls_html+= inv_format.f_MakeVideoClip &
    ("/avi/cubespin.avi", "Spinning Cube", &
       TRUE, 1, 1, FALSE, "", 1, 1, 150, 150, 3, 3)
...
```

f_RedirectClient

Description

Returns the user to a specified URL.

Access

Public

Syntax

instancename.**f_RedirectClient** (*redirecturl*)

Argument	Description
instancename	Instance name of u_html_format
redirecturl	String containing the URL to be accessed and displayed on the client's browser

Return value

Blob. Returns HTML syntax prefixed by the text/html content type if the function succeeds and a blob containing an empty string if *redirecturl* is null or empty.

Examples

This example, which returns a blob, calls the f_RedirectClient function. It assumes an ib_firsttime boolean instance variable:

```
Blob   lblb_return

IF ib_firsttime THEN
   ib_firsttime = FALSE
   RETURN(inv_format.f_RedirectClient &
      ("http://www.powersoft.com/"))
ELSE
   SetNull(lblb_return)
   Return(lblb_return)
END IF
```

f_ReturnHTMLText

Description

Prefixes the text/html content type (also called MIME type) to the passed HTML text and returns it as a blob.

Access

Public

Syntax

instancename.**f_ReturnHTMLText** (*htmltext*)

Argument	Description
instancename	Instance name of u_html_format

Argument	Description
htmltext	String containing HTML to which the function prefixes the text/html content type

Return value

Blob. Returns HTML syntax prefixed by the text/html content type if the function succeeds and a blob containing an empty string if *htmltext* is null or empty.

Usage

For functions returning a string, Web.PB adds the text/html prefix automatically. In most cases you don't need to call this function.

Examples

This example, which returns a blob, calls the f_ReturnHTMLText function. It assumes an ib_firsttime boolean instance variable:

```
Blob      lblb_return
String    ls_html

IF ib_firsttime THEN
   ib_firsttime = FALSE
   ls_html = inv_format.f_BeginPage()
   ls_html += inv_format.f_BeginPageBody()
   ls_html += inv_format.f_MakeHeading &
      ("Sample page", 2, 1)
   ls_html += inv_format.f_EndPageBody()
   ls_html += inv_format.f_EndPage()
   RETURN(inv_format.f_ReturnHTMLText(ls_html))
ELSE
   SetNull(lblb_return)
   RETURN(lblb_return)
END IF
```

f_ReturnNonText

Description

Prefixes the specified content type (also called MIME type) to the passed data and returns it as a blob.

Access

Public

Syntax

instancename.**f_ReturnNonText** (*mimedata*, *mimetype*)

Argument	Description
instancename	Instance name of u_html_format

Argument	Description
mimedata	Blob containing the data to be passed. This argument is passed by reference
mimetype	String specifying the content type

Return value

Blob. Returns the specified data if the function succeeds and a blob containing an empty string if *mimedata* or *mimetype* are invalid.

Usage

Call this function to return nontextual data (such as graphics) to the user. This function can return blobs smaller than 32K.

Examples

This example, which returns a blob, calls the f_ReturnNonText function:

```
Blob      lblb_blob
Boolean   lb_fileexists
String    ls_mimetype, ls_picture
String    ls_imagedir
Long      flen, fnum, ll_bytesread

ls_imagedir = "c:\pwrs\pb5i32\"
ls_picture = "header.jpg"

// Return a null blob the last time
IF ib_lasttime = TRUE THEN
   setNull(lblb_blob)
   RETURN lblb_blob
END IF

ls_picture = ls_imagedir + ls_picture
lb_fileexists = FileExists(ls_picture)
IF NOT lb_fileexists THEN
   ib_lasttime = TRUE
   RETURN inv_format.f_ReturnPlainText &
      (ls_picture + " does not exist")
END IF

flen = FileLength(ls_picture)
fnum = FileOpen(ls_picture, StreamMode!, Read!)
IF fnum = -1 THEN
   ib_lasttime = TRUE
   RETURN inv_format.f_ReturnPlainText &
      (ls_picture + " could not be opened")
END IF
```

```
ls_mimetype = "image/jpeg~r~n"
ls_mimetype += " content-length: " + String(flen)

ll_bytesread = FileRead(fnum, lblb_blob)
FileClose(fnum)
ib_lasttime = TRUE

RETURN inv_format.f_ReturnNonText &
    (lblb_blob, ls_mimetype)
```

f_ReturnPlainText

Description

Prefixes the text/plain content type (also called MIME type) to the passed text and returns it as a blob.

Access

Public

Syntax

instancename.**f_ReturnPlainText** (*plaintext*)

Argument	Description
instancename	Instance name of u_html_format
plaintext	String containing unformatted text to which the function prefixes the text/plain content type

Return value

Blob. Returns unformatted text if the function succeeds and a blob containing an empty string if *plaintext* is null or empty.

Usage

Call this function to return plain text to the user.

Examples

This example, which returns a blob, calls the f_ReturnPlainText function. It assumes an ib_firsttime boolean instance variable:

```
Blob      lblb_return
String    ls_text

IF ib_firsttime THEN
    ib_firsttime = FALSE
    ls_text = "This is a sample of plain text"
    RETURN(inv_format.f_ReturnPlainText(ls_text))
ELSE
    SetNull(lblb_return)
```

```
    RETURN(lblb_return)
END IF
```

f_SetBaseFont

Description

Creates HTML for the BaseFont element, which specifies one of seven predefined font sizes for the page.

Access

Public

Syntax

instancename.**f_SetBaseFont** (*fontsize, fontname, fontcolor*)

Argument	Description
instancename	Instance name of u_html_format
fontsize	Integer between 1 and 7 specifying the font size
fontname	String specifying the font name. Specify an empty string to use the default font name
fontcolor	Long specifying the font color. Specify -999 to use the default font color FOR INFO For a list of color specifications, see f_BeginPageBody on page 220

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Usage

The *fontname* and *fontcolor* arguments apply to Internet Explorer only.

Examples

This example calls the f_SetBaseFont function:

```
String    ls_html

ls_html = inv_format.f_BeginPage()

ls_html += inv_Format.f_BeginPageHeading()
ls_html += inv_Format.f_SetPageTitle &
   ("Web.PB Class Library Example")
ls_html += inv_Format.f_SetBaseFont(2, "Arial", -1)
ls_html += inv_Format.f_EndPageHeading()
...
```

f_SetBaseURL

Description

Creates HTML for the Base element, which specifies the absolute URL base to be used for any relative URL links in the page.

Access

Public

Syntax

instancename.**f_SetBaseURL** (*baseurl, window*)

Argument	Description
instancename	Instance name of u_html_format
baseurl	String specifying the base URL for the page
window	Integer specifying the default window for the page. Values are: ♦ 0 Blank ♦ 1 Parent ♦ 2 Self ♦ 3 Top

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_SetBaseURL function:

```
...
ls_html += inv_format.f_SetBaseURL&
    ("http://www.powersoft.com/userconf97/docs", 2)
...
```

f_SetBGSound

Description

Creates HTML for the BgSound element, which specifies a soundtrack for an HTML page.

Access

Public

Syntax

instancename.**f_SetBGSound** (*soundurl, loop*)

Argument	Description
instancename	Instance name of u_html_format
soundurl	String specifying the URL of the AU, WAV, or MIDI file to play

Argument	Description
loop	Integer specifying the number of times to loop through *soundurl*. Specify -1 for an infinite loop

Return value

String. Returns HTML syntax if the function succeeds and an empty string if *soundurl* is null or an empty string.

Usage

This function works with Internet Explorer only.

Examples

This example calls the f_SetBGSound function:

```
...
ls_html += inv_format.f_BeginPageBody()
ls_html += inv_format.f_SetBGSound &
    ("/tada.wav", 4)
...
```

f_SetPageTitle

Description

Creates HTML syntax for the Title element. This element specifies the text displayed on the title bar of the browser.

Access

Public

Syntax

instancename.**f_SetPageTitle** (*titletext*)

Argument	Description
instancename	Instance name of u_html_format
titletext	String specifying the title text

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs.

Examples

This example calls the f_SetPageTitle function:

```
String    ls_html
String    ls_temp
ls_html += inv_format.f_BeginPage()
ls_html += inv_format.f_BeginPageHeading()
ls_html += inv_format.f_SetPageTitle&
   ("Base page for testing u_html_format")
ls_html += inv_format.f_EndPageHeading()
...
```

u_html_template

Description

HTML template service. This object contains functions to read a template file and make replacements to predefined substitution text.

Ancestry

Inherits from NonVisualObject.

Library

Webpb.pbl

Usage

To use u_html_template:

1 Define an HTML template file with HTML elements and substitution parameters (prefixed by two ampersands in this example):

```
<HTML>
<HEAD>
<TITLE>Acme Corporation</TITLE>
<BASEFONT SIZE=3 COLOR=BLACK FACE="ARIAL">
</HEAD>

<BODY BGCOLOR=TEAL>
<H3>AcmeWeb:Enterprise</H3>
<P>Records &&Start to &&End of &&Total
<P>&&Data
</HTML>
```

2 Declare an instance variable in your user object of type u_html_template:

```
u_html_template   inv_template
```

This user object uses PowerBuilder's autoinstantiate feature, so there is no need to code a CREATE statement.

3 Call the f_OpenTemplate function to access your template file:

```
String    ls_html

ls_html = inv_template.f_OpenTemplate &
   ("c:\webapp\templates\formtemp.htm")
```

4 Call the f_Replace and f_ReplaceAll functions as necessary to customize the template:

```
ls_html = inv_template.f_Replace &
   (ls_html, "&&Start", "1")
ls_html = inv_template.f_Replace &
   (ls_html, "&&End", "10")
```

```
ls_html = inv_template.f_Replace &
   (ls_html, "&&Total", "87")
ls_html = inv_template.f_Replace &
   (ls_html, "&&Data", &
lds_page.Object.DataWindow.Data.HtmlTable)
```

5 Return the template's HTML string:

```
RETURN ls_html
```

See also

u_html_form
u_html_format

Instance variables

U_html_template includes instance variables:

Instance variable	Description	Data type	Access	Usage
ii_errorcode	Contains an error code for the last u_html_template function	Integer	Public	If a u_html_template function returns an empty string, use this instance variable to access high-level error information
is_documentdir	Document directory	String	Public	The f_SetEnvironment function sets this instance variable
is_errormsg	Contains an error description for the last u_html_template function	String	Public	If a u_html_template function returns an empty string, use this instance variable to access detailed error information
is_templatedir	Template directory	String	Public	The f_SetEnvironment function sets this instance variable

Functions

U_html_template includes precoded object functions:

f_OpenTemplate	f_ReplaceAll
f_Replace	f_SetEnvironment

f_OpenTemplate

Description

Returns the specified template file into a string variable.

Access

Public

Syntax

instancename.**f_OpenTemplate** (*filename*)

Argument	Description
instancename	Instance name of u_html_template
filename	String specifying the DOS filename of the HTML template file. If you don't specify a fully concatenated filename, specify path information using the f_SetEnvironment function

Return value

String. Returns the HTML template file if the function succeeds and an empty string if *filename* could not be opened.

Usage

Call this function to open an HTML template file.

Examples

This example calls the f_OpenTemplate function (it assumes that you haven't called f_SetEnvironment previously):

```
String    ls_html

ls_html = inv_template.f_OpenTemplate &
   ("c:\webapp\templates\formtemp.htm")
...
```

f_Replace

Description

Replaces the first occurrence of the substitution parameter with the replace string.

Access

Public

Syntax

instancename.**f_Replace** (*htmlsyntax, subparm, replacevalue*)

Argument	Description
instancename	Instance name of u_html_template

Argument	Description
htmlsyntax	String containing the HTML template string
subparm	String specifying the substitution parameter to be replaced
replacevalue	String containing the text to replace the first occurrence of *subparm*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *htmlsyntax* was NULL or an empty string
- -2 *subparm* was NULL or an empty string
- -3 *replacevalue* was NULL or an empty string
- -4 *replacevalue* could not be found within *htmlsyntax*

Usage

Call this function to replace a single substitution parameter.

Examples

This example calls the f_Replace function:

```
String    ls_html
DataStore   lds_page
...
ls_html = inv_template.f_Replace &
    (ls_html, "&&Start", "1")
ls_html = inv_template.f_Replace &
    (ls_html, "&&End", "10")
ls_html = inv_template.f_Replace &
    (ls_html, "&&Total", "87")
ls_html = inv_template.f_Replace &
    (ls_html, "&&Data", &
       lds_page.Object.DataWindow.Data.HtmlTable)
...
```

f_ReplaceAll

Description

Replaces all occurrences of the substitution parameter with the replace string.

Access

Public

Syntax

instancename.**f_ReplaceAll** (*htmlsyntax*, *subparm*, *replacevalue*)

Argument	Description
instancename	Instance name of u_html_template
htmlsyntax	String containing the HTML template string
subparm	String specifying the substitution parameter to be replaced
replacevalue	String containing the text to replace all occurrences of *subparm*

Return value

String. Returns HTML syntax if the function succeeds and an empty string if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *htmlsyntax* was NULL or an empty string
- -2 *subparm* was NULL or an empty string
- -3 *replacevalue* was NULL or an empty string
- -4 *replacevalue* could not be found within *htmlsyntax*

Examples

This example calls the f_ReplaceAll function:

```
String    ls_html
...
ls_html = inv_template.f_ReplaceAll &
   (ls_html, "&&ClientName", &
      "First State Bank")
...
```

f_SetEnvironment

Description

Establishes the document and template directories.

Access

Public

Syntax

instancename.**f_SetEnvironment** (*documentdir*, *templatedir*)

Argument	Description
instancename	Instance name of u_html_template
documentdir	String specifying the fully qualified pathname of the document directory containing template HTML files. If template HTML files are stored in a template directory below *documentdir*, specify *templatedir* also
templatedir	String specifying the pathname of the template directory. Specify only the portion of the pathname below *documentdir*

Return value

None.

Usage

Call this function to establish the document and template directories. If you specify these directories, you don't need to specify a full pathname when calling the f_OpenTemplate function.

Examples

This example calls the f_SetEnvironment function:

```
...
inv_template.f_SetEnvironment &
    ("c:\Program Files\WebSite\htdocs", &
       "templates")
...
```

u_session

Description

Session management service. This object contains functions you call to establish a session, which subsequent user object functions can reference to obtain user, state, and transaction information.

This object updates the www_session and www_session_argument tables, found in the webpb database, distributed with the Internet Tools.

FOR INFO For complete information on using u_session, see "Managing distributed sessions" on page 179.

Ancestry

Inherits from NonVisualObject.

Library

Webpb.pbl

Usage

To use u_session:

1 Declare an instance variable in your user object of type u_session:

```
u_session   inv_session
```

This user object uses PowerBuilder's autoinstantiate feature, so there is no need to code a CREATE statement.

2 Create a session ID by calling the f_GenerateID function:

```
Long   ll_sessionid

ll_sessionid = inv_session.f_GenerateID()
```

3 Add the session to the database by calling the f_NewSession function, passing a transaction object, the user's IP address, and the session ID:

```
inv_session.f_NewSession(SQLCA, remote_addr, &
   ll_sessionid)
```

Before calling this function you must initialize transaction object fields and connect to the database. Typically you initialize and connect either in the server Application's ConnectionBegin event or at the beginning of the user object function.

Accessing the user's IP address
To access the user's IP address, pass the remote_addr argument to your user object function. Remote_addr is a CGI environment variable that contains the user's IP address.

4 Check the ii_errorcode instance variable to see if the function succeeded:

```
IF inv_session.ii_errorcode <> 0 THEN
   RETURN("Error creating session ID.")
END IF
```

5 Save session arguments as necessary:

```
inv_session.f_SetArgumentValue &
   (SQLCA, ll_sessionid, "userid", as_userid)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN(inv_session.is_errormsg)
END IF
inv_session.f_SetArgumentValue &
   (SQLCA, ll_sessionid, "password", as_userpass)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN(inv_session.is_errormsg)
END IF
```

6 In the HTML that the function returns, add a hidden field containing the session ID:

```
ls_sessionid = &
   inv_html_form.f_MakeHidden &
      ("as_sessionid", ll_sessionid)
```

7 Include a session ID argument (as_sessionid in this example) in the user object function named in the displayed form's ACTION attribute.

8 Use the session ID to call u_session functions in subsequent user object functions as necessary to access and maintain session information.

See also

u_transaction

Instance variables

U_session includes instance variables:

Instance variable	Description	Data type	Access	Usage
ii_errorcode	Contains an error code for the last u_session function	Integer	Public	After calling certain u_session functions, use this instance variable to determine whether an error occurred

Instance variable	Description	Data type	Access	Usage
is_errormsg	Contains an error description	String	Public	If an error occurs, use this instance variable to access detailed error information

Functions

U_session includes precoded object functions:

f_CleanUpSessions	f_SetArgumentValue
f_GenerateID	f_UpdateLastAccess
f_GetArgumentValue	f_VerifySessionID
f_NewSession	

f_CleanUpSessions

Description

Deletes all sessions before the specified date and time. This function deletes rows from the www_session and www_session_argument tables.

Access

Public

Syntax

instancename.**f_CleanUpSessions** (*transaction, enddatetime*)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
enddatetime	DateTime variable before which the function deletes sessions

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- -1 *transaction* was invalid
- -2 *enddatetime* was NULL
- -3 Error declaring the cursor
- -4 Error opening the cursor
- -5 Error fetching the cursor
- -6 Error closing the cursor
- -8 Error deleting rows in www_session_argument
- -9 Error deleting rows in www_session

Usage

Call this function to remove old sessions from the database.

You typically call this function on a regular basis from the application server's console application.

Removing old sessions
What constitutes an old session is a site-specific consideration. Depending on your application, 10 minutes of inactivity might signal an inactive session. In other cases, 60 minutes of inactivity might signal an inactive session.

Examples

This example calls the f_CleanUpSessions function:

```
...
inv_session.f_CleanUpSessions&
    (SQLCA, DateTime(Today(), Now()) )
CHOOSE CASE inv_session.ii_errorcode
CASE 0
   RETURN("Old sessions removed")
CASE IS <0
   RETURN(inv_session.is_errormsg)
END CHOOSE
...
```

f_GenerateID

Description

Creates a new session ID.

Access

Public

Syntax

instancename.**f_GenerateID** ({ *transaction* })

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function with the *transaction* argument (passed by reference)

Return value

Long. Returns the session ID. if the function succeeds and -1 if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *transaction* was invalid
- -2 Unable to retrieve from www_new_session_id
- -3 Unable to update www_new_session_id

Usage

If you call this function with no arguments, it creates a new session ID equal to the number of seconds since 1/1/95. If you call this function passing a Transaction object, it creates a new session ID by incrementing the www_new_session_id table.

Examples

This example calls the f_GenerateID function:

```
Long   ll_sessionid

ll_sessionid = inv_session.f_GenerateID()
inv_session.f_NewSession &
   (SQLCA, remote_addr, ll_sessionid)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN("Error creating session ID." &
      + inv_session.is_errormsg)
END IF
...
```

f_GetArgumentValue

Description

Accesses the specified session argument value.

Access

Public

Syntax

instancename.**f_GetArgumentValue** (*transaction, sessionid, name*)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the value to be returned
name	String specifying the argument identifier, as specified in the f_SetArgumentValue function

Return value

String. Returns the requested value if the function succeeds and an empty string if an error occurs. If an error occurs, check the ii_errorcode instance variable for details:

- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 *name* was either NULL or an empty string
- -4 Retrieval failed for the www_session_argument table

Usage

Call this function to access a session argument. You save session arguments in the f_SetArgumentValue function. Before calling this function, you must establish a Transaction object.

Examples

This example calls the f_GetArgumentValue function:

```
...
ls_return = inv_session.f_GetArgumentValue &
    (SQLCA, ll_sessionid, "password")
...
```

f_NewSession

Description

Registers a new session by adding a row to the www_session table in the webpb database.

Access

Public

Syntax

instancename.**f_NewSession** (*transaction, ipaddress, sessionid*)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
ipaddress	String specifying the IP address of the browser user
sessionid	Long specifying the session ID

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- -1 *transaction* was invalid
- -2 *ipaddress* was either NULL or an empty string
- -3 *sessionid* was either NULL or an empty string
- -4 Error inserting the row into www_session

Usage

To access the user's IP address, pass the remote_addr argument to your user object function. Remote_addr is a CGI environment variable that contains the user's IP address.

Examples

This example calls the f_NewSession function:

```
Long   ll_sessionid

ll_sessionid = inv_session.f_GenerateID()
inv_session.f_NewSession &
   (SQLCA, remote_addr, ll_sessionid)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN("Error creating session ID." &
      + inv_session.is_errormsg)
END IF
...
```

f_SetArgumentValue

Description

Establishes a session argument. If the passed argument already exists for the current session, this function updates the value.

This function saves rows in the www_session_argument table.

Access

Public

Syntax

instancename.**f_SetArgumentValue** (*transaction, sessionid, name, value*)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with *name* and *value*
name	String specifying the argument identifier
value	String specifying the value associated with *name*

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 *name* was either NULL or an empty string
- -4 Error inserting or updating the row in www_session_argument

Usage

You access session arguments in other functions by calling the f_GetArgumentValue function. Before calling this function, you must establish a Transaction object for u_session.

Examples

This example calls the f_SetArgumentValue function (in this example, remote_addr, as_userid, and as_userpass are passed arguments):

```
Long   ll_sessionid

ll_sessionid = inv_session.f_GenerateID()
inv_session.f_NewSession &
   (SQLCA, remote_addr, ll_sessionid)
IF inv_session.ii_errorcode <> 0 THEN
   RETURN("Error creating session ID." &
            + inv_session.is_errormsg)
END IF
inv_session.f_SetArgumentValue(SQLCA, &
   ll_sessionid, "userid", as_userid)
inv_session.f_SetArgumentValue(SQLCA, &
```

```
        ll_sessionid, "password", as_userpass)
...
```

f_UpdateLastAccess

Description

Updates the last-accessed date and time for the specified session.

Access

Public

Syntax

instancename.**f_UpdateLastAccess** (*transaction*, *sessionid*, { *datetime* })

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the value to be returned
datetime	DateTime variable specifying the last accessed date and time. The default is the current date and time

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 *datetime* was NULL
- -5 The new date/time is less than the existing date/time
- -6 Error updating the row in www_session

Usage

Call this function to update active sessions so they won't be removed by the f_CleanUpSessions function. Before calling this function, you must establish a Transaction object.

Examples

This example calls the f_UpdateLastAccess function:

```
...
inv_session.f_UpdateLastAccess &
   (SQLCA, Long(as_sessionid), &
      DateTime(Today(), Now()) )
```

. . .

f_VerifySessionID

Description

Reports whetherthe specified session is valid. A valid session:

- Exists in the www_session table
- Has been updated within the last hour (by a function calling the f_UpdateLastAccess function)

Access

Public

Syntax

instancename.**f_VerifySessionID** (*transaction*, *sessionid*,)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID to be verified

Return value

Boolean. Returns TRUE if *sessionid* is valid and current and FALSE if it is not. If the function returns FALSE, you can access more information from the ii_errorcode instance variable:

- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 Error accessing the row in www_session

Usage

Call this function to verify that a session is valid and current. If the session is invalid or not current, return the user to the initial page. Before calling this function, you must establish a Transaction object.

To use this function, your application must update the session as appropriate by calling f_UpdateLastAccess function.

Examples

This example calls the f_VerifySessionID function:

```
. . .
IF NOT inv_session.f_VerifySessionID &
    (SQLCA, Long(as_sessionid)) THEN
    ls_html += inv_html_form.f_BeginForm &
       ("/cgi-bin/pbcgi60.exe/myapp/" &
```

```
            + "uo_webtest/f_displaylogin", 0)
        ls_html += "<P>Session has expired."
        ls_html += "<P>"
        ls_html += inv_html_form.f_MakeSubmit &
           ("Begin Logon")
        ls_html += inv_html_form.f_EndForm()
        RETURN(ls_html)
    END IF
    ...
```

u_transaction

Description

Transaction management service. Use this object to maintain information for application-specific units of work, called transactions. Transactions are associated with sessions and a session can have zero or more transactions.

This object updates the www_transaction, www_transaction_argument, and www_transaction_page tables, found in the webpb database, distributed with the IDT.

FOR INFO For complete information on using u_transaction, see "Managing distributed sessions" on page 179.

This is not a database transaction
In this context, a transaction is an application-specific unit of work, not a database transaction.

Ancestry

Inherits from NonVisualObject.

Library

Webpb.pbl

Usage

To use u_transaction:

1 (Optional) Establish a session, as described in u_session.

2 Declare an instance variable in your user object of type u_transaction:

```
u_transaction   inv_transaction
```

This user object uses PowerBuilder's autoinstantiate feature, so there is no need to code a CREATE statement.

3 Create a transaction ID by calling the f_GenerateID function:

```
Long   ll_transid

ll_transid = inv_transaction.f_GenerateID()
```

4 Add the transaction to the database by calling the f_NewTransaction function:

```
inv_transaction.f_NewTransaction &
   (SQLCA, ll_sessionid, ll_transid)
```

Before calling this function you must initialize database Transaction object fields and connect to the database. Typically you initialize and connect either in the server Application's ConnectionBegin event or at the beginning of the user object function.

5 Check the ii_errorcode instance variable to see if the function succeeded:

```
IF inv_transaction.ii_errorcode <> 0 THEN
   RETURN("Error creating trans ID.")
END IF
```

6 Save transaction arguments as necessary:

```
inv_transaction.f_SetArgumentValue &
   (SQLCA, ll_sessionid, ll_transid, &
      "deptid", as_deptid)
IF inv_transaction.ii_errorcode <> 0 THEN
   RETURN(inv_transaction.is_errormsg)
END IF
```

7 In the HTML that the function returns, add hidden fields containing the session ID and the transaction ID:

```
ls_sessionid = &
   inv_html_form.f_MakeHidden &
      ("as_sessionid", String(ll_sessionid))
ls_transactionid = &
   inv_html_form.f_MakeHidden &
      ("as_transactionid", &
         String(ll_transid))
```

8 Include session ID and transaction ID arguments (as_sessionid and as_transactionid in this example) in the user object function named in the displayed form's ACTION attribute.

9 Use the session ID and transaction ID to call u_transaction functions in subsequent user object functions as necessary to access and maintain transaction information.

See also

u_session

Instance variables

U_transaction includes instance variables:

Instance variable	Description	Data type	Access	Usage
ii_errorcode	Contains an error code for the last u_transaction function	Integer	Public	After calling certain u_transaction functions, use this instance variable to determine whether an error occurred
is_errormsg	Contains an error description	String	Public	If an error occurs, use this instance variable to access detailed error information

Functions

U_transaction includes precoded object functions:

f_CleanUpTransactions	f_NewTransaction
f_GenerateID	f_SetArgumentValue
f_GetArgumentValue	f_SetTransactionPage
f_GetTransactionPage	f_VerifyTransactionID

f_CleanUpTransactions

Description

Deletes all transactions before the specified date and time. This function deletes rows from the www_transaction, www_transaction_argument, and www_transaction_page tables.

Access

Public

Syntax

instancename.**f_CleanUpTransactions** (*transaction*, *enddatetime*)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
enddatetime	DateTime variable before which the function deletes transactions

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- -1 *transaction* was invalid
- -2 *enddatetime* was NULL
- -3 Error declaring the cursor
- -4 Error opening the cursor
- -5 Error fetching the cursor
- -6 Error closing the cursor
- -7 Error deleting rows in www_transaction_argument
- -8 Error deleting rows in www_transaction_page
- -9 Error deleting rows in www_transaction

Usage

Call this function to remove old transactions from the database.

You typically call this function on a regular basis from the application server's console application.

Removing old transactions
What constitutes an old transaction is a site-specific consideration. Depending on your application, five minutes of inactivity might signal an inactive transaction. In other cases, 20 minutes of inactivity might signal an inactive transaction.

Examples

This example calls the f_CleanUpTransactions function:

```
...
inv_transaction.f_CleanUpTransactions &
   (SQLCA, DateTime(Today(), Now()) )
CHOOSE CASE inv_transaction.ii_errorcode
CASE 0
   RETURN("Old transactions removed")
CASE IS <0
   RETURN(inv_transaction.is_errormsg)
END CHOOSE
...
```

f_GenerateID

Creates a transaction ID. There are two syntaxes:

To	Use
Create a transaction ID using an optionally specified date and time	Syntax 1
Create a sequential transaction ID	Syntax 2

Syntax 1

Create a transaction ID using an optionally specified date and time

Description

Creates a transaction ID based on the number of seconds since either 1/1/95 or a specified date and time.

Access

Public

Syntax

instancename.**f_GenerateID** ({ *connectdatetime* })

Argument	Description
instancename	Instance name of u_transaction
connectdatetime (optional)	DateTime variable specifying the date and time the user initially connected to the application

Return value

Long. Returns the transaction ID.

Examples

This example calls the f_GenerateID function:

```
...
ll_transid = inv_transaction.f_GenerateID()
inv_transaction.f_NewTransaction &
    (SQLCA, Long(as_sessionid), ll_transid)
...
```

Syntax 2

Create a sequential transaction ID

Description

Generates a sequential transaction ID, incrementing the www_new_transaction_id table in the webpb database.

Access

Public

Syntax

instancename.**f_GenerateID** (*transaction*)

Argument	Description
instancename	Instance name of u_transaction

Argument	Description
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)

Return value

Long. Returns the transaction ID if the function succeeds and -1 if an error occurs. If an error occurs, ii_errorcode contains values as follows:

- -1 *transaction* was invalid
- -2 Unable to retrieve from www_new_transaction_id
- -3 Unable to update www_new_transaction_id

Examples

This example calls the f_GenerateID function:

```
...
ll_transid = inv_transaction.f_GenerateID(SQLCA)
inv_transaction.f_NewTransaction &
   (SQLCA, Long(as_sessionid), ll_transid)
...
```

f_GetArgumentValue

Description

Accesses the specified transaction argument value.

Access

Public

Syntax

instancename.**f_GetArgumentValue** (*transaction, sessionid, transactionid, name*)

Argument	Description
instancename	Instance name of u_transaction
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the transaction
transactionid	Long specifying the transaction ID associated with the value to be returned
name	String specifying the argument identifier, as specified in the f_SetArgumentValue function

Return value

String. Returns the requested value if the function succeeds and an empty string if an error occurs. If an error occurs, check the ii_errorcode instance variable for details:

- -1 *transaction* was invalid
- -2 *session* was either NULL or less than 1
- -3 *transactionid* was either NULL or less than 1
- -4 *name* was either NULL or an empty string
- -5 Retrieval failed for the www_transaction_argument table

Usage

You save transaction attributes in the f_SetArgumentValue function.

Before calling this function, you must establish a Transaction object for u_transaction.

Examples

This example calls the f_GetArgumentValue function:

```
...
ls_deptid = inv_transaction.f_GetArgumentValue &
    (SQLCA, Long(as_sessionid), &
       Long(as_transactionid), "deptid")
...
```

f_GetTransactionPage

Description

Retrieves a previously saved page of data.

Access

Public

Syntax

instancename.**f_GetTransactionPage** (*transaction*, *sessionid*, *transactionid*, *pagenumber*, *pagedata* {, *startrow* {, *startcolumn* } })

Argument	Description
instancename	Instance name of u_transaction
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the transaction
transactionid	Long specifying the transaction ID associated with the page to be returned
pagenumber	Integer specifying the page whose data is to be returned

Argument	Description
pagedata	DataStore into which the function places the page's rows (passed by reference)
startrow (optional)	Long specifying the row in *pagedata* at which the function begins appending rows
startcolumn (optional)	Long specifying the column in *pagedata* at which the function begins appending columns

Return value

Integer. Returns values as follows:

- >0 The number of rows inserted into *pagedata* (success)
- <0 An error occurred

Examples

This example calls the f_GetTransactionPage function (as_page, as_sessionid, and as_transactionid are passed arguments):

```
Integer   li_return
DataStore   lds_page

lds_page = CREATE DataStore
lds_page.dataobject = "d_employee"

li_return = &
   inv_transaction.f_GetTransactionPage(SQLCA, &
      Long(as_sessionid), Long(as_transactionid), &
         Integer(as_page), lds_page)
RETURN lds_page.Object.DataWindow.Data.HtmlTable
```

f_NewTransaction

Description

Registers a new transaction by adding a row to the www_transaction table in the webpb database.

Access

Public

Syntax

instancename.**f_NewTransaction** (*transaction, sessionid, transactionid*)

Argument	Description
instancename	Instance name of u_transaction
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)

Argument	Description
sessionid	Long specifying the session ID
transactionid	Long specifying the transaction ID

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- 0 Success
- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than zero
- -3 *transactionid* was either NULL or less than zero
- -4 Error inserting the row into www_transaction

Examples

This example calls the f_NewTransaction function:

```
Long   ll_transid
...
ll_transid = inv_transaction.f_GenerateID(SQLCA)
inv_transaction.f_NewTransaction&
   (SQLCA, Long(as_sessionid), ll_transid)
...
```

f_SetArgumentValue

Description

Establishes a transaction argument. If the passed argument already exists for the current transaction, this function updates the argument's value.

This function saves rows in the www_transaction_argument table.

Access

Public

Syntax

instancename.**f_SetArgumentValue** (*transaction*, *sessionid*, *transactionid*, *name*, *value*)

Argument	Description
instancename	Instance name of u_transaction
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the transaction

Argument	Description
transactionid	Long specifying the transaction ID associated with the value to be saved
name	String specifying the argument identifier
value	String specifying the value associated with *name*

Return value

None. After calling this function, check the ii_errorcode instance variable for errors:

- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 *transactionid* was either NULL or less than 1
- -4 *name* was either NULL or an empty string
- -5 Retrieval or update failed for the www_transaction_argument table

Usage

You access transaction arguments in other functions by calling the f_GetArgumentValue function.

Examples

This example calls the f_SetArgumentValue function:

```
...
inv_transaction.f_SetArgumentValue &
   (SQLCA, Long(as_sessionid), ll_transid, &
      "deptid", ls_deptid)
...
```

f_SetTransactionPage

Description

Caches a DataStore into separate rows in the www_transaction_page table. This allows you to limit the amount of information returned to the browser with each page.

Access

Public

Syntax

instancename.**f_SetTransactionPage** (*transaction*, *sessionid*, *transactionid*, *rowsperpage*, *datastore* {, *beginrow*, *endrow* {, *begincolumn*, *endcolumn* } })

Argument	Description
instancename	Instance name of u_transaction

Argument	Description
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the transaction
transactionid	Long specifying the transaction ID associated with the value to be saved
rowsperpage	Integer specifying the number of rows on each page
datastore	DataStore containing the rows to be saved in the www_transaction_page table. The DataWindow object associated with *datastore* cannot have columns of type TimeStamp (passed by reference)
beginrow (optional)	Long specifying the first row to be included in the saved pages. The default is the first row. If you specify this argument you must also specify *endrow*
endrow (optional)	Long specifying the last row to be included in the saved pages. The default is the last row. The default is the first row. If you specify this argument you must also specify *beginrow*
begincolumn (optional)	Integer specifying the first column to be included in the saved pages. The default is the first column. The default is the first row. If you specify this argument you must also specify *endcolumn*
endcolumn (optional)	Integer specifying the last column to be included in the saved pages. The default is the last column. The default is the first row. If you specify this argument you must also specify *beginrow*

Return value

Integer. Returns values as follows:

- 1 Function succeeded
- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 *transactionid* was either NULL or less than 1
- -4 *rowsperpage* was either NULL or less than 1
- -5 *datastore* was either NULL or contained no rows
- -6 *beginrow* was NULL, less than one, or greater than the number of rows in *datastore*
- -7 *endrow* was less than *beginrow*
- -8 *begincolumn* was NULL, less than one, or greater than the number of columns in *datastore*
- -9 *endcolumn* was less than *begincolumn*
- -10 Insert failed for www_transaction_page
- -11 UpdateBlob failed for www_transaction_page
- -12 *datastore* contained a column of type TimeStamp

Usage

Call this function to convert a DataStore into sets of rows in the www_transaction_page table.

Examples

This example calls the f_SetTransactionPage function:

```
Integer   li_return, li_numpages
DataStore   lds_data, lds_page
Long      ll_transid

lds_data.DataObject = "d_employee"
lds_page = CREATE DataStore
lds_page.DataObject = "d_employee"
lds_data.SetTransObject(gtr_trans)
li_return = lds_data.Retrieve()
IF li_return < 1 THEN
   RETURN("Retrieval error.<P>Return code was " &
      + String(li_return))
ELSE
   ll_transid = inv_transaction.f_GenerateID(SQLCA)
   inv_transaction.f_NewTransaction &
      (SQLCA, Long(as_sessionid), ll_transid)
```

```
        li_numpages = &
           inv_transaction.f_SetTransactionPage(SQLCA, &
              Long(as_sessionid), ll_transid, 10, &
                 lds_data)
        IF li_numpages < 1 THEN RETURN("No Rows Cached")
        li_return = &
           inv_transaction.f_GetTransactionPage(SQLCA, &
              Long(as_sessionid), ll_transid, 1, lds_page)
        IF li_return < 1 THEN
           RETURN("Page Retrieval Error: " + &
              inv_transaction.is_errormsg)
        END IF
     END IF
     RETURN(lds_page.Object.DataWindow.Data.HtmlTable)
```

f_VerifyTransactionID

Description

Reports whether the specified transaction is valid. A valid transaction:

- Exists in the www_transaction table
- Was created within the last three hours

Access

Public

Syntax

instancename.**f_VerifyTransactionID** (*transaction*, *sessionid*, *transactionid*)

Argument	Description
instancename	Instance name of u_session
transaction	Transaction object used to access the webpb database. You must connect to the webpb database before calling this function (passed by reference)
sessionid	Long specifying the session ID associated with the transaction to be verified
transactionid	Long specifying the transaction ID to be verified

Return value

Boolean. Returns TRUE if *transactionid* is valid and current and FALSE if it is not. If the function returns FALSE, you can access more information from the ii_errorcode instance variable:

- -1 *transaction* was invalid
- -2 *sessionid* was either NULL or less than 1
- -3 *transactionid* was either NULL or less than 1
- -4 Error accessing the row in www_transaction

Usage

If the session is invalid or not current, return the user to the initial page. Before calling this function, you must establish a Transaction object.

Examples

This example calls the f_VerifyTransactionID function:

```
...
IF NOT inv_transaction.f_VerifyTransactionID &
   (SQLCA, Long(as_sessionid), &
      Long(as_transactionid)) THEN
   ls_html += inv_html_form.f_BeginForm &
      ("/cgi-bin/pbcgi60.exe/myapp/" &
         + "uo_webtest/f_displaylogin", 0)
   ls_html += "<P>Transaction has expired."
   ls_html += "<P>"
   ls_html += inv_html_form.f_MakeSubmit &
      ("Begin Logon")
   ls_html += inv_html_form.f_EndForm()
   RETURN(ls_html)
END IF
...
```

PART 3

Plug-Ins

This part describes how to use the PowerBuilder window plug-in and DataWindow plug-in components of the Internet Tools.

CHAPTER 7

Using the PowerBuilder Window Plug-In

About this chapter

This chapter describes how to develop, test, and deploy a PowerBuilder application that will be displayed as a plug-in application in a Web page.

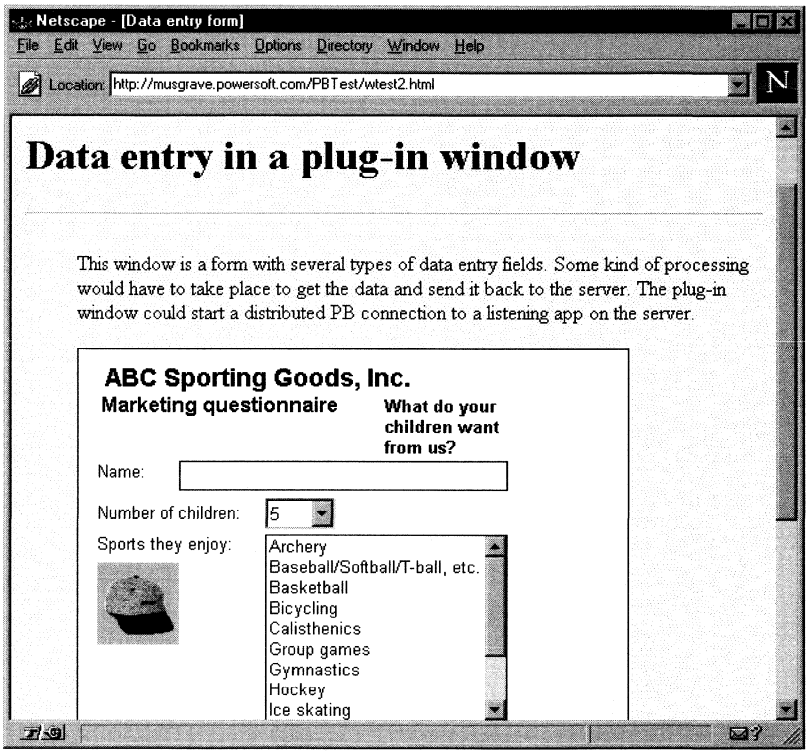
Contents

Before you begin

This chapter assumes you are familiar with HTML and URLs and how a Web browser obtains pages from a Web server, and that you have access to a Web server.

About the PowerBuilder window plug-in

The PowerBuilder window plug-in lets you display a PowerBuilder child window on a Web page viewed in a browser that supports Netscape plug-ins.



Features

The PowerBuilder child window can include all the familiar controls, including DataWindows, OLE objects, OCX (ActiveX) controls, and tree controls. You can also open other (popup or response) windows from the child window.

As the user interacts with controls in the child and other windows, scripts for the controls' events are executed just as they are in standalone PowerBuilder applications. Database access by the plug-in application occurs locally using the client's defined database connections.

The objects in the application can be contained in one or more PowerBuilder Dynamic Libraries (PBDs).

Standard and secure versions

There are two versions of the PowerBuilder window plug-in: standard and secure.

Standard PowerBuilder window plug-in The standard PowerBuilder window plug-in displays a PowerBuilder child window in an HTML page. The standard window-plug-in is implemented by NPPBA60.DLL on Windows and the PowerBuilder Window Plugin shared library on Power Macintosh.

Secure PowerBuilder window plug-in The secure PowerBuilder window plug-in is a secure version of the standard window plug-in. Using the secure version ensures that PowerBuilder applications downloaded over the Internet will not damage a client system or access information on a client workstation. The secure window-plug-in is implemented by NPPBS60.DLL on Windows and the PowerBuilder Secure Window shared library on Power Macintosh.

FOR INFO For considerations to keep in mind when using the secure version of the PowerBuilder window plug-in, see "Using the secure PowerBuilder window plug-in" on page 317.

Supported platforms

PowerBuilder provides the window plug-in for the following operating system platforms:

- Windows NT
- Windows 95
- Power Macintosh

Supported browsers

The PowerBuilder window plug-in requires use of a browser that supports Netscape plug-ins, such as:

- Netscape Navigator Version 3.x or higher
- Microsoft Internet Explorer Version 3.x or higher

Security

The standard PowerBuilder window plug-in is a nonsecure application, meaning that it can access local files and can run local applications. These types of processing are often undesirable in an uncontrolled Web environment but may be perfectly acceptable on a corporate intranet where access is controlled.

If security is important at your site, consider using the secure version of the PowerBuilder window plug-in to build your application.

FOR INFO For more about the secure PowerBuilder window plug-in, see "Using the secure PowerBuilder window plug-in" on page 317.

What kinds of applications make good plug-ins?

HTML forms provide some user interaction via a limited user interface. The PowerBuilder window plug-in takes you beyond HTML. You can present a fully developed application window with a rich user interface design to a Web page. You can access data sources defined on the client workstation. You can use PowerBuilder's distributed computing capabilities to access the server and its data sources.

Examples

Your application might be:

- A data analysis window with master and detail DataWindows
- A user interface design that lets the user make choices using TreeView, ListView, and PictureListBox controls
- A data entry form that uses PowerBuilder's distributed computing capabilities to process the data on the server
- A data entry form that uses the client's defined database connection (either network or local) to process the data on the client machine
- (*On Windows only*) A window that uses PowerBuilder's OLEControl control to display an ActiveX control

How the PowerBuilder window plug-in works

The PowerBuilder window plug-in displays a PowerBuilder child window inside a fixed space reserved on the Web page. The user can interact with the controls on the page, and the PowerBuilder scripts for the window and its controls can execute any PowerBuilder code. When the user switches to another Web page, the PowerBuilder window is closed and the PowerBuilder DLLs or shared libraries are unloaded from memory.

You include the plug-in on an HTML page using the HTML Embed element. The Embed element names one or more PBDs that contain PowerBuilder objects and the name of the child window object that is displayed on the page.

The PowerBuilder window plug-in implements the Netscape Plug-in API and requires a browser that supports this API (for information, see "Supported browsers" on page 313.)

Client-server interactions, in detail

This table describes in detail what happens between the client and server when the user views an HTML document with a PowerBuilder window plug-in.

DLLs and shared libraries

On Windows, the PowerBuilder window plug-in and deployment components are DLLs. On Macintosh, they are shared libraries. (The PowerBuilder window plug-in is not supported on UNIX.)

Step	What the client does	How the server responds
1	The Web browser requests the HTML document from the server	The server sends a header identifying the document's MIME type (text/html)
2	The browser receives the MIME type and prepares to receive an HTML document	The server sends the HTML document
3	The browser receives the HTML document and displays it	—
4	The browser recognizes the Embed element, reserves space for the plug-in on the page, and requests the PBD file from the server	The server sends a header identifying the PBD's MIME type, which can be one of the following: **Standard window plug-in** application/vnd.powerbuilder6 **Secure window plug-in** application/vnd.powerbuilder6-s
5	The browser receives the MIME type and prepares to receive a PBD file	The server sends the PBD file
6	The browser receives the PBD file	—
7	The browser searches its plug-in directory for the DLL or shared library that corresponds to the MIME type (see server step 4)	—
8	The browser loads the plug-in DLL or shared library	—
9	The plug-in looks for and loads the PowerBuilder deployment DLLs or shared libraries	—
10	If the Embed element includes LIBRARY attributes, then the client requests the specified PBD files	The server sends a header identifying the PBD's MIME type (see server step 4)

Step	What the client does	How the server responds
11	The browser receives the MIME type and prepares to receive additional PBD files	The server sends the PBD file
12	PowerBuilder displays the child window specified in the Embed element	—
13	The child window executes its Open script	—
14	If a script calls the CommandParm function, PowerBuilder queries the browser for the value of the COMMANDPARM attribute in the Embed element	—

Requirements

The PowerBuilder window plug-in uses the PowerBuilder deployment DLLs or shared libraries as well as the plug-in DLL or shared library to provide a full range of PowerBuilder functionality.

Each client that will browse pages containing PowerBuilder window plug-ins needs supporting software installed on the local machine:

- PowerBuilder deployment DLLs or shared libraries
- Window plug-in DLL or shared library

The PowerBuilder window plug-in is especially useful in an intranet application where you have control over the setup of client machines.

FOR INFO For details about setting up client machines on each supported platform, see "Setting up users' workstations" on page 334.

Name and location on different platforms

The name of the PowerBuilder window plug-in DLL (on Windows) or shared library (on Macintosh) and where PowerBuilder installs it depends on the version of the PowerBuilder window plug-in you are using and your operating system platform:

Component	On Windows	On Power Macintosh
Standard PowerBuilder window plug-in	PB6\IT\PLUGIN32\ NPPBA60.DLL	Powersoft 6.0 Folder:Internet Tools:Netscape Plugins:PowerBuilder Window Plugin

Component	On Windows	On Power Macintosh
Secure PowerBuilder window plug-in	PB6\IT\PLUGIN32\ NPPBS60.DLL	Powersoft 6.0 Folder:Internet Tools:Netscape Plugins:PowerBuilder Secure Window

Using the secure PowerBuilder window plug-in

How to use it

You follow the same basic steps to develop and deploy applications using the secure PowerBuilder window plug-in as you do when using the standard PowerBuilder window plug-in. (These steps are described in "Developing and deploying a PowerBuilder window plug-in application" on page 318.)

Then only difference is that the secure PowerBuilder window plug-in uses a special version of the standard window-plug-in DLL or shared library. The secure version of the window plug-in is NPPBS60.DLL on Windows and the PowerBuilder Secure Window shared library on Macintosh.

What you do

To use the secure PowerBuilder window plug-in, make sure NPPBS60.DLL or the PowerBuilder Secure Window shared library is in the Web browser's plug-ins directory on each client workstation.

On Macintosh, the PowerBuilder Installer will install a copy of the PowerBuilder Secure Window shared library to the browser's plug-ins folder if it finds a Web browser on the machine.

Restricted functionality

Using the secure PowerBuilder window plug-in severely restricts the PowerBuilder application running on the client workstation, denying access to the client system except for printing. For this reason, using the secure window plug-in may not be appropriate in all situations.

These types of activities are restricted when you use the secure PowerBuilder window plug-in:

Activity	Restriction
External functions	Calling an external function causes an execution time error
Certain PowerScript functions	Calling a restricted PowerScript function causes an execution time error
Database connection	Calling PowerScript functions that result in database access causes an execution time error

Activity	Restriction
Internet access	Applications that use the secure PowerBuilder window plug-in can establish an Internet connection only to the current Web server
E-mail	Calling PowerScript Mail functions causes an execution time error
OLE	Calling PowerScript OLE functions causes an execution time error
Distributed computing	You cannot connect to PowerBuilder application servers
Dynamic Data Exchange (DDE)	Calling PowerScript DDE functions causes an execution time error

Developing and deploying a PowerBuilder window plug-in application

Basic steps

There are four main tasks involved in developing a plug-in application.

❖ **To create and deploy a PowerBuilder window plug-in application:**

1 Create, test, and build the PowerBuilder application.

2 Create an HTML page that embeds your PowerBuilder application window.

3 Configure the Web server by registering the appropriate content type (MIME type) and copying the HTML page and PBD files for the application to appropriate directories.

4 On all client workstations, install the standard or secure PowerBuilder window plug-in DLL or shared library and the PowerBuilder deployment DLLs or shared libraries.

Resulting components

When everything is set up, the various computers will have the following components:

Computer	Component
Server	MIME type registered FOR INFO For instructions, see "Specifying the MIME type" on page 332
	HTML page with an Embed element for the PBD

Computer	Component
	One or more PBDs containing your Application objects
Client	PowerBuilder window plug-in DLL or shared library in the browser's plug-ins directory
	PowerBuilder deployment DLLs or shared libraries installed using the Deployment Kit for your platform
	(*On Windows only*) Directory for the PowerBuilder deployment DLLs listed in the system path
	Any additional software required by the plug-in application, such as database connection software or OLE servers and custom controls

What's next

The rest of this chapter describes the four steps for creating and deploying a PowerBuilder window plug-in application:

- Creating the PowerBuilder application
- Creating an HTML page
- Setting up the server
- Setting up users' workstations

Creating the PowerBuilder application

The starting point of your PowerBuilder window plug-in application is a child window displayed in a Web page. In the window you can include controls and write scripts for events. Your scripts can open other windows, read and write files, and run other programs on the client machine. The application can contact a PowerBuilder server application to initiate a distributed PowerBuilder session.

Design choices for plug-in applications

The application you design for use as a plug-in can be much the same as other PowerBuilder applications you develop. But there are some restrictions and considerations to keep in mind in the following areas:

- Window management
- Objects
- Scripts and variables
- Data access
- External files

Window management

Initial child window Your initial window needs to be a child window that lives in the browser frame.

You *can*:

- Include a title bar on the child window

  But you should not use a control menu, maximize box, or minimize box on that window.
- Open popup and response windows from the child window (but not main or MDI windows)

You *cannot*:

- Have a menu for the child window
- Open another child window from the initial window

On Macintosh Although child windows are not a typical Macintosh window type, they behave on Macintosh as they do on Windows. On Macintosh, you can open modeless dialog (popup window), modal dialog (response windows without title bar), and movable modal dialog (response window with title bar) windows from the child window.

Closing windows When the client browses to another Web page, the child window on the current Web page is closed. But other windows remain open unless your application closes them. You must close them in the child window's Close or CloseQuery events.

Do not try to stop the child window from being closed in the CloseQuery event by setting a return value. You can't prevent the browser from changing to another page and removing the window from view. If the user returns to the page, another instance of your application will be started.

Objects

Objects in your PBDs Your plug-in application has access to all objects in the PBDs. This includes, functions, structures, and user objects.

System objects Your plug-in application has access to system objects that PowerBuilder instantiates, such as the SQLCA Transaction object and the Message object.

Application object You can use the optional APPLICATION attribute of the Embed element to specify the name of your PBD's Application object. This gives your plug-in application access to the Application object's Open and Close events.

If you do *not* specify the APPLICATION attribute:

- Your plug-in application does not have access to the Application object, and thus events like SystemError and Idle are not available. You cannot treat variables and functions defined in an Application object as global.
- Any scripts in the Application object will be used only during testing within PowerBuilder. You cannot do any application setup in the Application object's scripts.

FOR INFO For information about specifying the APPLICATION attribute, see "Attributes of the Embed element" on page 328.

Scripts and variables

Global variables If you use the APPLICATION attribute of the Embed element to specify your PBD's Application object, your plug-in application has access to global variables and global functions used in the application.

If you do *not* specify the APPLICATION attribute, the plug-in application cannot use global variables. You must define all variables as instance, shared, or local.

FOR INFO For information about specifying the APPLICATION attribute, see "Attributes of the Embed element" on page 328.

Referencing the initial window You can't reference the initial child window by name in your scripts because PowerBuilder doesn't create a variable to hold the instance of it. (By contrast, when you instantiate a window yourself by coding the Open function, you always place the instance in a variable that you can then reference.)

As a result, the following code produces an error at execution time (because the window variable w_mychild does not exist):

```
// This code produces an execution time error:
w_mychild.title = "The initial window"
```

But you *can* code:

```
// In the child window itself:

this.title = "The initial window"
// or
title = "The initial window"

// In controls of the child window:

parent.title = "The initial window"
```

Scripts for application setup All application setup must be done in the child window, including connecting to the DBMS. The first scriptable events to occur are the constructors for the controls in the window. Then the Open event for the window occurs.

The Activate event does not occur for a child window; so don't do application setup there.

Data access

If your application accesses a DBMS, each client must have a connection to the data source. The connection must be defined on the client's machine, not the server. The data source might be a local or a network DBMS.

FOR INFO For information about how to connect to a DBMS from the client machine, see *Connecting to Your Database*.

The constructor events for controls occur before the Open event of the window. If you connect to the DBMS in the window's Open event, the constructor events can't get data. You can get data for controls in the window's Open event or you can post events from the constructor events or the window's Open event.

Paths for external resources

Paths you specify for files must be valid on the client workstation.

If your application uses images as external files, the images must be available at the path specified in the PowerBuilder object. Instead of using external files, you can build image resources into PBDs, as described in "Building the dynamic libraries" on page 325.

If your application reads or writes local files, the path for those files must be valid on each client's machine.

On Windows If a path refers to a network drive by mapped drive letter, all clients must use the same drive letter that the application uses. As an alternative, specify the server name in the path instead.

For example, both of these paths are valid when o: is mapped to \\marketing\drive. But the second path, that uses a server name, is more likely to *remain* valid.

```
O:\pbapps\connect.bmp
\\marketing\drive\pbapps\connect.bmp
```

On Macintosh Use full path statements to reference external files. For example:

```
MacServer:PowerBuilder Apps:Employee.bmp
```

Defining the starting window in the Window painter

You use the Window painter to create the starting window, as you do for any window you create in PowerBuilder.

❖ **To create your application's starting (or only) window:**

1 In the Window painter, create a new window.

2 On the window's property sheet, set the window type to child.

3 Add other controls as needed.

4 Write scripts for events of the window and controls.

❖ **To convert an existing application to run as a plug-in:**

1 Change the type of the opening window to child.

2 If you do *not* plan to specify the APPLICATION attribute of the Embed element, do both of the following:

- Remove references to global variables
- Move application setup code from the application's Open event or MDI frame events to the child window's Open event

FOR INFO For information about specifying the APPLICATION attribute, see "Attributes of the Embed element" on page 328.

3 Depending on the application's design, you may need to redesign how it opens other windows.

About child windows

Child windows cannot have menus. They are never considered the active window—and therefore the Activate event is never triggered. They can have title bars and can be minimized, maximized, and resized. In the plug-in environment, the child window is always restricted to the space allotted by the WIDTH and HEIGHT attributes specified on the Web page, as follows:

- Maximizing causes the window to fill the space allotted by the WIDTH and HEIGHT attributes
- Minimizing displays the window's icon and title at the bottom of the plug-in's allotted space
- If the child window is resizable, the user can drag the borders to make the window smaller (but not larger) than the allotted space

As a result, it is not useful to allow minimizing, maximizing, or resizing of the child window.

On Macintosh
Although child windows are not a typical Macintosh window type, they behave on Macintosh as they do on Windows.

Testing the application in PowerBuilder

Before you try your application in a client browser, you can test it in PowerBuilder by defining a main window that opens the child window.

❖ **To test your PowerBuilder window plug-in application:**

1 In the Window painter, create a new window whose type is main (the default).

2 Write a script for the Open event that opens the plug-in's starting child window.

```
Open(w_child)
```

3 For convenience, you can set appropriate window sizes in the Window painter:

- Make the main window large enough to display the child
- Position the child window in the upper-left corner (use the Position tab of the property sheet)

4 Run the test window by clicking the Run Window icon or pressing CTRL+SHIFT+W (on Windows) or COMMAND+SHIFT+W (on Macintosh).

Debugging in PowerBuilder

To use the PowerBuilder debugger, you need to run an application instead of a single window. Define an Application object with a script that opens the main window. Then you can use the Run or Debug commands to test the application.

Building the dynamic libraries

The *PowerBuilder User's Guide* describes how to use the Library painter or Project painter to build dynamic runtime libraries. The procedure is the same for a plug-in application. This section highlights the choices you need to make for building a PBD for a plug-in application.

In the Web environment, file size is important—because clients are (unless it happens to be cached because they ran it recently).

Organizing objects in PBLs

Before you build your application, you should use the Library painter to organize the objects your application uses in PBLs, which will be the sources for the plug-in application's PBDs. The following suggestions will help you optimize the resulting libraries:

- To minimize file size, include only objects the application uses; remove any objects that are not needed.
- Include any objects that are dynamically created, such as DataWindow objects used in DataStores or assigned dynamically to DataWindow controls and proxy objects.
- Include ancestor objects.

Using PowerBuilder resource files

Several controls can use external files for images. These include PictureListBox, TreeView, Picture, and PictureButton controls, pointers, and bitmap objects in DataWindow objects. If your application uses external files for the images, it is unlikely that the client has the same images on the same system path.

Instead of finding some way to install the pictures on client machines, you can use one or more PowerBuilder resource (PBR) files so that the images are built into the PBDs. The resulting PBDs will be larger but self contained.

Because you are building a PBD, not an executable, you do not need to include DataWindow objects in the resource file. All PowerBuilder objects in the source PBL are included in the resulting PBD.

Images and other resources on a network
Instead of building the files into the PBD, you can put the files in a generally accessible network directory. But the path to the files must be identical to the path named in the PowerBuilder objects. This means that, in the Windows environment, each client must use the same drive letter to map the network drive, or you can specify the server name in the path.

❖ **To define a PBR file:**

1. Open the PowerBuilder File Editor (SHIFT+F6) or some other text editor and create a new file with the extension PBR. (You can add the File Editor icon to the toolbar.)
2. List each image or other resource on its own line. List the path and filename exactly as it is named in the object property sheet or script.

Shortcut
Look at the object's property sheet and use Edit>Copy (CTRL+C on Windows or COMMAND+C on Macintosh) to copy the filename to the clipboard. Then paste it into the editor.

3 Save the file.

If your application includes several PBLs each with objects using their own resources, you will want to create a PBR file for each PBL. The PBR file will list filenames for resources that are used in one PBL.

Using Macintosh resource files

On the Macintosh, you can use Macintosh resource files in addition to PowerBuilder resource files (PBRs) if you want to supply your own version of the resources that your application needs.

FOR INFO For instructions on defining a Macintosh resource file, see the chapter on creating an executable in the *PowerBuilder User's Guide*.

Building the PBDs

Build your runtime libraries (PBDs) in the Library painter or Project painter as described in the *PowerBuilder User's Guide*. Keep in mind:

- **Deselect Machine code** Your plug-in application must have PBDs, not machine code DLLs.
- **Specify a PBR file for each PBD** If the objects in the PBL use resources that you have listed in a PBR file, put the PBR name in the Resource File Name textbox.

FOR INFO For instructions on defining a PBR file and building your runtime libraries (PBDs), see the chapter on creating an executable in the *PowerBuilder User's Guide*.

What's next

After creating and building the PowerBuilder PBDs for your plug-in application, you need to create the HTML page that displays it.

Creating an HTML page

You include a PowerBuilder window on a Web page with the Embed element. Element attributes specify the space allocated for the window, the name of the PBD, and the name of the child window in the PBD.

A sample Embed element might look like this:

```
<EMBED SRC=plugin_tree.pbd WIDTH=370 HEIGHT=320
WINDOW=w_emp_by_dept>
```

Attributes of the Embed element

The Embed element is part of the HTML specification for plug-ins. It defines several standard attributes, and PowerBuilder defines additional attributes.

HTML attributes

HTML attributes name the file to be downloaded to the client and the space reserved for the plug-in on the Web page:

HTML attribute	Value
SRC	A URL identifying the object to be downloaded When the browser processes the Embed element, it requests the resource from the server and finds the DLL or shared library that handles the content type in its plug-ins directory For a PowerBuilder window plug-in, the object is a PBD containing the child window that starts the application
WIDTH	The width of the viewing window in pixels
HEIGHT	The height of the viewing window in pixels

The WIDTH and HEIGHT attributes define the maximum width and height of the child window. If the child window is resizable, the user can make it smaller than the specified size, but not larger.

PowerBuilder attributes

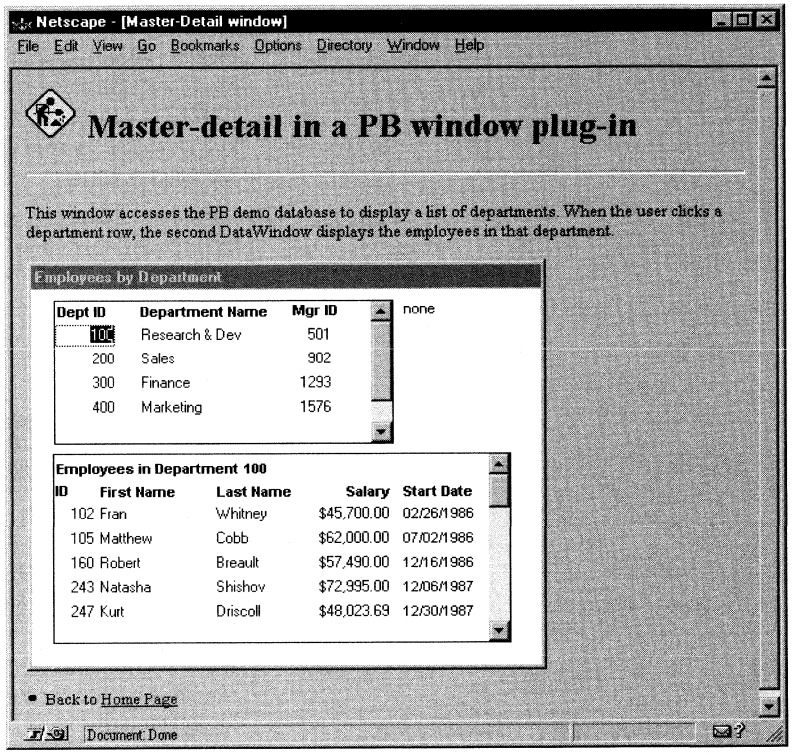
The PowerBuilder attributes for the Embed element let you identify the window object that starts your application, additional libraries, parameters to pass to the application, and the name of your Application object:

PowerBuilder attribute	Value
WINDOW	The class name of the child window in the PBD

PowerBuilder attribute	Value
LIBRARY (optional)	A URL specifying an additional PBD that contains objects that your application needs. You must specify the *full URL* for the PBD, rather than a relative one You can specify more than one LIBRARY attribute. Specify a LIBRARY attribute for each additional PBD your application needs Don't specify a LIBRARY attribute for the file specified for SRC FOR INFO For an example of HTML code that uses the LIBRARY attribute, see "Embed element with additional attributes" on page 331
COMMANDPARM (optional)	A string that you want to pass to your window. To access this string from within your window, call the CommandParm function FOR INFO For an example of HTML code that uses the COMMANDPARM attribute, see "Embed element with additional attributes" on page 331
APPLICATION (optional)	The name of the Application object in the PBD. This gives your plug-in application access to the Application object's Open and Close events, as well as to global variables and global functions used in the application When you use the APPLICATION attribute, the Application object's Open and Close events execute by default, and cannot be overridden Make sure the Application object's Open event does *not* open the child window specified by the WINDOW attribute. If it does, your application will fail FOR INFO For an example of HTML code that uses the APPLICATION attribute, see "Embed element with additional attributes" on page 331

Sample page

This sample page includes the PowerBuilder window plug-in showing a window with master and detail DataWindow controls:



Here is the HTML code that produces this page. Note the use of the Embed element (shown in bold) to specify the PowerBuilder window plug-in.

Opening element

```
<HTML>
```

Document head

```
<HEAD>
    <TITLE>Master-Detail window</TITLE>
    </HEAD>
    <BODY>
```

A small image

```
<IMG SRC="undercon.gif" BORDER=0 HEIGHT=38 WIDTH=40>
```

H1 heading

```
<H1>Master-detail in a PB window plug-in</H1>
```

Horizontal rule

```
<P><HR></P>
```

Paragraph	`<P>This window accesses the PB demo database to display a list of departments. When the user clicks a department row, the second DataWindow displays the employees in that department.</P>`
Embed element in a paragraph	`<P>`**`<EMBED SRC=plugin_tree.pbd WIDTH=370 HEIGHT=320 WINDOW=w_emp_by_dept>`**`</P>`
Link to site's home page	`<LI>Back to <A HREF="http://www.mycompany.com/index.html">Home Page</A> </LI>`
Closing elements	`</BODY> </HTML>`

Embed element with additional attributes

If your plug-in application uses additional libraries, a COMMANDPARM string, and the APPLICATION attribute to provide access to the PBD's Application object, the Embed element might look like this:

```
<EMBED SRC=plugin_tree.pbd WIDTH=370 HEIGHT=320
WINDOW=w_emp_by_dept
LIBRARY=http://www.mycompany.com/pb/extra1.pbd
LIBRARY=http://www.mycompany.com/pb/extra2.pbd
COMMANDPARM="Eastern region"
APPLICATION=plugin_tree>
```

What's next

After defining the HTML page that displays your plug-in application, you need to set up the Web server.

Setting up the server

Setting up the server involves:

1 Specifying the MIME type

2 Putting the files on the server

Specifying the MIME type

Using the appropriate software for your Web server, register the **MIME type** for the PowerBuilder window plug-in. The MIME types you can use are:

For this plug-in	Register this MIME type
Standard PowerBuilder window plug-in	application/vnd.powerbuilder6
Secure PowerBuilder window plug-in	application/vnd.powerbuilder6-s

The file extension associated with the MIME type is PBD.

Your server documentation may use the term **content type** instead of MIME type.

Putting the files on the server

You need to copy PowerBuilder libraries and the HTML file to appropriate directories on your server.

Files	Location	What you do
HTML page	The HTML documents directory	Copy the HTML page to the directory specified in any URLs that link to the page
PBD files named in the SRC and LIBRARY attributes of the Embed element	The HTML documents directory or some other directory as appropriate	Copy the PBD files to the directories you specified in the Embed element attributes

About URLs

The URLs you specify in your HTML page are logical paths as defined by the Web server. On Windows, for example, the system path for your PBD might be:

```
C:\WEBSITE\HTDOCS\PB\PLUGIN_APP.PBD
```

If htdocs is defined as the server's documents directory, you can use the following relative path when specifying the SRC attribute of the Embed element. This path is relative to the documents directory:

```
pb/plugin_app.pbd
```

For the LIBRARY attribute, do not specify a relative URL for the PBD. You must specify a full URL. For example:

```
http://www.mycompany.com/pb/plugin_app.pbd
```

What's next

After building the plug-in application, creating its HTML page, and setting up the server, you need to make sure client workstations are set up to view the plug-in application.

Setting up users' workstations

Users who will view a Web page that includes a PowerBuilder window plug-in application need supporting software installed on their client workstation. They also need a connection to the Web server.

Required components

Each client workstation needs these components to view a Web page containing a PowerBuilder window plug-in application:

Component	Details
An Internet or intranet connection	Available within your corporation or from an Internet service provider
A Web browser that supports Netscape plug-ins	Available from the browser vendor. Examples are: ♦ Netscape Navigator Version 3.x or higher ♦ Microsoft Internet Explorer Version 3.x or higher **On Macintosh** Make sure the browser has sufficient memory allocation to accommodate the application's plug-in requirements
PowerBuilder deployment DLLs or shared libraries	Install the Deployment Kit as described in the deployment chapter for your platform in *Application Techniques* **On Windows** The PowerBuilder deployment DLLs belong in the application directory or in a directory on the system path. The PowerBuilder window plug-in DLL must know the location of the PowerBuilder deployment DLLs. To accomplish this, you can add the directory for the PowerBuilder deployment DLLs to the system path, or you can add the directory to the application path key for the browser (in the Windows registry) **On Macintosh** The PowerBuilder deployment shared libraries belong in the folder System Folder:Extensions:Powersoft 6.0

Component	Details
Standard or secure PowerBuilder window plug-in DLL or shared library	**On Windows** If it isn't there already, copy the NPPBA60.DLL file (standard) or NPPBS60.DLL file (secure) from the Internet Tools PLUGIN32 directory to the browser's plug-ins directory **On Macintosh** If it isn't there already, copy the PowerBuilder Window Plugin (standard) or PowerBuilder Secure Window (secure) shared library from the Internet Tools Netscape Plugins folder to the browser's plug-ins folder
Other files	If you didn't include image resources in the PBDs, copy them to the paths specified in the object properties if they aren't there already If the plug-in application connects to a database, set up client software for the DBMS FOR INFO For how to connect to a DBMS from the client machine, see *Connecting to Your Database*

Viewing the Web page and plug-in application

When the required software is installed, the user can view the Web page with the plug-in application.

When the user navigates to the URL of the Web page:

- The text on the page displays with a reserved space for the PowerBuilder window
- The client downloads the PBDs from the server
- The browser displays the child window within the Web page
- The user interacts with controls in the window and scripts for events execute
- The user navigates away from the page, which closes the window and causes the PowerBuilder DLLs or shared libraries to be unloaded from memory

CHAPTER 8

Using the DataWindow Plug-In

About this chapter

This chapter describes how to plan and deploy a Powersoft report in a Web page using the DataWindow plug-in.

Contents

Before you begin

This chapter assumes you are familiar with HTML and URLs and how a Web browser obtains pages from a Web server, and that you have access to a Web server.

About the DataWindow plug-in

The DataWindow plug-in lets you display a Powersoft report (PSR) on a Web page viewed in a browser that supports Netscape plug-ins.

A PSR file contains a report definition (source and object) as well as the data contained in the report when the PSR file was created. Because its data is saved with it, a PSR file does not require a database connection. But the data is static and cannot be refreshed.

Supported platforms

PowerBuilder provides the DataWindow plug-in for the following operating system platforms:

- Windows NT
- Windows 95
- Power Macintosh

Supported browsers

The DataWindow plug-in requires use of a Web browser that supports Netscape plug-ins, such as:

- Netscape Navigator Version 3.x or higher
- Microsoft Internet Explorer Version 3.x or higher

Security

There are no security issues for the DataWindow plug-in. It does not run any local applications and it does not write local files unless the user explicitly chooses to save the report locally.

The DataWindow plug-in displays PSR files only. PSR files are read-only; so there is no need for a secure version of the DataWindow plug-in.

Restrictions

The PSR *cannot* have the RichText presentation style.

How the DataWindow plug-in works

The DataWindow plug-in displays a PSR with its data and formatting as it was previewed and saved from the Report painter, DataWindow painter, DataWindow control, or DataStore. The PSR displays all the formatting in the DataWindow or report object, including rotated text, colored text, shading, or edit styles like checkboxes or radio buttons.

The data is a report—there is no data entry or database access.

A popup menu in the Web browser lets the client print the report or save it in several formats. If clients save the report as a local PSR file, they can view it in InfoMaker, which has more tools for searching, filtering, and sorting the data.

The DataWindow plug-in implements the Netscape Plug-in API and requires a browser that supports this API (for information, see "Supported browsers" on page 338).

Client-server interactions, in detail

This table describes in detail what happens between the client and server when the user views an HTML document containing a DataWindow plug-in.

DLLs and shared libraries
On Windows, the DataWindow plug-in is a DLL. On Macintosh, it is a shared library. (The DataWindow plug-in is not supported on UNIX.)

Step	What the client does	How the server responds
1	The Web browser requests the HTML document from the server	The server sends a header identifying the document's MIME type (text/html)
2	The browser receives the MIME type and prepares to receive an HTML document	The server sends the HTML document
3	The browser receives the HTML document and displays it	—
4	The browser recognizes the Embed element, reserves space for the plug-in on the page, and requests the PSR file from the server	The server sends a header identifying the PSR's MIME type (application/datawindow)
5	The browser receives the MIME type and prepares to receive a PSR file	The server sends the PSR file
6	The Web browser receives the PSR file	—
7	The browser searches its plug-in directory for the DLL that corresponds to MIME type application/datawindow	—

Step	What the client does	How the server responds
8	The Web browser loads the plug-in DLL and displays the PSR file	—

Requirements

Each client that will browse pages containing DataWindow plug-ins needs the DataWindow plug-in DLL or shared library installed on the local machine.

Name and location on different platforms

The name of the DataWindow plug-in DLL (on Windows) or shared library (on Macintosh) and where PowerBuilder installs it depends on your operating system platform.

Platform	DataWindow plug-in location
Windows	PB6\IT\PLUGIN32\NPDWE60.DLL
Power Macintosh	Powersoft 6.0 Folder:Internet Tools:Netscape Plugins: DataWindow Plugin

Developing and deploying a DataWindow plug-in

What you do

There are four main tasks involved in displaying a PSR in the DataWindow plug-in.

❖ **To display a PSR in the DataWindow plug-in:**

1. Save a PSR file.
2. Create an HTML page that embeds the DataWindow plug-in.
3. Configure the Web server by registering the appropriate content type (MIME type) and copying the HTML page and PSR files to appropriate directories.
4. On all client workstations, install the DataWindow plug-in DLL or shared library.

Resulting components

When everything is set up, the various computers will have these components:

Computer	Component
Server	MIME type application/datawindow registered for the file extension PSR
	HTML page with an Embed element for the PSR
	PSR file

Computer	Component
Client	DataWindow plug-in DLL or shared library in the browser's plug-ins directory

What's next

The rest of this chapter describes the four steps for displaying a PSR in the DataWindow plug-in:

- Saving a Powersoft report
- Creating an HTML page
- Setting up the server
- Setting up users' workstations

Saving a Powersoft report

Creating a PSR file

There are several ways that you or a user can create a PSR file.

❖ **To create a PSR file:**

- Do one of the following:
  - In the PowerBuilder DataWindow painter, save data as a PSR while previewing a DataWindow object
  - In InfoMaker or the PowerBuilder Report painter, mail a report using electronic mail
  - In a PowerBuilder application, call the SaveAs function and specify a SaveAsType of PSReport!

Resources for the PSR

A DataWindow or report object can display bitmap objects and custom pointers. These external resources must be made available on the client workstation. They are not automatically downloaded from the server.

Paths for external resources must be valid on the client workstation. In the DataWindow or Report painter, you can edit paths so that they specify directories relative to current directory instead of absolute paths that are unlikely to exist on the user's computer.

In an intranet environment, you can store external resources on a network drive that is accessible to all users. In Windows environments, a network path can name the network drive instead of using a mapped drive letter.

OLE objects and custom controls

OLE servers and custom controls must be installed and registered on each client workstation.

Restrictions

You *cannot* use a PSR file in the DataWindow plug-in whose original DataWindow or report had the RichText presentation style.

What's next

After saving a PSR file, you need to create the HTML page that displays it.

Creating an HTML page

You include a Powersoft report on a Web page with the Embed element. Element attributes specify the space allocated for the report and the report filename.

A sample Embed element might look like this:

```
<EMBED src=April_sales.psr WIDTH=370 HEIGHT=320>
```

Attributes of the Embed element

The Embed element is part of the HTML specification for plug-ins. For the DataWindow plug-in, you specify only standard HTML attributes.

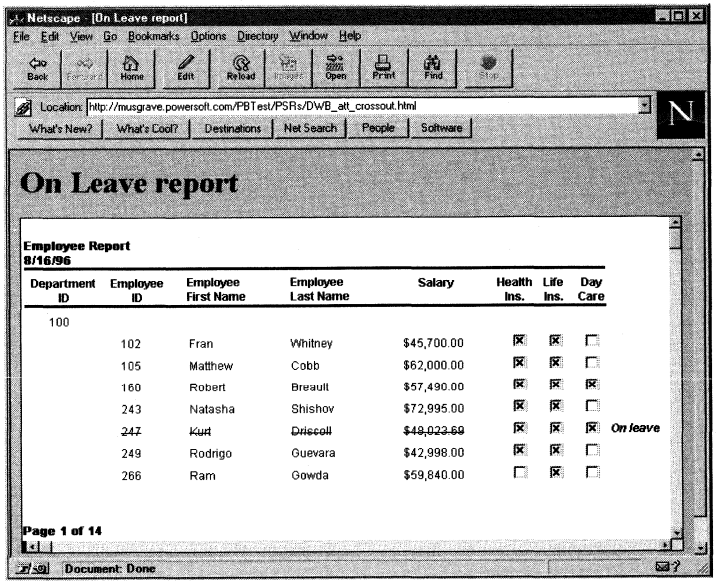
HTML attributes

HTML attributes name the file to be downloaded to the client and the space reserved for the plug-in on the Web page.

HTML attribute	Value
SRC	A URL identifying the object to be downloaded When the browser processes the Embed element, it requests the resource from the server and finds the DLL or shared library in its plug-ins directory that handles the content type For a DataWindow plug-in, the object is a PSR file
WIDTH	The width of the viewing window in pixels
HEIGHT	The height of the viewing window in pixels

Sample page

This sample page includes the DataWindow plug-in displaying a PSR file:



Here is the HTML code that produces this page. Note the use of the Embed element (shown in bold) to specify the PSR file.

Opening element

```
<HTML>
```

Document heading

```
<HEAD>
<TITLE>On Leave report</TITLE>
</HEAD>

<BODY>
```

H1 heading

```
<H1>On Leave report</H1>
```

Embed element in a paragraph

```
<P><EMBED src=DWB_att_crossout.psr WIDTH=680 HEIGHT =
350></P>
```

Closing elements

```
</BODY>
</HTML>
```

What's next

After defining the HTML page that displays your PSR file, you need to set up the Web server.

Setting up the server

Setting up the server involves:

1 Specifying the MIME type

2 Putting the files on the server

Specifying the MIME type

Using the appropriate software for your Web server, register the **MIME type** for the DataWindow plug-in. The MIME type is:

```
application/datawindow
```

The file extension associated with the MIME type is PSR.

Your server documentation may use the term **content type** instead of MIME type.

Putting the files on the server

You must copy the PSR and HTML files to appropriate directories on your server.

Files	Location	What you do
HTML page	The HTML documents directory or subdirectory	Copy the HTML page to the directory specified in any URLs that link to the page
PSR file named in the SRC attribute of the Embed element	The HTML documents directory or some other directory as appropriate	Copy the PSR file to the directory you specified in the Embed element attribute

About URLs

The URLs you specify in your HTML page are logical paths as defined by the Web server. For example, the system path for your PSR might be:

```
C:\WEBSITE\HTDOCS\PB\DWB_ATT_CROSSOUT.PSR
```

If htdocs is defined as the server's documents directory, the logical path for the URL would be relative to the documents directory:

```
pb/dwb_att_crossout.psr
```

What's next

After saving the PSR, creating an HTML page, and setting up the server, you need to make sure client workstations are set up to view the page containing the DataWindow plug-in.

Setting up users' workstations

Users who will view a Web page that includes a DataWindow plug-in needs supporting software installed on their client workstation. They also need a connection to the Web server.

Required components

Each client workstation needs these components to view a Web page containing a DataWindow plug-in:

Component	Procedure
An Internet or intranet connection	Available within your corporation or from an Internet service provider
A Web browser that supports Netscape plug-ins	Available from the browser vendor. Examples are: ♦ Netscape Navigator Version 3.x or higher ♦ Microsoft Internet Explorer Version 3.x or higher **On Macintosh** Make sure the browser has sufficient memory allocation to accommodate the application's plug-in requirements
DataWindow plug-in DLL or shared library	**On Windows** If it isn't there already, copy the NPDWE60.DLL file from the Internet Tools PLUGIN32 directory to the browser's plug-ins directory **On Macintosh** If it isn't there already, copy the DataWindow Plugin shared library from the Internet Tools Netscape Plugins folder to the browser's plug-ins folder
Other files	If the original DataWindow object or report included bitmap objects or custom pointers from external files, you need to copy those files to the client workstation. The path on the client workstation must correspond to the path saved in the PSR If the original included OLE objects or custom controls, you must install and register the OLE server or custom control on the client workstation. The path for an OLE object must be valid on the client

Viewing the Web page and the PSR

While viewing a PSR in the DataWindow plug-in, the user can:

- Navigate through the data in the PSR using scrollbars
- Change values in rows and columns if the source of the PSR was an editable DataWindow object (but because there is no database connection, data cannot be updated in a database)
- Right-click on the PSR to display a popup menu for saving, printing, and navigating

InfoMaker users who save the report from the Web page as a local PSR file can open that PSR file in InfoMaker to search, filter, and sort the data.

PART 4

PowerBuilder Window ActiveX

This part describes how to use the PowerBuilder window ActiveX component of the Internet Tools.

CHAPTER 9

Using the PowerBuilder Window ActiveX

About this chapter

This chapter describes how to use the PowerBuilder window ActiveX.

Contents

About the PowerBuilder window ActiveX

The PowerBuilder window ActiveX lets you display a PowerBuilder child window on Web pages viewed in a browser that supports ActiveX.

Features

The PowerBuilder window can include all the familiar controls, including DataWindows, OLE objects, OCX (ActiveX) controls, and TreeView controls. You can also open other (popup or response) windows from the child window.

As the user interacts with controls in the windows, scripts for the controls' events are executed just as they are in standalone PowerBuilder applications. Additionally, you can call PowerBuilder functions and react to PowerBuilder events by coding VB Script or JavaScript within the HTML page.

Database access by the PowerBuilder window ActiveX application occurs using the client's locally defined database connections.

The objects in the application can be contained in one or more PowerBuilder Dynamic Libraries (PBDs).

Supported platforms

PowerBuilder provides versions of the PowerBuilder window ActiveX for the following operating system platforms:

- Windows 95
- Windows NT 4.0

Supported browsers

The PowerBuilder window ActiveX requires use of a browser that supports ActiveX, such as:

- Microsoft Internet Explorer Version 3.x (or higher)
- Netscape Navigator Version 3.x using the NCompass ScriptActive plug-in

Kinds of applications that work with the PowerBuilder window ActiveX

HTML forms provide some user interaction via a limited user interface. The PowerBuilder window ActiveX takes you beyond HTML. You can present a fully developed application window with a rich user interface design to a Web page. You can access data sources defined on the client workstation. You can use PowerBuilder's distributed capabilities to access an application server and its data sources.

Examples

Your application might be:

- A data analysis window with master and detail DataWindows
- A user interface design that lets the user make choices using TreeView, ListView, and PictureListBox controls
- A data entry form that uses a distributed application to process the data on the server
- A data entry form that uses the client's defined database connection (either network or local) to process the data on the client machine
- A window that uses PowerBuilder's OLEControl control to display an ActiveX control

How the PowerBuilder window ActiveX works

The PowerBuilder window ActiveX displays a PowerBuilder child window inside a fixed space reserved on the Web page. The user can interact with the controls on the page, and the PowerBuilder scripts for the window and its controls can execute any PowerBuilder code. When the user switches to another Web page, the PowerBuilder window is closed and the PowerBuilder DLLs are unloaded from memory.

The PowerBuilder window ActiveX is included on an HTML page using the HTML Object element. It names one or more PBDs that contain PowerBuilder objects, the name of the child window object that is displayed on the page, and (optionally) the PowerBuilder Application object.

Security

Unsecured applications can access local files and can run local applications. These types of processing are often undesirable in an uncontrolled Web environment but may be perfectly acceptable on a corporate intranet where access is controlled. There are two versions of the PowerBuilder window ActiveX: one secure (PBRX60.OCX) and the other unsecure (PBRXS60.OCX). The secure version is extremely restricted, with no access to the client workstation.

Requirements

The PowerBuilder window ActiveX uses the PowerBuilder virtual machine as well as the ActiveX itself to provide a full range of PowerBuilder functionality.

Each client that will browse pages with the PowerBuilder window ActiveX needs supporting software installed on the local machine:

- PowerBuilder virtual machine (PBVM60.DLL)
- PowerBuilder window ActiveX

The PowerBuilder window ActiveX is especially useful in an intranet application where you have control over the setup of client machines.

Developing and deploying a PowerBuilder window ActiveX application

What you do

There are four main tasks involved in developing a PowerBuilder window ActiveX application.

❖ **To create and deploy a PowerBuilder window ActiveX application:**

1. Create, test, and build the PowerBuilder application.
2. Create an HTML page that includes your PowerBuilder application window.
3. Configure the Web server by copying the HTML page and PBD files for the application to appropriate directories.
4. On all client workstations, install the PowerBuilder window ActiveX control and the PowerBuilder deployment DLLs.

Resulting components

When everything is set up, the various computers will have the following components:

Computer	Component
Server	HTML page with an Object element for the window (or report) and PBD
	PowerBuilder window ActiveX installed and registered (optional)
	One or more PBDs containing your child windows and other PowerBuilder objects
Client	PowerBuilder window ActiveX installed and registered
	PowerBuilder deployment DLLs or shared libraries installed using the PowerBuilder Deployment Kit for your platform
	Directory for the PowerBuilder deployment DLLs listed in the system path

Computer	Component
	Microsoft DLLS: MFC042D.DLL MFC42D.DLL MSVCRTD.DLL URL.DLL URLMON.DLL
	Any additional software required by the PowerBuilder window ActiveX application, such as database connection software or OLE servers and ActiveX controls

Creating the PowerBuilder application

The starting point of your PowerBuilder window ActiveX application is a child window, which is displayed within a Web page. In the window, you can include controls and write scripts for events. Your scripts can open other windows, read and write files, and run other programs on the client machine. The application can contact a PowerBuilder server application to initiate a distributed session.

Defining the starting window in the Window painter

The application you design for use with the PowerBuilder window ActiveX can be much the same as other PowerBuilder applications you develop. However, there are some restrictions and considerations to keep in mind. This section discusses:

- Window management
- Scripts and variables
- Data access
- External files

Window management

Initial child window Your initial window needs to be a child window that lives in the browser frame.

You cannot:

- Have a menu for the child window
- Open another child window from the initial window

You can:

- Include a title bar on the child window

  But you should not use a control menu, maximize box, or minimize box on that window.
- Open popup and response windows from the child window (but not main or MDI windows)

Closing windows When the client browses to another Web page, the child window on the current Web page is closed. But other windows remain open unless your application closes them. You must close them in the child window's Close or CloseQuery events.

Do not try to stop the child window from being closed in the CloseQuery event by setting a return value. You can't prevent the browser from changing to another page and removing the window from view. If the user returns to the page, another instance of your application will be started.

Objects

Objects in your PBDs Your PowerBuilder window ActiveX application has access to all objects in the PBDs. This includes, functions, structures, and user objects.

System objects Your PowerBuilder window ActiveX application has access to system objects that PowerBuilder instantiates, such as the SQLCA Transaction object and the Message object.

Application object Your PowerBuilder window ActiveX application can access global variables, which are properties of the Application object.

PowerBuilder requires that you have a current Application object during development, but only the application Open event and global variables are used in your application. Any other scripts in the Application object will only be used during testing within PowerBuilder.

Scripts and variables

Scripts for application setup If you specify the PBAPPLICATION parameter in the HTML, the PowerBuilder window ActiveX executes the application Open event before opening the child window. You can use this event to establish a database connection and initialize global variables.

The HTML specifies the window to open
Do not open windows from within the application Open event.

If you don't specify the PBAPPLICATION parameter in the HTML, all application setup must be done in the child window, including connecting to the DBMS. The first scriptable events to occur are the constructors for the controls in the window. Then the Open event for the window occurs.

Note that the Activate event does not occur for a child window, so don't do application setup there.

Global variables The PowerBuilder window ActiveX application can access global variables.

Referencing the initial window You can't reference the initial child window by name in your scripts. That's because PowerBuilder doesn't create a variable to hold the instance of it. (By contrast, when you instantiate a window yourself by coding the Open function, you always place the instance in a variable that you can then reference.)

As a result, the following code produces an error at execution time (because the window variable w_mychild does not exist):

```
// This code produces an execution time error:
   w_mychild.title = "The initial window"
```

But you can code:

```
// In the child window itself:
   this.title = "The initial window"
   // or
   title = "The initial window"

// In controls of the child window:
   parent.title = "The initial window"
```

Data access

If your application accesses a DBMS, each client must have a connection to the data source. The connection must be defined on the client's machine, not the server. The data source might be a local or a network DBMS.

FOR INFO For information about how to connect to a DBMS from the client machine, see *Connecting to Your Database*.

Remember that the Constructor events for controls occur before the Open event of the window. That mean if you connect to the DBMS in the window's Open event, the Constructor events can't get data. You can get data for controls in the window's Open event or you can post events from the Constructor events or the window's Open event.

Paths for external resources

Paths you specify for files must be valid on the client workstation.

If your application uses images as external files, the images must be available at the path specified in the PowerBuilder object. Instead of using external files, you can build image resources into PBDs.

If your application reads or writes local files, the path for those files must be valid on each client's machine.

On Windows In Windows environments, if a path refers to a network drive by mapped drive letter, all clients must use the same drive letter that the application uses. As an alternative, specify the server name in the path instead.

For example, both of these paths are valid when o: is mapped to \\marketing\drive. However, the second path that uses a server name is more likely to *remain* valid:

```
O:\pbapps\connect.bmp
```

```
\\marketing\drive\pbapps\connect.bmp
```

Defining the starting window in the Window painter

You use the Window painter to create the starting window, as you do for any window you create in PowerBuilder.

❖ **To create your application's starting window:**

1. In the Window painter, create a new window.
2. On the window's property sheet, set the window type to Child.
3. Add other controls as desired.
4. Write scripts for events of the window and controls.

❖ **To convert an existing application to run as a PowerBuilder window ActiveX:**

1. Change the type of the opening window to child.
2. Move application setup code from the application's Open event or MDI frame events to the child window's Open event.

Depending on the application's design, you may need to redesign how it opens other windows.

About child windows

Child windows cannot have menus. They are never considered the active window therefore, the Activate event is never triggered. They can have title bars and can be minimizable, maximizable, and resizable. However, in the PowerBuilder window ActiveX environment, the child window is always restricted to the space allotted by the WIDTH and HEIGHT attributes specified on the Web page:

- Maximizing causes the window to fill the space allotted by the WIDTH and HEIGHT attributes
- Minimizing displays the window's icon and title at the bottom of the ActiveX control's allotted space
- If the child window is resizable, the user can drag the borders to make the window smaller (but not larger) than the allotted space

As a result, it is not useful to allow minimizing, maximizing, or resizing of the child window.

Testing the application in PowerBuilder

Before you try your application in a client browser, you can test it in PowerBuilder by defining a main window that opens the child window.

❖ **To test your PowerBuilder window ActiveX application:**

1. In the Window painter, create a new window whose type is main (the default).
2. Write a script for the Open event that opens the child window:

```
Open(w_child)
```

3. For convenience, you can set appropriate window sizes in the Window painter:
   - Make the main window large enough to display the child
   - Position the child window in the upper-left corner (you set the position on the Position tab of the property sheet)
4. Run the test window by clicking the Run Window icon.

Debugging in PowerBuilder

To use the PowerBuilder debugger, you need to run an application instead of a single window. Define an Application object with a script that opens the main window. Then you can use the Run or Debug commands to test the application.

Organizing objects in PBLs

In the Web environment, file size is important. Clients are downloading your application each time they run it (unless it happens to be cached because they ran it recently).

Before you build your application, use the Library painter to organize the objects your application uses in PBLs, which will be the sources for the PowerBuilder window ActiveX application's PBDs. Follow these suggestions:

- To minimize file size, include only objects the application uses. Remove any objects that are not needed.
- Do include any objects that are dynamically created, such as DataWindows used in DataStores or assigned dynamically to DataWindow controls and Proxy objects.
- Do include ancestor objects.

Using PowerBuilder resource files

Several controls can use external files for images. These include PictureListBox, TreeView, Picture, and PictureButton controls, pointers, and bitmap objects in DataWindow objects. If your application uses external files for the images, it is unlikely that the client has the same images on the same system path.

Instead of finding some way to install the pictures on client machines, you can use one or more PowerBuilder resource (PBR) files so that the images are built into the PBDs. The resulting PBDs will be larger but self-contained.

Because you are building a PBD, not an executable, you do not need to include DataWindow objects in the resource file. All PowerBuilder objects in the source PBL are included in the resulting PBD.

Images and other resources on a network
Instead of building the files into the PBD, you can put the files in a generally accessible network directory. But the path to the files must be identical to the path named in the PowerBuilder objects. In the Windows environment, this means each client must use the same drive letter to map the network drive (or you can specify the server name in the path).

❖ **To define a PBR file:**

1 Open the File Editor (SHIFT+F6) or some other text editor and create a new file with the extension .PBR.

2 List each image or other resource on its own line. List the path and filename exactly as it is named in the object property sheet or script.

Shortcut
Look at the object's property sheet and use EditCopy to copy the filename to the clipboard. Then paste it into the editor.

---

3 Save the file.

If your application includes several PBLs each with objects using their own resources, you will want to create a PBR file for each PBL. The PBR file will list filenames for resources that are used in one PBL.

Building the PBDs

Build your dynamic libraries (PBDs) using either the Library painter or the Project painter. Do the following:

- **Deselect Machine code** Your PowerBuilder window ActiveX application must have PBDs, not machine code DLLs.

- **Specify a PBR file for each PBD** If the objects in the PBL use resources that you have listed in a PBR file, put the PBR name in the Resource File Name textbox.

FOR INFO For instructions on defining a PBR file and building your dynamic libraries (PBDs), see the chapter on creating an executable in the *PowerBuilder User's Guide*.

What's next

After creating and building the PowerBuilder PBDs for your PowerBuilder window ActiveX application, you need to create the HTML page that displays it.

Creating an HTML page

To include a PowerBuilder window on a Web page, you use the Object element. Element attributes specify the class ID for the PowerBuilder window ActiveX, the space allocated for the window, the name of the PBD, and the name of the child window in the PBD.

A sample Object element might be:

```
OBJECT NAME="PBRX1" WIDTH=225 HEIGHT=83
CLASSID="CLSID:CEC58653-C842-11CF-A6FB-00805FA8669E"
PARAM NAME="_Version" VALUE="65536"
PARAM NAME="_ExtentX" VALUE="5962"
PARAM NAME="_ExtentY" VALUE="2164"
PARAM NAME="_StockProps" VALUE="0"
PARAM NAME="PBWindow" VALUE="w_helloworld"
PARAM NAME="LibList"
VALUE="http://www.company.com/rknnt.pbd;"
PARAM NAME="PBApplication" VALUE="rknnt"
/OBJECT
```

Attributes of the Object element

The Object element is part of the HTML specification for ActiveX controls. It defines several standard attributes, and PowerBuilder defines additional attributes.

HTML attributes

HTML attributes specify the class ID, the name, and the space reserved for the PowerBuilder window ActiveX on the Web page:

HTML attribute	Value
NAME	Name of the object when referenced in code or when submitted as part of a form
CLASSID	The class ID of the registered ActiveX control. The syntax is: **CLSID:***class_id*
CODEBASE	A URL identifying the location of the OCX or CAB file to be downloaded if the client machine does not contain the PowerBuilder window ActiveX (the client machine must still have the PowerBuilder virtual machine and any other required DLLs on the system path)

HTML attribute	Value
WIDTH	The width of the viewing window in pixels
HEIGHT	The height of the viewing window in pixels

The WIDTH and HEIGHT attributes define the maximum width and height of the child window. If the child window is resizable, the user can make it smaller than the specified size, but not larger.

Param elements

To specify properties for the PowerBuilder window ActiveX, include Param elements. Param elements let you identify the window object that starts your application, the Application object, additional libraries, and additional parameters to pass to the PowerBuilder window ActiveX. This table lists the PowerBuilder-specific Param elements:

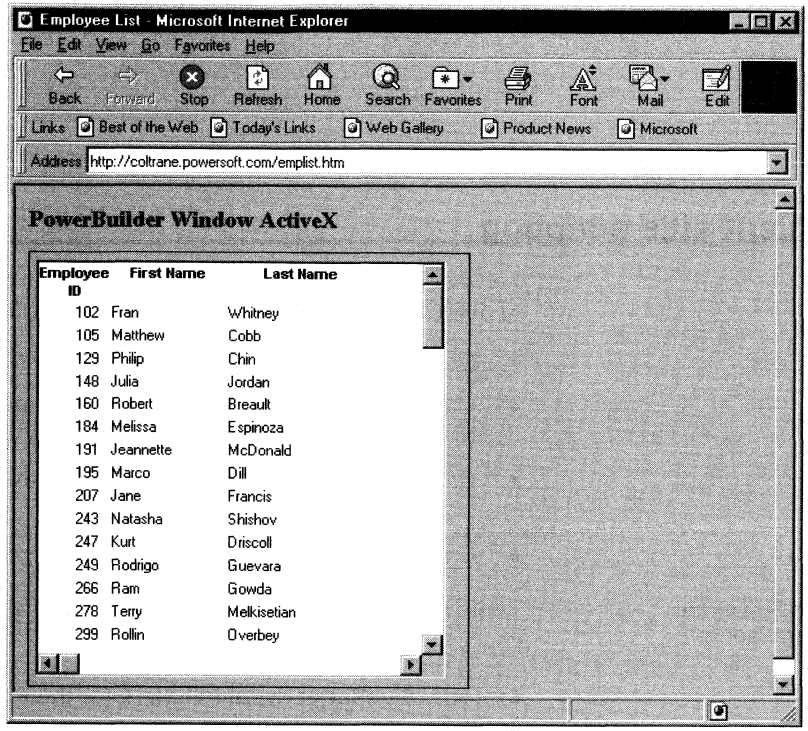
Property	Value
PBWINDOW	The class name of the child window in the PBD
LIBLIST	A list of PowerBuilder dynamic libraries (PBD files) required by the application. Separate multiple entries with a semicolon
PBAPPLICATION (optional)	The PowerBuilder Application object
DISPLAYERRORS (optional)	Boolean indicating whether to display execution time errors
COMMANDPARM (optional)	A string you want to pass to your window. To access this string from within your window, call the CommandParm function

Coding the Object element

To minimize coding errors, use an ActiveX-aware HTML editor (such as the PowerSite editor, the ActiveX Control Pad, or Front Page) when coding an Object element and its parameters.

Basic page

This sample page includes the PowerBuilder window ActiveX:



Here is the HTML code that produces this page (note the use of the Object element to specify the PowerBuilder window ActiveX):

```
<HTML>
<HEAD>
<TITLE>Employee List</TITLE>
</HEAD>
<BODY>
<H1>PowerBuilder window ActiveX</H1>
<P>
<OBJECT ID="PBRX1" NAME="PBRX1" WIDTH=357 HEIGHT=269
CLASSID="CLSID:CEC58653-C842-11CF-A6FB-00805FA8669E">
<PARAM NAME="_Version" VALUE="65536">
<PARAM NAME="_ExtentX" VALUE="9440">
<PARAM NAME="_ExtentY" VALUE="7112">
<PARAM NAME="_StockProps" VALUE="0">
```

```
<PARAM NAME="PBWindow" VALUE="w_emplist">
<PARAM NAME="LibList"
VALUE="http://www.company.com/rknnt.pbd;">
<PARAM NAME="PBApplication" VALUE="rknnt">
</OBJECT>
</BODY>
</HTML>
```

Client-side scripting

You can interact with the window displayed in a PowerBuilder window ActiveX by adding JavaScript or VBScript to the HTML page. You can:

- Code event handlers that respond to events that occur in the window
- Call PowerScript functions to obtain pointer information, print, set redraw, or set a timer
- Call the InvokePBFunction function to invoke a user-defined window function
- Call the TriggerPBEvent function to trigger a user event on the window

Coding event handlers

Your HTML page can contain JavaScript or VBScript event handlers for the PowerBuilder window ActiveX.

❖ **To code JavaScript event handlers for the PowerBuilder window ActiveX:**

1 Insert the PowerBuilder window ActiveX into the HTML page, specifying all necessary properties.

```
<OBJECT NAME="PBRX1" WIDTH=225 HEIGHT=83
CLASSID="CLSID:CEC58653-C842-11CF-A6FB-
00805FA8669E">
<PARAM NAME="_Version" VALUE="65536">
<PARAM NAME="_ExtentX" VALUE="5962">
<PARAM NAME="_ExtentY" VALUE="2164">
<PARAM NAME="_StockProps" VALUE="0">
<PARAM NAME="PBWindow" VALUE="w_helloworld">
<PARAM NAME="LibList"
VALUE="http://www.company.com/rknnt.pbd;">
<PARAM NAME="PBApplication" VALUE="rknnt">
</OBJECT>
```

2 Within the heading of the HTML page, code a function to be called when the event occurs. This example simply displays the arguments to the Clicked event:

```
function wasClicked(flags, xpos, ypos) {
   document.buttonForm.passedFlags.value = flags;
   document.buttonForm.passedXPos.value = xpos;
   document.buttonForm.passedYPos.value = ypos;
}
```

3 Within the body of the HTML page, code an event handler that calls the function when the event occurs:

```
<SCRIPT LANGUAGE="JavaScript" FOR="PBRX1"
Event="Clicked(flags, xpos, ypos)">
<!--
wasClicked(flags, xpos, ypos);
-->
</SCRIPT>
```

Coding style

Alternatively, you can omit the function call, placing all code within the event handler, as shown next.

❖ **To code VBScript event handlers for the PowerBuilder window ActiveX:**

1 Insert the PowerBuilder window ActiveX into the HTML page, specifying all necessary properties:

```
<OBJECT NAME="PBRX1" WIDTH=225 HEIGHT=83
CLASSID="CLSID:CEC58653-C842-11CF-A6FB-
00805FA8669E">
<PARAM NAME="_Version" VALUE="65536">
<PARAM NAME="_ExtentX" VALUE="5962">
<PARAM NAME="_ExtentY" VALUE="2164">
<PARAM NAME="_StockProps" VALUE="0">
<PARAM NAME="PBWindow" VALUE="w_helloworld">
<PARAM NAME="LibList"
VALUE="http://www.company.com/rknnt.pbd;">
<PARAM NAME="PBApplication" VALUE="rknnt">
</OBJECT>
```

2 Within the body of the HTML page, code an event handler that processes the event. This example simply displays the arguments to the Clicked event:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Sub PBRX1_Clicked(flags, xpos, ypos)
   document.buttonForm.passedFlags.value = flags
   document.buttonForm.passedXPos.value = xpos
   document.buttonForm.passedYPos.value = ypos
end sub
-->
</SCRIPT>
```

Calling PowerScript functions

The PowerBuilder window ActiveX allows you to call certain PowerScript functions on the window displayed in the Active control:

- **PointerX** Returns the distance from the left edge of the window to the pointer
- **PointerY** Returns the distance from the top of the window to the pointer
- **Print** Prints the window
- **SetRedraw** Turns on or off automatic redrawing of the window after every change
- **Timer** Causes the window's Timer event to occur repeatedly at the specified interval

FOR INFO For more information on these functions, see the *PowerScript Reference.*

As with all ActiveX controls, the PowerBuilder window ActiveX provides an AboutBox function, which you can call to see information about the control.

❖ **To use JavaScript to call a PowerScript function:**

1 Code a function to call the PowerScript function. This example calls the PowerScript Timer function:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
var cumSeconds = 0;
function setPBTimer( f ) {
var li_return
var li_interval
li_interval = parseInt(f.timerInterval.value);
li_return = PBRX1.Timer(li_interval);
if (li_return != 1) {
   alert("Set Timer failed");
```

```
}
}
//-->
/SCRIPT
```

2 Code a function, anchor, or button that calls the function. This example uses a button on a form to call the function defined above, which resets the timer interval:

```
<FORM NAME="clockForm">
<P>Timer Interval:
<INPUT TYPE=Text NAME="timerInterval" Size="5">
<P><INPUT TYPE=BUTTON VALUE="Set Timer"
ONCLICK="setPBTimer(this.form)">
</FORM>
```

❖ **To use VBScript to call a PowerScript function:**

1 Code a function to call the PowerScript function. This example calls the PowerScript Timer function:

```
<SCRIPT LANGUAGE="VBScript">
<!--
dim cumSeconds
cumSeconds = 0
Sub pbSetTime()
  dim li_return
  dim li_interval
  li_interval = clockForm.timerInterval.value
  li_return = pbrx1.Timer(li_interval)
  if li_return  1 THEN
    msgBox "Set Timer failed"
  end if
  end sub
//-->
</SCRIPT>
```

2 Code a function, anchor, or button that calls the function. This example uses a button on a form to call the function defined above, which resets the timer interval:

```
<FORM NAME="clockForm">
<P>Timer Interval:
<INPUT TYPE=Text NAME="timerInterval" Size="5">
<HR>
<P>Mirror of PB Time:
<INPUT TYPE=Text NAME="pbTime" Size="8">
```

```
<HR>
<P>INPUT TYPE=BUTTON VALUE="Set Timer"
onClick="call pbSetTime()">
</FORM>
```

Calling user-defined functions

The PowerBuilder window ActiveX provides the InvokePBFunction function, which you can use to call a user-defined window function.

VBScript and JavaScript differ

If your user-defined functions contain arguments and you are using JavaScript, you must use SetArgElement to specify the arguments; you cannot specify the arguments explicitly in the InvokePBFunction function.

❖ **To code JavaScript that invokes a user-defined function:**

1 Define window functions as needed.

2 Code a JavaScript function that calls the InvokePBFunction function, specifying the user-defined function to invoke. This example initializes arguments and calls the of_args window function:

```
function invokeFunc(f) {
  var retcd;
  var rc;
  var numargs;
  var theFunc;
  var theArg;
  retcd = 0;
  numargs = 1;
  theArg = f.textToPB.value;
  PBRX1.SetArgElement(1, theArg);
  theFunc = "of_args";
  retcd = PBRX1.InvokePBFunction(theFunc, numargs);
  rc = parseInt(PBRX1.GetLastReturn());
  if (rc != 1) {
    alert("Error. Empty string.");
  }
  PBRX1.ResetArgElements();
}
```

3 Code a function, anchor, or form button that invokes your JavaScript function:

```
<P>INPUT TYPE=BUTTON VALUE="Invoke Func"
ONCLICK="invokeFunc(this.Form)">
```

Defining arguments in JavaScript
When coding in JavaScript, define function and event arguments by calling the SetArgElement function.

❖ **To code VBScript that invokes a user-defined function:**

1 Define window functions as needed.

2 Code a VBScript function that calls the InvokePBFunction function, specifying the user-defined function to invoke. This example initializes arguments and calls the of_args window function:

```
Sub invokeFunction_OnClick()
  Dim retcd
  Dim myForm
  Dim args(1)
  Dim rc
  Dim numargs
  Dim theFunc
  Dim rcfromfunc
  retcd = 0
  numargs = 1
  rc = 0
  theFunc = "of_args"
  Set myForm = Document.buttonForm
  args(0) = buttonForm.textToPB.value
  retcd = PBRX1.InvokePBFunction(theFunc,
    numargs, args)
  rc = PBRX1.GetLastReturn()
  if rc  1 then
    msgbox "Error. Empty string."
  end if
  PBRX1.ResetArgElements()
end sub
```

3 Code a function, anchor, or form button whose click invokes your VBScript function:

```
<P><INPUT TYPE=BUTTON Name=invokeFunction
VALUE="Invoke Function">
```

Calling user events

The PowerBuilder window ActiveX provides the TriggerPBEvent function, which you can use to call a user event on the window.

❖ **To code JavaScript that triggers a user event:**

1 Define user events on the window as needed.

2 Code a function to call the user event. This example initializes arguments and calls the ue_args window function:

```
function triggerEvent(f) {
  var retcd;
  var rc;
  var numargs;
  var theEvent;
  var theArg;
  retcd = 0;
  numargs = 1;
  theArg = f.textToPB.value;
  PBRX1.SetArgElement(1, theArg);
  theEvent = "ue_args";
  retcd = PBRX1.TriggerPBEvent(theEvent, numargs);
  rc = parseInt(PBRX1.GetLastReturn());
  if (rc != 1) {
    alert("Error. Empty string.");
  }
  PBRX1.ResetArgElements();
}
```

3 Code a function, anchor, or form button that calls the TriggerPBEvent function:

```
<P><INPUT TYPE=BUTTON VALUE="Trigger Event"
ONCLICK="triggerEvent(this.Form)">
```

❖ **To code VBScript that triggers a user event:**

1 Define user events on the window, as needed.

2 Code a function to call the user event. This example initializes arguments and calls the ue_args window function:

```
Sub TrigEvent_OnClick( )
  Dim retcd
  Dim myForm
  Dim args(1)
  Dim rc
  Dim numargs
  Dim theEvent
  retcd = 0
  numargs = 1
```

```
    rc = 0
    theEvent = "ue_args"
    Set myForm = Document.buttonForm
    args(0) = buttonForm.textToPB.value
    retcd = PBRX1.TriggerPBEvent(theEvent,
      numargs, args)
    rc = PBRX1.GetLastReturn()
    if rc  1 then
      msgbox "Error. Empty string."
    end if
    PBRX1.ResetArgElements()
  end sub
```

3 Code a function, anchor, or form button whose click invokes your VBScript function:

```
<P><INPUT TYPE=BUTTON NAME=TrigEvent
VALUE="Trigger Event">
```

Setting up the server

Setting up the server involves placing files on the server.

You need to copy PowerBuilder libraries, the PowerBuilder window ActiveX module, and the HTML file to appropriate directories on your server:

Files	Location
HTML page	The HTML documents directory Copy the HTML page to the directory specified in any URLs that link to the page
PBD files named in the LibList attributes of the Param element	A directory named in the application path for the Web server application or any directory on the system path
PowerBuilder window ActiveX module	The HTML documents directory or some other directory as specified in the Object element's CODEBASE attribute

What your users need

Users who will view the Web page that includes a PowerBuilder window ActiveX need supporting software installed on their client workstation and a connection to the Web server:

Component	Details
An Internet or intranet connection	Available within your corporation or from an Internet service provider
A Web browser that supports ActiveX	Available from the browser vendor. Examples are Netscape Navigator 3.x (or higher) and Microsoft Internet Explorer 3.x (or higher) using the NCompass ScriptActive plug-in
PowerBuilder deployment DLLs	Install the Deployment Kit as described in the deployment chapter for your platform in *Application Techniques* The PowerBuilder deployment DLLs belong in the application directory or in a directory on the system path. The PowerBuilder window ActiveX *must be able to access* the PowerBuilder virtual machine (and any other required deployment DLLs). To accomplish this, you can add the directory for the PowerBuilder deployment DLLs to the system path, or you can place the deployment DLLs in the Windows\System directory
PowerBuilder window ActiveX module	Copy the file PBRX60.OCX to the client workstation and register. Alternatively, you can add the CODEBASE attribute to the Object element, which causes the browser to download and register it when the page is loaded
Other files	If you didn't include image resources in the PBDs, copy them to the paths specified in the object properties if they aren't there already If the PowerBuilder window ActiveX application connects to a database, set up client software for the DBMS FOR INFO For how to connect to a DBMS from the client machine, see *Connecting to Your Database*

Viewing the Web page and PowerBuilder window ActiveX application

When the required software is installed, the user can view the Web page with the PowerBuilder window ActiveX application.

When the user navigates to the URL of the Web page:

- The text on the page displays with a reserved space for the PowerBuilder window
- The client downloads the PBDs from the server
- The browser displays the child window within the Web page
- The user interacts with controls in the window and scripts for events execute
- The user navigates away from the page, which closes the window and causes the PowerBuilder DLLs or shared libraries to be unloaded from memory

Events for the PowerBuilder window ActiveX

The PowerBuilder window ActiveX can respond to certain events occurring within the child window. These are outbound events, which execute first within the PowerBuilder window and then in the PowerBuilder window ActiveX. You can add JavaScript or VB Script code that responds to these events. The events are listed in the following table:

Event	Occurs
Activate	Just before the window becomes active
Clicked	When the user clicks in an unoccupied area of the window (any area with no visible, enabled object)
Close	When the window is closed
Deactivate	When the window becomes inactive
DoubleClicked	When the user double-clicks in an unoccupied area of the window (any area with no visible, enabled object)
Hide	Just before the window is hidden
Key	When the user presses a key and the insertion point is not in a line edit
MouseDown	When the user presses the left mouse button in an unoccupied area of the window (any area with no visible, enabled object)
PBMouseMove	When the pointer is moved within the window
PBMouseUp	When the user releases the left mouse button in an unoccupied area of the window (any area with no visible, enabled object)
RButtonDown	When the user presses the right mouse button in an unoccupied area of the window (any area with no visible, enabled object)
Resize	When the user or a script opens or resizes a window
Show	When a script executes the Show function for this window (the event occurs just before the window is displayed)
SystemKey	When the insertion point is not in a line edit and the user presses ALT or ALT plus another key
Timer	When a specified number of seconds elapses after the Timer function has been called

Index

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W