Welcome to the

ProvideX Web Server Reference …

Introduction

The **ProvideX Web Server Reference** contains technical information to help you set up and use the ProvideX Web Server. The reference is intended primarily as a resource for application programmers and analysts. It's designed to help you with your ProvideX Web Server's installation, configuration and associated web-based internet programming. Please note that ProvideX Web Server version 1.2 must be used with ProvideX 4.12C or higher.

Before you begin, you'll find it helpful to familiarize yourself with the **Glossary** and list of descriptive **Conventions**, next.

Glossary of Common Terms

The chart below lists some of the naming conventions used in *The ProvideX Web Server Reference*. In addition, common internet acronyms are expanded to their full terms. See any standard browser manual for detailed definitions of internet acronyms.

Term/Acronym	**In this manual:**
base launcher	is the basic ProvideX Web Server launcher or engine. It's responsible for reading the configuration and launching the individual web servers (port monitors).
CGI	*Common Gateway Interface*
client/browser	is a browser or query engine sending a request for information or tasks.
cookie	is a variable=data pair you can use to transfer status and tracking information from one page to another.
DNS	*Domain Name Server*
FTP	*File Transfer Protocol*
GMT	*Greenwich Mean Time*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
task handler	is our ProvideX Web Server application that processes requests from the port monitor and returns completed results to the port monitor. (Task handlers are also referred to as subservers.) See **Task Handlers, p. 6**.
port monitor	is an individual web server that monitors a given port or TCP/IP socket. The terms "port monitor", "web server" and "server" are sometimes used interchangeably in this manual and in the ProvideX Web Server dialogue box prompts on-screen. See **Port Monitors, p. 6**.
SMTP	*Simple Mail Transfer Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TCP/IP Socket	is the number you assign to the port or socket. The typical default socket for a web server number is 80. Each port monitor must have a unique socket or port number.
URL	Uniform Resource Location request. Use standard URL coding conventions for requests to the Web Server. Requests to run ProvideX programs follow standard CGI coding conventions.
Web Server	*case-sensitive, mixed case* is our *ProvideX Web Server* application. Any other usage *(case insensitive)* is a port monitor. The terms "port monitor", "web server" and "server" are sometimes used interchangeably in this manual and in the Web Server dialogue boxes on-screen.

*The word **application** in this manual is a literal term meaning use/usage. An application can be as simple as a function, a few lines of code, or a subroutine controlling a process (e.g. creating a web page) in a single program, or it can be a complex series of programs (e.g. a lead program with subprograms, a full web site, the Web Server).*

Conventions

Symbols in Use

Format Conventions

The following conventions have been applied to the *Format* for each directive in *The ProvideX Web Server Reference:*

Symbol(s)	Meaning
`[ ]`	Square brackets enclose optional elements in the syntax format. For example, with the special HTML code format **`~~argument[?suffix]~~`** you can use either an argument with a suffix (e.g., **`~~Total?F:####0.00~~)`**, or just an argument (e.g., **`~~Total~~)`**.
`{ }`	Curly brackets enclose a list of elements where you must select one item. For example, with **`{yes|no}`**, you must select either **`yes`** or **`no`**.
|	Vertical bars | pipes are used to separate a series of possible values, as in **`{yes|no}`**.
`...`	Dots indicate the continuation of a list of elements.

Punctuation/Syntax Conventions

If you're new to ProvideX, here are some of the punctuation conventions you'll see in the syntax in this manual:

Symbol(s)	Meaning

! *(exclation mark)*	Use the exclamation mark in the ProvideX language to begin a programmer's remark. It's syntactically the same as using a REMark directive. For example,
	`! This is a remark.`
	`REM This is a remark, too.`

`" ... "` *(double quotes)*	Use double quotes to enclose string literals in your ProvideX programs. Double quotes are ***not*** used in URL syntax. For instance, **`?stringvar=abc`** in your URL is the equivalent of **`stringvar$="abc"`** in your ProvideX code.
`#!` *(pound sign + exclamation)*	an ASCII CGI program starts with the pound sign plus exclamation mark. Use the mime type **`application/providex`** for ASCII ProvideX programs. CGI programs (other than ProvideX programs) are not currently supported.
`%` *(percent sign)*	A leading %percent sign in your ProvideX program denotes a global variable, as in the Web Server system variables **`%exit_code`** and **`%print_fn`**. A leading %percent sign is also used in Web Server syntax as a prefix for characters that are invalid in HTML but still needed in your transmission to the browser. (Send them as their HTA value, prefixed by the %sign.)
& *(ampersand)*	Use the ampersand as the separator between arguments in a URL statement. For example: **`http://server/myapp/myprog?stringvar=abc&nextvar&a_var`**
***** *(asterisk)*	When you include the leading asterisk, as in *web, ProvideX will prefix your program name with the full pathname to the ProvideX lib directory automatically and will change the asterisks to underscores. For example, **`*web/merge`** would become something like **`g:\software\pvx\lib\_web\merge`**. You can also use asterisks in the ProvideX language to enclose specialty file names like *MEMORY* and as operators (e.g. in multiplying value*1.05 and in 'C'-type operators, etc.).

Symbol(s)	Meaning
+ *(plus sign)*	Use the plus sign in your URLs to encode spaces on the data side of a variable=data pair or spaces between arguments on the variable side (no equals sign). For example: `http://www.pvx.com/makewish;init?this+that&what=dog+cat+pig` (Plus signs in your ProvideX language syntax have the usual conventional meanings, i.e. as addition operators, etc.)
: *(colon)*	Use the trailing colon to denote the end of a line label in your ProvideX program. An entrypoint in your URL such as `yourprog;entrylabel` refers to the `ENTRYLABEL:` line label in your program where the task handler would start executing your web application.

;
(semi-colon)
	Use the semicolon in an URL to indicate an entrypoint in a program (i.e., a line label where execution should start). For example:
	`call "yourprog;start_here",this_arg,that_arg`
	`http://server/makewish;init?this+that&what=dog+cat`

<...> *(greater and less than)*	Use greater and less than symbols as brackets to enclose special HTML attributes or object tags. As an example, in a ProvideX program: `0020 PRINT (%PRINT_FN)"<html><head>"`

?
(question mark)
	Use a leading question mark to denote the start of an argument list in your URL :
	`http://myapp/mypvxprog?myvar&stringvar=abc&anothervar`
	You also use a question mark to denote the start of a suffix in a Web Server specialty code in HTML pages. For example:
	`~~Total?F:####0.00~~`
	`~~myprog;filltable?T~~`
	In conventional ProvideX syntax, a question mark is the same as a PRINT directive. For example, you can use either:
	`? (%PRINT_FN)"<html><head>"` or
	`PRINT (%PRINT_FN)"<html><head>"`

~~ ... ~~ *(enclosed by double tildes)*	Use double tildes to enclose Web Server specialty codes in HTML template pages. For example: `~~myprog;filltable?T~~`

General Information

General Information

Description

The ProvideX Web Server is HTTP/1.0 compliant. It lets you integrate your HTML and your ProvideX programs in your web applications. The ProvideX Web Server's coding standards for an HTML web page, for running a ProvideX program and passing it data follow standard CGI conventions. You can create HTML using ProvideX and/or using your favourite HTML editor (e.g. Microsoft Word, Frontpage, etc.)

You can use the ProvideX Web Server as your only web server or you can use it to augment your existing web server. For instance, you can use APACHE, Microsoft-IIS or Netscape to serve your main site and use the ProvideX Web Server to run your ProvideX web applications and create web pages to gain access to your data.

Applications

You can use the ProvideX Web Server to:

- integrate your ProvideX programs directly with standard Web Server operations
- gain direct access to ProvideX data files, ODBC data files and other ProvideX file structures for use in your web applications (i.e. your ProvideX programs)
- use almost any ProvideX programming feature in your web application
- give your users access to your web applications without the time delay of launching a CGI application. (The Web Server accepts and runs CGI-compliant URLs as ProvideX programs.)
- perform high volume simultaneous request processing (as many as 1,000 at a time) and automatic restarting of stopped/stalled requests
- reconfigure your web server on-the-fly

You can transfer files of any size and any type with the Web Server. There is no delay before the file transfer begins. You can also force the browser to **`Save As`** the file name you specify for display to the user. Transferring files of any size has no effect on Memory requirements. (No additional RAM is used).

Components of the Web Server

Components

Description:

Our ProvideX Web Server consists of the ***Web Server Configuration*** application and three operating components, described below:

See also **To Edit Configuration Details, p. 12**

The Base Launcher (The Web Server Application)

Our ProvideX Web Server runs using one base launcher application (as a background task). This is the engine that launches your individual web servers (port monitors) and tracks their configuration and activity.

Port Monitors

Port monitors are individual web servers, each responsible for monitoring a unique TCP socket number. You can configure up to 100 port monitors per Web Server base launcher. Your port monitors share the use of your licenced task handlers, but any given port monitor can use as many task handlers as you have licenced and idle/available (up to 1000 simultaneously).

A port monitor's tasks include:

- monitoring a specific TCP socket (port), waiting for HTTP requests from browsers,
- checking the validity and security of incoming requests,
- delegating tasks to task handlers,
- checking for stalled jobs,
- cancelling stalled tasks that have timed out
- returning data from the task handlers to the browser (following the rules outlined in the HTTP Specification 1.0, RFC 1945).

Task Handlers

The task handler is our application that processes requests coming in through port monitors. Task handlers control processing by finding the requested files/pages or running your ProvideX program(s) and returning the resulting data to the port monitor when tasks are completed. We allow you to licence up to a maximum of 1,000 ProvideX Web Server task handlers.

You can have all of your task handlers active simultaneously (as many as you've licenced, potentially working simultaneously for one given port monitor). That is, a given port monitor can use as many task handlers as are necessary and idle/available, up to your licenced maximum. If there are no idle task handlers available when a request is received from a browser, then the Web Server queues the requests until task handlers are available or until the browser disconnects.

NOTE: **The Web Server uses less than 1.5 MB of RAM for each instance under UNIX and approximately 2.5 MB of RAM under Windows. Each task handler runs an instance of the ProvideX interpreter. If your ProvideX Web Server engine is handling 1,000 simultaneous browser requests, it's using concomitant task handlers and memory.**

About Timeouts

Port monitors check, approximately every 30 seconds, for stalled jobs. Task handlers are given at least 90 seconds to complete their tasks and return the data to the port monitor.

You can send long jobs if they respond periodically. However, your ProvideX application must be completed before the port monitor can send any data to the browser since the header must contain information about mime types and the length of the job, etc. The ProvideX Web Server will not currently send any response to the browser until job processing is complete. *Browsers also have a timeout, typically about 60 seconds, for a response from the server.*

If your *Restart Stalled Tasks* setting is ON (see configuring the **PVX Web and Debug Mode, p. 15**), the port monitor terminates stalled task handlers by the timeout and starts a new task handler in its place.

NOTE: **If the port monitor terminates a stalled task, it will inform the browser that the page requested was not available.**

Your Environment

The Web Server takes a snapshot of your ProvideX environment before it begins processing. You can create a START_UP program in the lib/_web directory and set global variables, system parameters, message libraries, PRECISION settings, global variables and globally OPENed files. When the Web Server is about to RUN your ProvideX program, it will use your settings for each task handler it launches. This gives you your own program environment and behaviour but doesn't affect the Web Server or task handlers themselves.

NOTE: **When your application is completed and control returns to the Web Server, the Web Server CLOSEs all the file channels (local and global) that you OPENed in your ProvideX program(s). It does not close any of its own files then, nor does it close any files that you OPENed in your START_UP. This allows you to OPEN common files in your START_UP program so that they are always available to any Providex program your Web Server runs.**

To Set Up the ProvideX Web Server

Setup

Environment

Your operating system must be 32-bit compatible. Then, as a prerequisite to installing the ProvideX Web Server 1.2, make sure that ProvideX 4.12C (minimum version number) is installed.

You can install the ProvideX Web Server in the following environments:

- Windows 95
- Windows 98
- Windows 2000
- Windows NT
- all UNIX systems

Installation Notes

In Windows, use a semi-colon to remark out the **`Library=`** line in your **`\windows\pvx.ini`** or **`winnt\pvx.ini`** file until after the Web Server is installed or activated. The software comes in two forms, one an add-on and the other packaged with ProvideX. After you've installed the Web Server, you can create your icons, as described below.

In UNIX, the Web Server comes as a .taz file (compressed .tar format). The installation has 3 shell scripts:

`runconfig` (runs the Web Server Configuration)
`runserver` (starts the Web Server in background)
`stopserver` (stops the Web Server)

To install it, log in as root, create and change to your pvxweb directory, rename the .taz to your_name.tar.Z and uncompress it. Then, extract the files, activate the Web Server by running **`pvxactv`** using the standard ProvideX activation methods. Use the full pathname for the **`pvxactv`** program (e.g. **`/usr/pvxweb/pvxactv`**).

Example:

```
# mkdir /pvxweb
# cd /pvxweb
# umask 0
# mv xxxxx.taz xxxxx.tar.Z
# uncompress xxxxx.tar.Z
# tar xvf xxxxx.tar
```

For Windows Installations, Create Two Icons

Icon	Icon Set-Up / Format
1	Name: **ProvideX Web Server** Command Line: *your path*`\pvx\pvxwin32.exe webhide.ini *web\webserv` Start In: *your path*`\pvx\lib\_web` Run: `Minimized` **NOTE: Your `webhide.ini` file is designated as READ ONLY.** **Description:** This application is the basic ProvideX Web Server launcher or engine. It's responsible for reading the configuration and launching the individual port monitors.
2	Name: **ProvideX Web Server Configuration** Command Line: *your path*`\pvx\pvxwin32.exe webcfg.ini *web\webcfg` Start In: *your path*`\pvx\lib\_web` Run: `Minimized` **Description:** This application allows you to create or change port monitors and their configurations dynamically.

For Automatic Startup

- In Windows, you can add the ProvideX Web Server to your **Startup Group** so that the your port monitors will start automatically when your system boots.
- In UNIX, you can add an entry to **`/etc/inittab`** to run the base launcher when the UNIX box boots. Use an entry like:
  `pweb:234:once:/usr/pvxweb/runserver` or
  `pvxw:234:respawn:/pvx/pvx \*web/webserv </dev/null >/dev/null 2>&1`

Note that if you use **`once:`** the **`init`** daemon will only spawn the task once and if you execute the **`stopserver`** script, then the **`init`** daemon will not restart the task unless you use **`runserver`**. If you use **`respawn`**, **`stopserver`** will stop the Web Server processes, but the **`init`** daemon will immediately relaunch.

Change Icons

The **`<path-to-your-pvx> \lib\_web\webserv.dll`** contains the icons for the Web Server.

To Configure the Web Server

Overview

Web Server Configuration Icon

For a Web Server installed in a ***Windows environment***, select the Configuration icon to view the *Status/ Configuration* list. This displays the status of your base launcher (e.g. `Stopped/Running`) and any port monitors you have set up.

For a Web Servr installed in a ***UNIX environment***, you must use WindX (installed on your PC) to configure the Web Server. That is, you must gain access to the UNIX box through WindX, then change to the providex directory and run the `runconfig` script.

NOTE: **The *Web Server Configuration* application is a ProvideX *NOMADS* application which runs only in Windows. Some features are restricted to Windows. For instance, Debug mode is only for Windows (not accessible in UNIX through WindX).**

You can create and/or modify your individual port monitor's configuration by clicking on the following buttons in the Status/Configuration window:

Button	Purpose: Status/Configuration
Edit	Jumps to the ***Details Configuration*** dialogue, where you can set up: • Server Basics (i.e. port monitor basics) • Port Style • Task Handlers In addition, buttons in Details Configuration allow you to jump to dialogues for: • Virtual directories • Hit counter lists • *(Templates will be available in a future version.)* • SMTP values (i.e. simple mail protocols and settings) • PVX Web and Debug Mode • Statistics See **To Edit Configuration Details, p. 12** for more information.
Mime	Jumps to the ***Mime Configuration*** dialogue (a listing of available mime types for file extensions such as `jpeg, zip, dll, bin`). You can add, edit or remove mime types. See **Mime Configuration, p. 17**.
Logging	*(Logging will be available in a future version.)*
Templates	*(Templates will be available in a future version.)*

Button	Purpose: Status/Configuration
Admin	Jumps to the ***Administrator Configuration*** listing of time zones. Make sure you select the local time zone that's appropriate for your application. Important: **All web documents use GMT (Greenwich Mean Time). The ProvideX Web Server converts times automatically for you (offsetting your selected local time zone from GMT). The wrong setting here can have adverse effects on your results (e.g. when you send or receive data, cookies, etc., that rely on date-time information such as expiry times).** You can toggle check boxes on this screen to ON to hide the Web Server and your port monitors (so that they don't appear on your task bar) and to tell ProvideX to shut down all port monitors when the Base Launcher is **`Stopped`**.

Status Control

The Web Server refreshes the *Status/Configuration* list every 5 seconds, returning the current status of your base launcher and your configured port monitors. The base launcher is suspended from overseeing the port monitors while the Configuration application is running. (The Configuration application takes responsibility for dynamic changes to the various port monitors.)

You can use the ***Start/Stop*** toggle button at any time for any port monitor or the base launcher.

NOTE **You don't have to shut down or restart a port monitor when you change its configuration. The Web Server will dynamically update itself.**

Menu Bar: About WebServer

The Status/Configuration menu bar offers you many of the above selections, as well as a ***Help>About*** option, which jumps to the ***About WebServer*** screen. This screen displays your WebServer version and O/S information and also provides active links so that you can jump to our ***Support*** site and/or to the ***ProvideX Web Site***.

To Edit Configuration Details

Description:

When you click on the ***Edit*** Button in the Web Server Status/Configuration display, the ***Details Configuration*** dialogue box appears. The prompts and setup information are listed in the charts in the following sections for the three main areas of this dialogue box:

- Server Basics (i.e. port monitor basics)
- Port Style
- Task Handlers

You can create up to 100 port monitors (each with a different "root" directory, if desired). Each port monitor must oversee a different, unique TCP/IP port (socket number).

Server Basics Configuration

Prompt	Server Basics Web Server Details Configuration
Server	Your server name can be up to 14 characters long and is case-sensitive. The server name is solely for your reference. Example: `Test Server`
Description	We recommend using your site name, e.g. `www.mydomain.com`. This description is used for Directory Browsing if the browser does not send a host_name.
TCP Socket	This is the TCP/IP socket number you want this port monitor to oversee. Don't use a duplicate of one that's in use by other applications. Web servers typically default to port 80. You can override the default with any valid TCP socket number. For example, you can use `4000` as your port number to resolve a potential conflict with Apache, which defaults to port 80. (Valid range: 1 to 65535. Ports below 2000 are normally reserved for standard Internet applications like FTP, mail services.)
Default Root Dir	The Web Server will treat this as the *"root" directory* for a given port monitor and will restrict clients' access to files in this "root" directory and its subdirectories. For example: `\pvx410\direxions\web\webroot`. *If you make use of Virtual Directories, then the Default Root Directory will be used if either the browser does not send the HOST NAME typed in by the user or if you don't have a directory set up for the specific HOST NAME in your Virtual Directory Mapping. See* **Details Configuration Buttons, p. 14**.
Default Doc	You can declare one or more d*efault document* names, e.g., `home.htm`. ***When the client requests a Directory:*** *If you have declared default documents,* then the Web Server responds to a browser's Directory request with the first of your listed default documents that exists in the directory. The Web Server scans the directory for default documents in the order in which they appear on your list. *If you don't declare default document names (or none of the documents exist)* then the Web Server responds to a browser's Directory request in one of two possible ways: • *If Directory Browsing is ON* the Web Server returns a directory listing • *If Directory Browsing is OFF* the Web Server returns an `Error 403 Forbidden`. (See **Port Style Configuration, p. 13**.)

Prompt	Server Basics Web Server Details Configuration
Hit Counter	Hit counters track how many times a specific URL on your site has been sent to a browser. Radio buttons give you three *Hit Counter* options: • **`Off`***:* No hit count is tracked. • **`Specified List`***:* Web Server only tracks the web pages or URLs you list in the *Hit Counter* dialogue. *(Click on the Hit Counter button to jump to the dialogue and create, modify or reset your list.)* • **`Automatic`**: Web Server tracks all pages hit (whether or not they are in the hit list already). The Web Server returns the current hit count to you in the global system variable **%hit_counter, p. 30**. The count is based on the URL of your ProvideX application.

Port Style Configuration

Prompt	Port Style Web Server Details Configuration
Deny Start	*ON:* The port monitor cannot be started. If the port monitor is already running when you toggle this ON, it will continue to run until you stop it. Then it can't be re-started. *OFF*: The base launcher will automatically start the port monitor or you can manually start it.
Server Enabled	*OFF*: Your port monitor returns a **`Server Busy`** message to any browser making a request. *(The port monitor will still accept administration commands.)* *ON*: The port monitor is available to serve incoming requests from clients.
Secure [HTTPS]	*Available in a future version.*
Directory Browsing	*ON*: When the user requests a directory listing, the Web Server returns a directory listing page. **Exception: If you assigned a value in Default Doc, p. 12 (in Server Basics configuration, above) and that page exists in the directory requested, then the default document is returned instead of the directory listing.** *OFF*: If you toggle this OFF and you haven't assigned a Default Document (or the default document doesn't exist) the Web Server returns an **`Error 403 Forbidden`** to the client.

Task Handlers Configuration

Prompt	Task Handlers Web Server Details Configuration
Max Idle Time (mins)	In minutes. If your port monitor is idle for the given number of minutes, the Web Server shuts down extra task handlers to the minimum number you set, below). Use value=0 (zero minutes) to prevent the shutdown of your extra task handlers.
Min Task Handlers	This is the minimum number of task handlers you want your given port monitor to launch automatically, e.g. `0` (zero is a valid value for minimum).
Max Task Handlers	This is the maximum number of task handlers you want your given port monitor to launch, e.g. `5`. The valid range is 1 to your total licenced number of task handlers (maximum 1,000). The Web Server can process many simultaneous incoming requests by passing the requests on to available task handlers for processing. **Reminder: Each task handler requires an instance of Providex. Watch your memory requirements.**

Details Configuration Buttons

Button	**Purpose: Details Configuration**
Virtual Dirs	Jumps to ***Virtual Directories*** mapping, where you can update, delete or display the virtual directory names you're using as your logical host names. You can map incoming requests by their declared host name to different directories on your hard drive. For example, with a port monitor listening to TCP port 80: Default Root Dir: `c:\inetpub\pvxweb` Host Name: `www.pvx.com` and Root Directory: `c:\inetpub\pvxweb` Host Name: `www.edias.com` and Root Directory: `c:\inetpub\edias` If a request doesn't include a host name, the Web Server will use the default root directory. Also, Web Server will change to the correct root directory before RUNning a ProvideX program.
Hit Counters	Jumps to the ***Hit Counter Configuration*** dialogue, which lists the number of hits and the date/time of the last hit for a given web page or URL. You can add, remove and reset items in the hit counter list.
Templates	*(Templates will be available in a future version.)*
SMTP	Jumps to ***SMTP Configuration*** where you can maintain the settings you want to use with ProvideX e-mail tools. If you assign a value for ***SMTP Server*** name, it's available to your applications in a variable named **%SMTP_gateway$, p. 31**. If you assign an ***SMTP Port*** number, it's available to your applications in a variable named **%SMTP_socket, p. 31**.
Debug	Jumps to ***PVX Web and Debug Mode*** configuration. (See **Troubleshooting, p. 15**).
Statistics	Jumps to ***WebServer Statistics*** which displays port monitor configuration details and statistics, including: • Server name, socket number and state (e.g. `Stopped`) • Status (e.g. `Old Statistics from Thu, 14 Oct 1999 12:24:54`) • Min and Max task handler settings, how many used and currently running • Number of requests served, maximum number and time in the queue.
Remove	Use this button to ***Remove*** a server (port monitor) from your Web Server configuration.

Troubleshooting

Debug Mode

Description:

Configure your troubleshooting settings in the ***PVX Web and Debug Mode*** screen, reached via the *Details Configuration* dialogue. You can use the *Restart Stalled Tasks* setting in conjunction with debug mode and debug windows to test and debug your code.

NOTE: **Debug mode and the debug windows are not available in UNIX.**

ALSO: **The Web Server automatically sets the `'NE'` system parameter to OFF when each of your jobs is completed. With 'NE' set to OFF, the Web Server can return values in such system variables and functions as `ERR, RET, and MSG(-1)` to report untrapped errors in your code.**

Error Conditions

There are two types of error conditions in dealing with the Web Server:

Type	Description
Hard Errors	Unrecoverable conditions. These are reported to the console. The specific port monitor exits. The base launcher automatically restarts any failed port monitor within 30 seconds of its going down.
Soft Errors	Recoverable conditions. These errors are reported to the browser sending the request that generated the error.

NOTE: **When you receive a message that the Web Server is unable to gain access to a TCP port, that usually indicates that another application has the TCP port open or the O/S has the socket in a WAIT state, pending the clearing of unclaimed data in the socket.**

For more information on error handling, see the PVX Web and Debug Mode chart, next. See also **%exit_code, p. 30** and **Base64 Error Messages, p. 35**.

PVX Web and Debug Mode

Prompt	PVX Web and Debug Mode Web Server Details Configuration
Restart Stalled Tasks	*ON:* The port monitor checks for stalled task handlers approximately every 30 seconds and • terminates a task handler that has been running longer than 90 seconds and is stalled due to an error or ESCAPE directive and • starts a new task handler in its place. Long jobs are acceptable if they respond periodically. *OFF:* If you include a `PRINT 'SHOW'(1),;ESCAPE` directive in your ProvideX program, control will transfer to the task handler and processing will stop at your ESCAPE. This gives you an opportunity to check your variables and debug the program if necessary. *This setting should be OFF when you're debugging a ProvideX program. (Then, task handlers are not monitored or restarted, so the Web Server won't kill your task in the middle of debugging.) If your Web Server is running on an NT server, jobs will appear on the console.*

Prompt	PVX Web and Debug Mode Web Server Details Configuration
Debug Mode	ON: The Web Server displays any ProvideX debug windows you've selected just before your ProvideX application runs. The task handler will automatically stop on the first line of your code so that you can debug the program if necessary. (No ESCAPE is necessary in your code.) After your task handler exits, only the Trace and Watch windows remain open. Any other debug windows you selected are closed automatically. **NOTE: Debug mode is not available when you run the Web Server in a UNIX environment (not even if you use WindX).**
Trace Window Watch Values BreakPoints Command Mode	**NOTE: These settings are only available when Debug Mode is set ON. They are not available in UNIX (not even if you use WindX).** When you turn ON your setting(s) for *Trace Window, Watch Values, BreakPoints or Command Mode*, the Web Server will turn on the given debugging window(s) automatically for debug mode.

Mime Configuration

Dialogue **To Set Up Mime Types**

Description:

When you click on the ***Mime*** Button in the Web Server Status/Configuration display, the ***Mime Configuration*** scroll box appears. You can add, edit or remove mime types. Note that mime_type values are always in lowercase. A few of the more common types and defaults are listed below. All types in the **text/** classification automatically have their end of line characters converted to $0D0A$ (carriage return, line feed).

Mime Type	
`text/plain`	is the *default* for data files and web pages: **`text`** is the general classification and **`plain`** is the specific mime type. The mime type for README and .INI files is **`text/plain`**.
`text/html`	is the *default* for ProvideX programs: use **`text`** as the general classification and **`html`** as the specific mime type. Exception: see **`providex`**, below.
`providex`	Use the mime type **`application/providex`** for ASCII ProvideX programs.
`octet-stream`	The mime type for plain binary is "**`application/octet-stream`**".
`pdf`	The mime type for general binary transfer is **`"application/pdf"`**. The web browser launches the PDF viewer to handle the file once it is received. If there is no viewer, then the file goes through the "Save As" dialogue in the browser.

You'll find more information on other mime types in Windows Explorer under *View>Folder Options>File Types Tab*.

Serving Simple Mime Type Requests

The ProvideX Web Server opens the requested file in binary mode and tries to make a determination of the file contents before it attempts to match the file extension to a mime type. First it checks to see if the file is:

- a directory,
- a ProvideX program in tokenized form
- an ASCII CGI program, starting with the #! symbols (not supported by the Web Server at present).

If the file fits none of the above categories, then the Web Server matches the file extension to the mime type to determine file type and control the transmission.

NOTE: **If your browser fails to display or download data from the Web Server, check to make sure that the file's corresponding mime type is properly set up. For example, a browser can't receive and display a `jpeg` image unless the file includes the jpeg file extension which is mapped to the mime type "`image/jpeg`" in your *Mime Configuration* dialogue list.**

Client/Browser Requests

Valid Requests and Methods

Description:

The ProvideX Web Server addresses two forms of browser requests:

- requests to return file contents (e.g. web pages, data) or
- requests to RUN your ProvideX programs to process data.

Requests contain a header block terminated by a blank line and can also (optionally) contain a data block known as the body. Each line of information sent to or from a web server or browser ends in a CRLF terminator (i.e. carriage return plus line feed: $0D0A$).

See also **URL Request to GET a File, p. 19**

URL Request to Run a ProvideX Program, p. 22

***web/merge To Merge Dynamic Data with HTML Templates, p. 39**

DireXions Examples (Appendix), p. 47

Request Methods

There are three methods or types of requests:

Method	Description
GET	is the most common method. It's typically used for files and small CGI requests. It contains the name of the file being requested and (optionally) can include command line arguments or a short variable list. There is no body information, just a header. The size limit is typically about 100 characters.
POST	is typically used for data from forms or for large CGI requests. POSTS can contain body information in addition to the header block. Parameters (command line arguments, variables) are relegated to the body of the request.
HEAD	is typically used by a browser or URL validator to check the validity of a request without receiving data. HEAD requests are handled like GET requests, except that only the HTTP Header will be returned. No HTTP Body is actually sent.

The main difference between the GET and POST methods is in the amount of information you want to pass in the request. Use the GET method for simple URL requests. There is a limit of about 100 characters when you're passing parameters (arguments, variables, etc.) in your URL request.

If you want to pass large amounts of information, use the POST method instead. Use the POST method, in your HTML pages. For example, in **Orders Web Page 1: login.htm, p. 52**:

```
<td><form method="POST" action="/orders/webord;logon">
```

Examples

The first example illustrates the GET method in a file request from MS Internet Explorer 5 (IE5):

```
GET /mypage.htm HTTP/1.1
Accept:   application/vnd.ms-excel,   application/msword,   applciation/vnd.ms-
powerpoint, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0b2; Windows 98)
Host: 127.0.0.1
Connection: Keep-Alive
```

This next example is a URL request to GET a file: `http://www.pvx.com:3000/support/home.htm`

URL Request to GET a File

Request to GET a File

Format

URL Request ***protocol://server[:socket]/filename.ext[?argument[&argument...]]***

Where:	
protocol	is the applicable network protocol, for example: `http://` or `ftp://`
server	is the server name. Use an IP (internet protocol) address like `127.0.0.1` or a DNS (domain name) like `www.pvx.com`

socket	is an optional TCP socket or port number (an integer preceded by a colon). Valid range: 1 to 65535.
	By default, all requests from a client are sent to TCP socket 80, unless you override the port number in your URL statement. To do this in any HTTP-compliant browser, include a colon and the port number right after the server name. For instance, the first URL below would default to TCP socket 80. The second would connect with TCP socket 3000.
	`http://www.pvx.com/support/home.htm`
	`http://www.pvx.com:3000/support/home.htm`

filename.ext	is the item the browser is requesting. This could be the name of: • an HTML page, e.g., `myfile.htm` • a directory (include a trailing forward slash) • a ProvideX program • or any file or any type (including a directory listing) that appears in the Web Server's Root Directory or subdirectories of the Root Directory. See also **URL Request to Run a ProvideX Program, p. 22**. **DireXions Examples (Appendix), p. 47**
argument(s)	optional argument(s). If you include arguments: • Mark the start of a list of arguments with a leading ?question mark. • You can use numeric variables, but they can't have an equals sign or a + sign (i.e. encoded space). By default, the Web Server sets the value of a numeric (Boolean) variable to 1. • You can use string variables in variable=data pairs, e.g. `data=abc+123`. (The plus sign represents an encoded space in the string. That is, abc+123 is the equivalent of "abc 123". There are no quotation marks enclosing the string value in the URL.) • You can include command line arguments. • If you include more than one argument, use an ampersand as a separator, e.g. `?myvar1&data=abc+123&nextvariable` **NOTE**: **The Web Server saves *command line* arguments in %ARGS$[1:%ARGS] where:** **• %ARGS$ stores the arguments (like the ARG( ) function return value) and** **• %ARGS stores the number of the arguments (like the NAR system variable).**

Description:

The Web Server uses the information in your URL statements to locate the file or information being requested. Each object on a page is a separate request from a browser to a web server. The ProvideX

Web Server can handle virtually any type of file and serve it back to the client. (Exception: The Web Server does not currently allow for CGI scripts other than ProvideX programs.)

NOTE: **Characters allowed in a URL are uppercase A-Z, lowercase a-z, integers 0-9, and $-_.+!*'( ), (i.e. the dollar sign, minus sign, underscore, period, plus sign, exclamation mark, asterisk, single quotation mark, standard brackets, and comma). Anything else must be encoded as a leading %percent sign plus the ASCII representation of the character's HEX value (i.e., its HTA( ) value).**

Requests: ***Examples***

The following example is a request for the file "**`myfile.htm`**" from the Root Directory of the web server **`www.pvx.com`**. Since no optional TCP socket is specified, the request would be made to the **`www.pvx.com`** web server on TCP socket 80 (the default).

The example: **`http://www.pvx.com/myfile.htm?numvar1&stringvar=abc%20def`** would pass two variables to the file:

> **`numvar1`** would become the numeric variable **`numvar1`** with a value of **`1`**
> **`stringvar`** would become the string variable **`STRINGVAR$`** with a value of "**`abc def`**" (%20 becomes the space).

This example **`http://127.0.0.1/somprog?one+two+three`** is a request for the file "**`someprog`**" from the web server currently running on the local machine and passes it three arguments, which the Web Server returns in ***%args$[%args]*** as **`%ARGS$[1]="one"`**, **`%ARGS$[2]="two"`** and **`%ARGS$[3]="three"`**. The value in **`%ARGS=3`**. See ProvideX Web Server Variables, p. 28.

The two examples below are requests for the program "**`someprog`**" from the sub-directory "**`somedir`**" in the web servers Root Directory. The first example creates a numeric variable, ONE=1. The second passes the values in **`%ARGS$[1]="one"`** and **`%ARGS=1`**.

```
http://127.0.0.1/somedir/someprog?one
```

```
http://127.0.0.1/somedir/someprog?+one
```

This example requests the file named "**`myprog`**" which, if it's a Providex program, will start execution at the label **`LOOP:`**

```
http://127.0.0.1/myprog;loop?comp=001&name=Your+Name+Here&copies=3&copy&now+then
```

It is passing the following from the URL for use in the ProvideX program:

> **`comp=001`** as **`COMP$="001"`**
> **`name=Your+Name+Here`** as **`NAME$="Your Name Here"`**
> **`copies=3`** as **`COPIES$="3"`**
> **`copy`** as **`COPY=1`**
> **`now+then`** as **`%ARGS$[1]="now"`**, **`%ARGS$[2]="then"`**, **`%ARGS=2`**

This example is a request to the web server listening on port 2000 to RUN the file "**`someprog`**", if it's a ProvideX program:

```
http://127.0.0.1:2000/someprog?name=you&name=me&name=them
```

Note that the arguments all have the same variable name. This is handled as a special case. When a variable name is passed more than once, the Web Server creates a single variable (in this instance, **`NAME$`**) and concatenates the data from each consecutive pass to the previous value in the variable as +SEP+extra data (in this instance, **`NAME$="you"+SEP+"me"+SEP+"them"`**.

Requests:	***Examples***

The Web Server builds a special COMPOSITE STRING named **`CGI$`**. When you pass variables (either string or Boolean numeric, but not arguments) the Web Server builds an IOLIST of your variable names and then DIMensions CGI$ to that IOLIST.   Variable names passed more than once only appear in the IOLIST once.

```
http://127.0.0.1/myprog;loop?comp=001&name=Your+Name+Here&copies=3&copy&now+then
```

For the example above, a **`PRINT LST(IOL(CGI$))`** directive returns

```
IOLIST COMP$,NAME$,COPIES$,COPY
```

You can assign the current data values to the DIMensioned composite string **`CGI$`** from the variables passed in the URL by using **`CGI$=REC(IOL(CGI$))`** when RUNning your program. This allows you to gain access to the data as composite string elements, in this instance, with the following values:

```
      CGI.COMP$="001"
      CGI.NAME$="Your Name Here"
      CGI.COPIES$="3"
      CGI.COPY=1
```

URL Request to Run a ProvideX Program

Request for a ProvideX Program

Format:

URL to Run Program	***protocol://server[:socket]/program[;entry][?argument[&argument...]]***

Where:	
protocol	is the applicable network protocol, for example: `http://` or `ftp://`
server	is the server name. Use an IP (internet protocol) address like `127.0.0.1` or a DNS (domain name) like `www.pvx.com`

socket	is an *optional* TCP socket or port number (an integer preceded by a colon). Valid range: 1 to 65535. Values below 2000 are normally reserved for standard Internet activities.
	By default, all requests from a client are sent to TCP socket 80, unless you override the port number in your URL statement. To do this in any HTTP-compliant browser, include a colon and the port number right after the server name. For instance, the first URL below would default to TCP socket 80. The second would connect with TCP socket 3000.
	`http://www.pvx.com/support/home.html`
	`http://www.pvx.com:3000/support/home.html`

program	is the name of the ProvideX program the browser is asking the Web Server to RUN. Note that **Your Web Application Must "Stand-Alone", p. 23**.
entry	is the optional line label where the ProvideX program starts. See also **Entry Points, p. 23**. If you omit this, ProvideX starts running your application at the first executable line of code in your program.
argument(s)	optional argument(s). Mark the start of your argument list with a leading ?question mark and when you have more than one argument, use an ampersand (&) as the separator between arguments. For example: `?myvar1&data=abc+123&nextvariable` See also **Variables as Arguments, p. 23** and **Command Line Arguments, p. 24**.

Description:

Here's how the Web Server works when it receives a URL request to run a program:

- The web server (port monitor) receives a browser's request to RUN your ProvideX program file and checks for validity. (Does the file exist? Is it a program file in tokenized code? ...). If the request is valid, it assigns the task to a task handler.
- The task handler parses variables and arguments from the request, loads the variables with available data, suspends itself and RUNs your ProvideX program, starting at the entrypoint if you specify one in the URL.
- Your program creates the output and directs it to the %print_fn channel. See also **%print_fn, p. 31**. Then your program exits. (*Note: If your program is creating output other than the default "text/html" mime type, make sure you assign the matching mime type to the Web Server's %content_type$ variable. See also* **%content_type$, p. 29** *and* **Mime Configuration, p. 17**.*)*
- The task handler regains control, processes your incoming data according to its mime type and builds the blocks of outgoing data. Then it builds the outgoing header information for your data and returns the data to the port monitor.
- The port monitor returns the data to the browser when the job is completed.

Your Web Application Must "Stand-Alone"

Incoming requests are processed by the next available task handler. All requests are independent of each other and do not interrelate or share information with each other. Each web application must stand alone in the here and now.

Note that task handlers ***will***:

- automatically CLOSE any files left open when your task is completed except for any files that were open when it launched, and
- automatically clear all LOCAL variables (leaving their values unavailable to the next task handler).

Task handlers ***will not*** automatically clear ***your*** GLOBAL variables and the global variables are not shared between task handlers (so you must initialize your global variables in your applications).

You can use a web concept called a "cookie" to share information from one request to another. See ***web/cookie To Set, Send and Retrieve Cookies, p. 36**.

You can use URLs (Uniform Resource Location requests) to pass arguments and information to your given applications. See the examples and explanations below.

NOTE: **The Web Server accepts standard CGI arguments as valid for ProvideX programs, but does not currently allow for other types of CGI scripts. ProvideX programs will run faster if in tokenized format. However, you can use ASCII text ProvideX programs if they have a file extension and you set the mime type for that extension to `application/providex`.**

The example here is a simple GET request to run a ProvideX program. *(The POST method would relegate variables and data to the body of the request. See* **Client/Browser Requests, p. 18** *for more information on the differences between GET and POST requests.)*

```
http://server/myapp/myprog;start_here?my_str_var=abc&my_B_var&next_var=123
```

Reminder: **Characters allowed in a URL are uppercase A-Z, lowercase a-z, integers 0-9, and $-_.+!*'( ), (i.e. the dollar sign, minus sign, underscore, period, plus sign, exclamation mark, asterisk, single quotation mark, standard brackets, and comma). Anything else must be encoded as a leading %percent sign plus the ASCII representation of the character's HEX value (i.e. its HTA( ) value). For example, a space character is represented as 20 in hex and you would use %20 in your URL.**

Entry Points

The ProvideX Programming Language allows you to use line-label entrypoints (e.g. in CALL directives). You can also include entrypoints in Web Server URLs to redirect processing in your web applications. In the example above, a ProvideX Web Server task handler would run the ProvideX program **`myprog`** starting execution at the program line labelled **`start_here:`** instead of at the first executable line of the program.

Variables as Arguments

If you include variables as arguments in your URL, ProvideX treats them as real variables in ProvideX itself. The task handler parses the URL and creates any variables it finds in your argument list. It pre-loads them (either with any data sent along with them or with Boolean values) before it runs your program. You can use:

- String variables in variable=data pairs. In the example above, the task handler would create a variable **`my_str_var$`** with a value=**`"abc"`**. (In the URL itself, there is no trailing $ dollar sign in the variable name, and there are no quotation marks enclosing the string value.) You can use a plus sign + in your URL to encode spaces in strings, as in **`data=abc+123`**, in this case, the equivalent of the ProvideX code **`data$="abc 123"`**.

- Numeric (Boolean) variables do not have an assignment with an equals sign nor can they have +plus signs in their names (no encoded spaces). The task handler sets the value of a numeric variable to 1. (In the example above, **`my_B_var`** would have an assigned value**`=1`**.)

You can use your URL arguments to load values into your application to set conditions controlling processing.

NOTE: **During a POST operation, the browser only sends the web server variables which are greater than NULL and <>0 (zero).**

ProvideX Web Server Variables:

In addition to your own variables, you can make use of the ProvideX Web Server's own variables in your programs. See the list of reserved **ProvideX Web Server Variables, p. 28** for details.

Command Line Arguments

Your URL can also include command line arguments. Note **`this`** and **`+that`** in the example below. ProvideX evaluates everything between the argument separators (?,&) as a unit. If a unit has no equals sign but includes a plus sign, the Web Server interprets the unit as command line arguments.

NOTE: **The task handler adds any command line arguments it finds to the %ARGS$[ ] array, containing from 1 argument to the number of arguments passed.**

Example:

```
http://server/makewish;init?this+that&what=ProvideX+Web+Server&mine
```

The task handler would parse this URL and load the arguments as:

- two command line arguments (**`this`** and **`that`**) in %ARGS$[1] and %ARGS$[2]
- one string variable with decoded spaces (**`what$="ProvideX Web Server"`**) and
- one Boolean variable (**`mine=1`**).

About Your ProvideX Programs

Programs: Notes and Examples

Description:

Use the information in this section and the examples throughout the manual as an aid when you create your web applications.

See also: DireXions Examples (Appendix), p. 47

Output from Your ProvideX Program

As output, you can return any data that is valid to the browser. This includes:

- HTML pages generated by your program code
- binary data which is to become a file on the end-user's system
- JAVA script (which becomes part of the page to be evaluated by the browser) and
- raw data. (Set the flag %raw_text, p. 31 in your program. Then the Web Server wraps the required HTML around your data.)

Your ProvideX programs can send files to a browser if you set the **`%content_type="application/octet-stream"`** and **`%content_filename$="somefile.ext"`** and then dump the file into the **`%print_fn`** channel (maximum 4096 bytes per WRITE RECORD). See also: ProvideX Web Server Variables, p. 28.

No Direct Interaction with User

Your ProvideX web application can do virtually anything except interact directly with the user interface. This means you can't use

- INPUT or OBTAIN directives
- the built-in graphics control objects such as BUTTONs, LIST_BOXes
- the print mnemonics, such as 'FRAME', 'TEXT', 'CS' etc.

Interaction with the user must be done ***indirectly*** through HTML forms or pages that can request information and return resulting responses and data to your program. Your program can construct an HTML page on the fly from the data. See entry points 9 to 13 in *web/util CALL EntryPoints, p. 45 and see *web/merge To Merge Dynamic Data with HTML Templates, p. 39.

Standard Output Channel

Think of coding for browsers as *block mode* transmission to a background task. The Web Server sets up a memory-resident logical file using the %print_fn, p. 31. This variable contains the logical file number or channel the Web Server is using for standard output. You dump your data into the memory file by using PRINT, WRITE and other output directives with "(%print_fn)" as the file channel as in **`PRINT (%PRINT_FN)DATA`**. Then the Web Server transmits the contents for you.

Limits on Output:

There are a few limits on the amount and type of output you can generate:

- Although the memory file has a block size of 6K, your individual WRITE, WRITE RECORD or PRINT output to the %print_fn channel should not exceed 4K, because the ProvideX Web Server needs part of the memory file for other purposes (e.g. to add end-of-line translations and characters which are needed by the browser but illegal in HTML code).
- Although you can generate more than one style of output in a single program (see Example: Basic HTML Output, p. 26 with raw text in a single program), each mime style must be generated through a separate request. You can't mix different styles in a single output to the

memory file.

- Remember that your application has a 90-second time-to-live. If the task handler doesn't complete your given ProvideX application before the timeout, the port monitor will kill your job and start a new task handler in its place. See also About Timeouts, p. 6.

Example: Basic Raw Text Output

The URLs and ProvideX program below were created for the ProvideX ***DireXions*** Technical Conference in March, 1999 to illustrate simple raw text requests. Three URL requests are set up in the browser's address field:

1. to run the program,
2. to dump the variables and
3. to list the program.

```
http://127.0.0.1:4000/web_text
http://127.0.0.1:4000/web_text;dump
http://127.0.0.1:4000/web_text;list
```

The simplest method of generating a report is by using our `%raw_text` flag, as in the ProvideX program below. Notice the WEB_TEXT entry point line labels which were requested in the URLs above. The task handler starts executing each of the three requests at its respective executable line or label entrypoint (`0010, DUMP:` and `LIST:`) below.

```
0010 ! WEB_TEXT (Report Style Output in Raw Text):
0020 LET %RAW_TEXT=1,%RAW_TITLE$="Example: Raw Text Output"
0030 LET C=HFN; OPEN (C,IOL=*)"CSTFILE"
0040 PRINT (%PRINT_FN)@(1),"CustID",@(20),"Customer Name",'LF'
0050 LOOP:
0060 READ (C,END=DONE)
0070 PRINT (%PRINT_FN)@(1),CST_ID$,@(20),CST_NAME$
0080 GOTO LOOP
0090 DONE:
0100 CLOSE (C)
0110 END
0120 !
0130 DUMP:
0140 LET %RAW_TEXT=1,%RAW_TITLE$="VARIABLE DUMP FOR WEB_TEXT"
0150 DUMP (%PRINT_FN)
0160 END
0170 !
0180 LIST:
0190 LET %RAW_TEXT=1,%RAW_TITLE$="LISTING OF WEB_TEXT PROGRAM"
0200 LIST (%PRINT_FN)
0210 END
```

Example: Basic HTML Output

In the next example, also created for the ProvideX ***DireXions*** Technical Conference, two URL requests are set up in the browser's address field:

```
http://127.0.0.1:4000/web_html
http://127.0.0.1:4000/web_html;list_and_dump
```

This is a simple example of a ProvideX application which contains code to generate HTML and raw text to illustrate that since each entrypoint routine (i.e. line `0020` and `LIST_AND_DUMP:`) stands alone as a separate request, each can have its own styling.

```
0020 PRINT (%PRINT_FN)"<html><head>"
0030 PRINT (%PRINT_FN)"<title>"+"This is a simple HTML example"+"<title>"
0040 PRINT (%PRINT_FN)"</head><body>"
```

```
0050 PRINT (%PRINT_FN)"<h1>"+"This is a simple HTML example"+"<h1><hr><p>"
0060 PRINT (%PRINT_FN)"Welcome to ProvideX Web Server</p>"
0070 PRINT (%PRINT_FN)"<p>This is an HTML output example</p>"
0080 PRINT (%PRINT_FN)"<p>You can use any HTML code as output as long as it "
0090 PRINT (%PRINT_FN)"adheres to standard HTML specifications."
0100 PRINT (%PRINT_FN)"</p><body></html>"
0110 END
0120 !
0130 LIST_AND_DUMP:
0140 LET %RAW_TEXT=1,%RAW_TITLE$="Listing + Variable Dump in Raw Text"
0150 LIST (%PRINT_FN)
0160 PRINT (%PRINT_FN)'FF'
0170 DUMP (%PRINT_FN)
0180 END
```

Example: Java Script

This example illustrates Java Script in a ProvideX web application. The example is not complete.

```
0010 PRINT (%PRINT_FN)"<script Language=JavaScript>"
0020 PRINT (%PRINT_FN)"IF(theForm.AdInfo.value==""){
0030 PRINT (%PRINT_FN)"alert("Please enter a value for the \"Ad Information\"
0030:field.");"
0040 PRINT (%PRINT_FN)"theForm.AdInfo.focus();"
0050 PRINT (%PRINT_FN)"return (false);"
0060 PRINT (%PRINT_FN)"}"
0070 PRINT (%PRINT_FN)"</script>"
```

ProvideX Web Server Variables

Webserver Variables

The ProvideX Web Server gives you access in your programs to some of its own variables. These variables are created or filled by the Web Server and/or with information received from the browser. A leading %percent sign indicates that the variable is global.

Web Server Variable	Value(s):
Warning	***The following variables are reserved for exclusive use by the ProvideX Web Server for special file channel values. Please do NOT use these file channels:*** ***%mime*** ***%tmpl*** ***%hits*** ***%serv*** ***%TCP_FN*** ***You can use %body_buff in your applications, but for READing only. See %body_buff, below.***
%allow$	Allowed methods recognized by the browser.
%args$[%args]	The **`%args$[ ]`** array contains the command line arguments passed to the task handler by the URL request, with **`%args`** (number of arguments passed) ranging from 1 to the total number of arguments passed.
%authorization$	USERID/PASSWORD screen if passed from the browser. Typically encoded in Base64.

%body_buff	The Web Server uses this file channel for an incoming request's body. It converts the first 32000 bytes into variables and ignores the rest, but leaves all of the body in %body_buff so that it's available if you need it. This is a *memory* file with a maximum record size of 8192 bytes. Data is blocked into more than one data record (beginning IND=0 to IND=n).
	
	You can use %body_buff in your applications, but it should only be READ from. The file whose channel is in the numeric variable %body_buff contains any body information sent in a POST request. If your POST sends more information than will fit in a single ProvideX STRING variable, then you must retrieve it manually from the %BODY_BUFF channel.
	
	NOTE: %CONTENT_TYPE$ will tell you the mime type of the body content. %CONTENT_LENGTH will tell you the body's total length (number of bytes).
	
	Example:
	`READ RECORD(%BODY_BUFF,IND=0)A$ !Reads first block`
	`READ RECORD(%BODY_BUFF,END=DONE) !Reads remaining blocks`

%cgi$	Contains a compiled IOLIST of all the variable names passed.
%content_encoding$	Data encoding if the browser pre-encodes the data.

Web Server Variable	Value(s):
%content_filename$	If you set this to a filename (no path), the Web Server adds a Content-Disposition header to the outgoing data stream. This notifies the browser to `Save As` the filename you supply. TIP: **You can use this to send a ProvideX program to a browser without running it. (A typical link to a browser page would RUN a ProvideX program if clicked on.)**
%content_length	Length of the returned content. This information is available to your program if given. You do not need to set this when your ProvideX application finishes. The task handler will calculate it, based on the data output and any automatic translation done to send your data back.
%content_type$	Content or mime type. If the browser sends a content type, the value is available to your ProvideX program. If your ProvideX application does not assign a %content_type$, the Web Server defaults to `text/html`. If your ProvideX application overrides the default %content_type$, then the Web Server will pass your value for the content type back to the browser. If the `%content_type$` begins with `"text/"` then the Web Server automatically performs any required end-of-line conversions.
%cookie.rcv	The number of cookies received from the browser
%cookie.snd	The number of cookies you SET and send to the browser. The cookies are stored in strings of %cookie.n$ where n is the cookie number.
%document_label$	The line label entrypoint for your program, if given in the URL (e.g. `myprog;entry`).
%document_root$	The root directory for the request.
%document_uri$	The filename portion after the protocol and server in the URL request. Note that uri is an acronym for Uniform Resource Indicator.

Web Server Variable	Value(s):
%exit_code	Error status on exit. The task handler will set %exit_code to an appropriate value. The value is typically 200 ("OK") unless there is a problem (such as in situations where the task prematurely disconnects, or there is no data output). Valid codes include: 200 OK 201 Created 202 Accepted 204 No Content 300 Multiple Choices 301 Moved Permanently 302 Moved Temporarily 304 Not Modified 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found 500 Internal Server Error 501 Not Implemented 502 Bad Gateway 503 Service Unavailable The Web Server returns an **`Error 403 Forbidden`** on browser requests for ProvideX data files (keyed, indexed ...). You can assign a value to %exit_code if you want to force the task handler to return a specific error code. For example, if you set **`%exit_code=401`**, the task handler will return an Error 401 to the browser.
%from$	Source if the browser sends a From.
%hit_counter	The current number of hits for your page (if any).
html_file$	The name of the HTML page you want *web/merge to evaluate and dynamically send to the %print_fn channel.
%HTTP_accept$	Mime content types the browser will accept from you. */* means any content type.
%HTTP_accept_encoding$	Any inline encoding or compression methods the browser has available for your use.
%HTTP_accept_language$	Language the browser wants you to send in.
%HTTP_cookie$	The string of cookies received from the browser
%HTTP_referer$	Referral source if the browser was pointed to your website by some other location. The name of the URL which originated the request.
%HTTP_user_agent$	Details about the browser type connecting to you.
%ismas90	Value is 1 if using the Web Server inside Mas90's internet access module. Otherwise, value is 0 (zero).
%lang$	The LANG file extension in use in ProvideX (default is **`.en`**)

Web Server Variable	**Value(s):**
%new_location$	This is an internal variable, set during a redirection request. It should be used only when the user's browser should go to another page automatically. When a browser makes a request for a URL, you can force the browser to "go somewhere else" or make another request for something else by setting `%new_location` to the URL you actually want the browser to request and by setting your `%exit_code=301.` *You must set both these values to force a browser to automatically generate a new request for the URL in %NEW_LOCATION$.*
%path_translated$	The root directory plus the filename portion of the URL requested, with delimiters based on the O/S on which the Web Server is currently running (the backslash \ for Windows or standard slash / for UNIX).
%print_fn	The setting for the file channel the Web Server uses as a standard output channel. In effect, the Web Server sets up a memory-resident logical file on the %print_fn channel. You dump your data into this file by using PRINT, WRITE and other output directives with %print_fn as the file channel and the Web Server transmits the contents for you. For example: `PRINT (%print_fn)data` The task handler will read the `data` in this example when your application is complete and send it back to the port monitor for transmission to the browser. Note: **Your ProvideX web application should use ONLY this as the file channel for data that is sent to the browser.**
%query_string$	The string of arguments following the filename in the URL.
%raw_next$	If set, %raw_next will add a NEXT button to the bottom of your %print_fn data if `%raw_text=1`. (%raw_text$ should be set to the URL you wish the button to jump to.)
%raw_next_lbl$	The text to appear on your "next" button. The default is the word "next".
%raw_text	0=output to browser as is 1=wrap HTML header/body around output from ProvideX application
%raw_title$	If %raw_text is set, then %raw_title$ will be the title used in the HTML generated around your %print_fn output.
%remote_ip$	The IP address (or DNS name) of the browser.
%request_method$	The HTTP request method (set to GET, HEAD or POST).
%server_name$	The name of your host, as supplied by the browser connecting to you (for example, `www.pvx.com`).
%server_port	The TCP socket number the web server is monitoring
%server_protocol$	Set to `HTTP/1.0`
%server_rootdir$	The default root directory for the given server.
%server_software$	Name of web server and version
%SMTP_gateway$	The SMTP gateway you selected in your ***SMTP*** dialogue.
%SMTP_socket	SMTP socket number you specified in your ***SMTP*** dialogue, prefixed by a semicolon. The default is `;25`

Web Server Variable	Value(s):
%tcp_fn	File channel the task handler is using to talk to the port monitor. Warning: **Do NOT perform any I/O on the %TCP_FN channel!**
%time_offset	The offset to GMT time, e.g. `=-500` (-5 hrs) or `=430` (+4 hrs 30 minutes)

Notes*:* **When you create ProvideX applications to service web requests:**

The Web Server automatically CLOSEs any and all file channels upon completing your ProvideX program task, except for:

- any files belonging to the Web Server itself and
- any files which were open when the Web Server launched (any files you OPENed in your START_UP).

Regular variables are destroyed automatically when your application exits.

Please clear any GLOBAL variables you create in your application. Since the Web Server works by multiple task handling, the values won't be available between tasks, and are therefore useless as a method of passing information from page to page. (Use cookies to pass information from one request to another.)

Do NOT issue a START command.

ProvideX Web Server Utilities

Utilities

The ProvideX Web Server comes with a number of utilities to help you build your web applications quickly and easily. (Note that ProvideX converts the * asterisk in program names to the path to the ProvideX lib directory.) The utilities are presented in alphabetical order in the following sections.

See ***web/base64 To Encrypt or Decode Data, p. 34**

***web/cookie To Set, Send and Retrieve Cookies, p. 36**

***web/mail To Send Mail to a Local SMTP Server, p. 38**

***web/merge To Merge Dynamic Data with HTML Templates, p. 39**

***web/util HTML Coding Utility, p. 45**

Utility *web/base64

Format:

Basic Call Format **call "*web/base64;*entrypoint*",*required_arguments***

Where:	
entrypoint	is the entry point (line label) where execution in the *web/base64 subprogram starts for the given routine.
required_ arguments	is a string or numeric variable name. For each entry point of the base64 utility, there are different argument lists. See the chart below for details.

Description:

The *web/base64 program contains routines to encrypt data in base64 format and to decode it or convert it back.

The Base64 algorithm encodes 3 bytes (value 0-255 each) into 4 bytes (value 0-63 each) using plain ASCII characters A-Z,a-z,0-9,+/ to allow you to transmit binary files using text-only protocols across the Internet. Base64 is most commonly used for Mime encoding of e-mail attachments and for authorization codes in web pages.

The *web/base64 utility is a CALLed program with various entry points and various ENTER argument lists. See the chart below for a list of Formats for the various entry points, with arguments and descriptions.

Format:	**Directive and Description:**
1. Encode String	**`call "*web/base64;ENCODE_STR",instring$,outstring$`** For encoding strings
2. Decode String	**`call "*web/base64;DECODE_STR",instring$,outstring$`** For decoding strings

3. Encode File	This format has three variations for encoding files:
	`call "*web/base64;ENCODE_FILE",infile$,outfile$`
	`call "*web/base64;ENCODE_FILE",infile$,outfile$,blocksize`
	`call "*web/base64;ENCODE_FILE",infile$,outfile$,blocksize,EOL$`
	Block size and EOL (End of Line terminator) are optional. If you include EOL, you must also include block size.

4. Decode File	**`call "*web/base64;DECODE_FILE",infile$,outfile$`** For decoding files.

Notes: ***On Encoding and Decoding Files***

The following notes apply when you are encoding or decoding files.

Note 1: For Encode or Decode: Filenames or Channel Numbers

Your infile$ and outfile$ values in Formats 3 and 4 can be either filenames or logical file (channel) numbers. You must use strings to pass channel numbers (i.e. "24" would perform the operation on channel 24). If you use channel numbers instead of filenames, you can have a file on disk encoded to a *memory* file for later program handling. If you assign a channel number in either infile$ or outfile$, then that channel will remain open. The only channels automatically closed are the ones the *web/base64 program opens.

Note 2: For Encode or Decode: Append to Existing File, If Any

The base64 program READs an incoming file from its beginning (FIFO). With an output file (using either filename or channel number), the Web Server creates the file if it doesn't exist and appends to the file if it does exist. That is, it tacks the output onto the end of an ongoing encryption/decryption.

Note 3: For Encode Only: Block Size

You can include an optional block size parameter in your arguments when you encode files (Format 3). This won't have any affect on a disk file but does let you specify how big encoded record blocks will be when data output is written to a *memory* file. Each record your program WRITEs to the file has a separate index.

The primary use for a block size limit is in e-mail where lines must be less than or equal to 98 bytes each. If you assign block size when you encode files, then the Web Server automatically segments records for you. If you omit block size, the default is 4096 bytes per block.

Note 4: For Encode Only: EOL (End of Line Terminator)

You can assign an end of line terminator by including the eol$ parameter when you encode files. Your eol$ terminator should be either $0a$ or $0d0a$. The eol$ is only used for encoding the file. If you include an eol$, it will be appended to each block. If you omit block size, then no eol$ will be appended.

Base64 Error Messages

When the Web Server encounters errors in encoding or decoding Base64, it returns the usual file I/O and handling error messages, including:

```
Error #0: Record/file busy
Error #12: File does not exist (or already exists)
Error #46: Length of string invalid
```
(When encoded, string will exceed maximum string length allowed.)
```
Error #48: Invalid input -- Try again
```
(Unknown character encountered during decode. Indicates corrupted data, or data not in Base64 format.)

Utility *web/cookie

Formats for *web/cookie EntryPoints

1. *Return: EntryPoint=GET* **call "*web/cookie;GET",*number,name$,data$***

2. *Send: EntryPoint=SET* **call "*web/cookie;SET",*name$,data$,path$,expires$,domain$[,secure]***

Where:	
number	is the ordinal number of the given cookie (range: 1 to 300, size limit=4K each). **Note: The Web Server sets %cookie.rcv, p. 29 to the total number of cookies received or 0 (zero) for no cookies received.**
name$	is the name of the cookie (like a variable name)
data$	is the data in the given cookie.
path$	is the minimum match you want with a filepath before the cookie is sent back. (If no match is found, the cookie is not sent back from the browser.) The *path$* should be followed by a trailing forward slash (/). Examples: • the slash alone (`/`) would match any URL on your website • `/sales/orders/` would match `/sales/orders/myapp` and `/sales/orders` but would not match `/sales` or `/support/orders`
expires$	expires$ is the date (and/or time) the cookie will expire. See **The "expires$" Parameter, p. 37** for a list of acceptable values you can assign.
domain$	domain$ is the domain name for your host site. The cookie is only valid for the current domain/host. If domain$>"" then it must include a minimum of 2 periods. For example: • `domain$=".pvx.com"` is a cookie that is valid for any path$ in the pvx.com domain • `domain$="proxy.pvx.com"` is a cookie that is only valid for any path$ on the "proxy" machine in the pvx.com domain. If `domain$=""` then the cookie will only be valid for the current browser session. It will be cleared if the user moves to another site.
secure	Available in future version. If you include this parameter, the cookie will only be returned if the TCP connection is secure. The Boolean values are: =1 (one) if you only want the cookie returned when the connection is secure =0 (zero) otherwise. The value is 0 (zero) if you omit secure, as, for example, in: `call "*web/cookie;SET",name$,data$,path$,expires$,domain$`

Description:

You can use our Web Server *web/cookie utility to SET and/or GET cookies. Each cookie is a VARIABLE=DATA pair with return characteristics (path to match, domain, expiry and secure). Although web server requests are separate and unrelated and can't share variables or data on the server, you can get around this by using cookies to pass data and variables to the browser which in turn can send that information back for different requests.

When you SET a cookie, it's included in the header along with your data and sent back to the browser. The return characteristics determine when or if the browser will return the cookie with a request. All cookie parameters are cleared when the task is complete.

Format 1: **call "*web/cookie;GET",*number,name$,data$***

Use Format 1 to retrieve a cookie. Your CALL should include three arguments. You must pass the number of the cookie (1 to %cookie.rcv). The program will return the name$ and the data$.

Format 2*:* **call "*web/cookie;SET",*name$,data$,path$,expires$,domain$***

Use Format 2 to create and send a cookie. Your CALL should include five arguments:

`name$`	the name of the cookie
`data$`	the data to send to the browser
`path$`	your file path criteria for matching the URL's path
`expires$`	the cookie's expiry date and/or time
`domain$`	the domain name of your website

You can also set a sixth parameter to make sure the TCP connection is **`secure.`**

If the conditions warrant it, the browser will return the cookie to the server when your user requests specific web pages. For instance, you can use SET to assign the user's LOGIN ID / PASSWORD to a cookie, so that the user only has to enter it once. With cookies, you can also create shopping cart style applications that "remember" the items and quantities chosen.

Each VARIABLE=DATA pair is 1 cookie. You can SET up to 300 cookies, each with a size limit of 4K. The browser will return as many as it believes match a given request.

Note: **The internal variable %cookie.snd, p. 29 is incremented for each cookie you SET. Cookies are stored in strings in temporary variables as `%cookie.nnn$` where nnn is the cookie number.**

The "expires$" Parameter

The chart below lists valid values for the **`expires$`** parameter in the **`*web/cookie;SET`** subroutine.

If	**Then:**
`expires$=""`	no expiry date is sent and the cookie will expire at the end of the current browser session.
`expires$="0"`	the date-time equivalent of RIGHT NOW will be sent.
`expires$=` "YYYYMMDDHHMMSS"	expiry is set to a specific date. The date must be in the format YYYYMMDDHHMMSS: for example, **`expires$="19990709143030"`** (HH is in 24 hour time).
`expires$="+nnnn"` or **`expires$="-nnnn"`**	the cookie will expire NOW plus or minus nnnn time units. The default time unit is seconds, but you can append an alpha code to change or override the current setting (Y=years, D=days, H=hours, M=minutes, S=seconds). **Examples:** **`expires$="+1Y"`** sets an expiry 1 year from NOW **`expires$="+30D"`** sets an expiry 30 days from NOW **`expires$="+5H"`** sets an expiry 5 hours from NOW

Utility *web/mail

Format:

call "*web/mail", SMTP_server$, From$, To$, Subject$, MessageText$, Return$

Where:	
SMTP_server$	is the IP address or DNS of the SMTP server. Note that TCP port `";25"` is forced.
From$	is the e-mail address of the person or application sending the mail.
To$	is a list of recipient e-mail addresses separated by the SEP character ($8a$).
Subject$	is the subject line of the e-mail.
MessageText$	is the text of the message. NOTE: **MessageText$ lines must end with $0d0a$ and must not exceed 100 characters per line, including the $0d0a$ terminator.**
Return$	If an error occurs in sending the message, the response from the SMTP server will be in Return$.

Description:

You can use the *web/mail utility for sending e-mail from a ProvideX application to a local SMTP Mail Server.

*web/merge To Merge Dynamic Data with HTML Templates

Utility ***web/merge**

Format:

Special Codes in HTML Page ***~~argument**[**?suffix**]**~~***

Where:	
~~ ... ~~	double tildes enclose special codes

argument	is a string or numeric variable name. You can use any variable in any style (local, global, system, array). For example:
	`~~address$~~`
	`~~total~~`
	`~~DAY~~`
	For some suffixes, your argument is the name of a program the merge utility will perform. For instance, for the ?T suffix, it's the program to fill the table which immediately follows your **`~~prog;stmtref?T~~`**. If there is no suffix, the argument is a simple variable name.

?suffix	is an optional suffix. If you include a suffix, your argument must be valid for the special processing done by the suffix declaration. Valid suffixes are: *?+* CheckBox / RadioButton is initialized based on the value in your argument variable. (Values: 1=ON, 0 (zero) =OFF) *?A* Do not run value through HTML escape code translator (i.e. don't perform HTML encoding: <=< >=&gt; "=&quot; &=&amp;) *?F:xxx* Convert using format mask specified (uses STR( ) function) *?L* Perform URL encoding (i.e. $20$+$8a$ becomes %20%8A, etc.) *?P* Perform the program name given. For example, ~~my_prog?P~~. Note that any performed program must have output in HTML format and must do its own escape coding. The Web Server won't check or convert output. *?X* Evaluate argument as string expression and substitute result *There are also three valid Table suffixes:* *?CT* (Colour Table) is a table suffix enhancement that allows you to set alternate row background colours. *?JT[/H]* (Jump Table) is a table suffix enhancement that allows you to fill the table with as many records as available up to the maximum number of records you specify. Use /H for custom positioning of hyperlinks. *?T* The basic Table definition suffix includes the name of your program for the merge utility to perform to fill the table which immediately follows your ~~prog;stmtref?T~~. See also <u>**Table Suffixes, p. 40**</u>. **Examples:** **`~~Total?F:####0.00~~`** **`~~dte(jul(0,0,0):"%Mz/%Dz/%Yl")?X~~`** **`~~myprog;filltable?T~~`**

Description:

A template is an HTML page with special coding embedded in it. You can create templates using any HTML editor. A ProvideX Web Server utility, the *web/merge program, reads your HTML page, scanning for specialty codes (arguments enclosed in double tildes: for example, **`~~total?F:####0.00~~`**). It evaluates the special codes embedded in the HTML page, substituting data as necessary, and sends the dynamically-created page to the output channel %print_fn, p. 31.

Note: **As a prerequisite to running the *web/merge program, set the value in HTML_FILE$ to the path / filename of your HTML template page.**

The Web Server performs the *web/merge program when you RUN it, so variables, files, etc. are available throughout.

Example:

```
0010 ! home.html - A misdirection
0020 LET MESS1$="Welcome",MESS2$="Please sign on",USERID$="",USERPASSWD$=""
0030 LET HTML_FILE$="orders/login.htm"  ! See Orders Web Page 1: login.htm, p. 52.
0040 RUN "*web/merge"
```

Table Suffixes

You can use a program name and entry point label as your argument with ?T (and with the ?CT and ?JT suffixes) in your HTML page or template, as in **`~~myprog;filltable?T~~`**. Your program should OPEN any files required, read the data needed to fill one row of your given table, and then EXIT, leaving the files OPEN and the record pointers intact. Your program should CLOSE the files and EXIT 2 (exit with an Error 2) at the end of the records (EOF). This will tell the ?T table logic to complete the code necessary to finish processing the HTML template.

For the example **`~~myprog;filltable?T~~`**, the merge program executes the specialty code by running the program **`myprog`** starting at the **`filltable:`** entry point. Each cell in your table would have a variable name (like ~~Desc$~~ ~~QTY~~ etc.) for the values to be returned when the program you name in your argument READs the file(s).

See also: Basic HTML Template Creation: Example, p. 41.

In Program "myprog;filltable"

The merge program starts execution at the **`filltable:`** line label in **`myprog`** in this example. The program OPENs the files and READs the first record to obtain a row's worth of data. It leaves the files open and EXITs back to the merge program. The merge logic fills the cells of the row and returns control to **`myprog;filltable.`** The program reads more data and EXITs back to the merge program's table logic (which fills in the cells for the next row). This continues until **`myprog;filltable`** does an EXIT 2, which indicates to the merge program that the end of the table has been reached. The merge program's table logic then finishes off the HTML table coding and proceeds to process the rest of the template.

```
0080 !
0090 filltable:
0100 if %myfile=0 then %myfile=hfn; open(%myfile)"myfile"
0130 read(%myfile,end=50)a$,b,desc$,qty
0140 exit
0150 close(%myfile);%myfile=0;exit 2
```

See also: Basic HTML Template Creation: Example, p. 41

The ?CT (Colour Table) Suffix

The ?CT table suffix lets you create a table with alternating background colours for the rows. For instance, for **`~~myprog;stmtref?CT~~`**, the ProvideX default is to alternate silver background (RGB 192,192,192) for the odd numbered rows and gray background (RGB 222,222,222) for the even numbered rows.

You can set one or both of the background colours to your own colour choice. If you set only the first colour, *web/merge creates the second colour by using a lighter shade of your first choice. Please note that some colours work better than others for alternating background shades.

To Set Row Background Colours:

In the program you use to load the table (`myprog`, in the example), use the `COLOR:` statement label as a keyword to notify *web/merge that you are assigning your own background colour(s) for alternating rows. Then, use your COLOR: routine to assign one or both colours to the *web/merge COLOR$ array, using the # pound symbol followed by the hex equivalents of your RGB values. For example, for silver (RGB 192,192,192), you would use #C0C0C0. The example below just reverses the order of the defaults:

```
2500 COLOR:
2510 LET COLOR$[1]="#323232" ! Gray
2520 LET COLOR$[2]="#C0C0C0" ! Silver
```

The ?JT (Jump Table) Suffix

The ?JT table suffix lets you fill your table with as many records as are available, up to the maximum number of records. (This differs from the ?T suffix, which fills the table using ALL the records, stopping only when ProvideX encounters your EXIT 2 directive.) Use the numeric variable `MAX_ROW` (keyword, default = 100) to set the maximum number of rows you want returned for each page produced using the ?JT suffix.

By default, a jump table is followed by default hyperlinks of next and previous MAX_ROW records, centred under the table. For greater control of hyperlink placement use the ?JT/H suffix. Then, instead of being centred directly under the table, the hyperlinks will be put where you place the following keywords %FRWD_HYPLINK$ and %BACK_HYPLINK$ for the specified table in your HTML template. For instance, to place the hyperlinks at the left margin:

```
~~myprog;filltable?JT/H~~
%back_hyplink$
%FRWD_HYPLINK$
```

Example:

For a ?JT Jump Table, you must READ your records based on a key=JK$ (a specialty variable). It's important to READ immediately after OPENing your file so that *web/merge can save the relevant next and previous hyperlinks. For example:

```
0510 OPEN (CHANNEL)"YOUR_FILE"
0520 READ (CHANNEL,ERR=*NEXT)
0530 GOTO YOUR_LOAD_ROUTINE
...
2000 YOUR_LOAD_ROUTINE:
2010 LET JK$=KEY(CHANNEL)
2020 READ (CHANNEL,KEY=JK$,ERR=*NEXT)
```

Basic HTML Template Creation: *Example*

The next example of basic HTML template creation was designed using Frontpage for the ProvideX ***DireXions*** Technical Conference in March, 1999. See also the DireXions Examples (Appendix), p. 47.

```
<html>

<head>
<title>Exercise - Basic Template Creation and Usage</title>
</head>
<body background="images/whttxtr1.jpg">
<div align="right">

<table border="1" cellpadding="3" width="100%">
  <tr>
    <td width="100%" colspan="2" bgcolor="#000000"><p
align="center"><big><big><big><strong><font
    color="#FFFFFF" face="Arial">The ProvideX Web Server Workshop</font></
```

```
strong></big></big></big></td>
  </tr>
  <tr>
    <td width="50%"><font face="Tahoma">Your IP Address is: ~~%remote_ip$~~</
font></td>
    <td width="50%"><font face="Tahoma">You've visited this page:
~~%hit_counter~~ times.</font></td>
  </tr>
</table>
</div><div align="right">

<table border="1" cellpadding="3" width="100%">
  <tr>
    <td width="66%" bgcolor="#000080"><font face="Tahoma"
color="#80FFFF"><strong>Directory
    Browse of: ~~%document_uri$~~</strong></font></td>
    <td width="34%" bgcolor="#000080"><font face="Tahoma"
color="#80FFFF"><strong>As of:
    ~~dte(0:&quot;%Mz/%Dz/%Yl %hz:%mz:%sz %P&quot;)?X~~</strong></font></td>
  </tr>
</table>
</div><div align="right">

<table border="1" cellpadding="3" width="100%">
  <tr>
    <td width="33%"><strong><font face="Tahoma">File Name</font></strong></td>
    <td width="33%"><strong><font face="Tahoma">Size (in bytes)</font></
strong></td>
    <td width="34%"><strong><font face="Tahoma">Modification Date</font></
strong></td>
  </tr>
</table>
</div>

<p>~~webpvx03;load_dir?T~~</p>
<div align="right">

<table border="1" cellpadding="3" width="100%">
  <tr>
    <td width="33%">~~filname$?A~~</td>
    <td width="33%">~~filtype$~~</td>
    <td width="34%">~~fildate$~~</td>
  </tr>
</table>
</div>

<p>If you have any problems with this web page, please contact our <a
href="mailto:webmaster@pvx.com">webmaster</a>.</p>
</body>
</html>
```

Basic Template Filling: ***Example***

```
0010 ! webpvx03 - Basic Template Filling - A Directory Browse
0020 !
0030 LET NOW$=DTE(0:"%Mz/%Dz/%Yz %h:%mz:%sz %P")
0040 LET HTML_FILE$="basetmpl.htm"
0050 PERFORM "*web/merge"
```

```
0060 END
0070 !
0080 !
0090 LOAD_DIR:
0100 IF %DIR=0 THEN LET %DIR=HFN; OPEN INPUT (%DIR)"."
0110 READ (%DIR,END=LOAD_DIR_DONE)FILNAME$
0120 LET CHAN=HFN; OPEN INPUT (CHAN,ERR=LOAD_DIR_NEXT)FILNAME$
0130 LET FILNAME$="<a href="+QUO+FILNAME$+QUO+">"+FILNAME$+"</a>"
0140 IF MID(FIB(CHAN),19,1)="D" THEN LET FILTYPE$="<Dir>" ELSE LET FILTYPE$=STR
0140:(DEC($00$+MID(FIN(CHAN),1,4)):"###,###,##0")
0150 LET FILDATE$=DTE(JUL(1970,1,1)+DEC($00$+MID(FIN(CHAN),5,4))/60/60/24,*:"%M
0150:z/%Dz/%Yz %h:%mz:%sz %P")
0160 LOAD_DIR_NEXT:
0170 CLOSE (CHAN)
0180 EXIT
0190 !
0200 !
0210 !
0220 LOAD_DIR_DONE:
0230 CLOSE (%DIR); LET %DIR=0; EXIT 2
```

Using Your HTML Editor: *Example*

You can use any HTML editor to create your web pages and templates. In this example, Microsoft Word was used to create a document page containing text, a graphic and a table, as follows:

This is a Word example, creating an HTML page
{EMBED Paint.Picture }
~~myprog;filltable?T~~

~~Description$~~	**~~Quantity~~**	**~~Price~~**	**~~Extended~~**

When the document page was ***Saved As HTML***, Word generated the HTML code below:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
<META NAME="Generator" CONTENT="Microsoft Word 97">
<TITLE>test</TITLE>
<META NAME="Template" CONTENT="C:\PROGRAM FILES\MSWORKS\OFFICE\html.dot">
</HEAD>
<BODY LINK="#0000ff" VLINK="#800080">

<FONT FACE="Arial" SIZE=2><P>This is a Word example, creating an HTML page.</P>
</FONT><P><IMG SRC="Image1.gif" WIDTH=196 HEIGHT=197></P>
<FONT FACE="Arial" SIZE=2><P>~~myprog;filltable?T~~</P></FONT>
<TABLE CELLSPACING=0 BORDER=0 CELLPADDING=7 WIDTH=732>
<TR><TD WIDTH="25%" VALIGN="TOP" HEIGHT=30>
<FONT FACE="Arial" SIZE=2><P>~~Description$~~</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP" HEIGHT=30>
<FONT FACE="Arial" SIZE=2><P>~~Quantity~~</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP" HEIGHT=30>
<FONT FACE="Arial" SIZE=2><P>~~Price~~</FONT></TD>
<TD WIDTH="25%" VALIGN="TOP" HEIGHT=30>
<FONT FACE="Arial" SIZE=2><P>~~Extended~~</FONT></TD>
</TR>
<TR><TD WIDTH="25%" VALIGN="TOP" HEIGHT=30><P></P></TD>
<TD WIDTH="25%" VALIGN="TOP" HEIGHT=30><P></P></TD>
<TD WIDTH="25%" VALIGN="TOP" HEIGHT=30><P></P></TD>
<TD WIDTH="25%" VALIGN="TOP" HEIGHT=30><P></P></TD>
```

```
</TR>
</TABLE>

<FONT FACE="Arial" SIZE=2><P>&nbsp;</P></FONT></BODY>
</HTML>
```

For more information on using your particular HTML editor, refer to your HTML editor’s manual.

Utility *web/util

Format:

Basic call Format **call "*web/util;*entrypoint*",*required_arguments***

Where:	
entrypoint	is the entry point (line label) where execution in the *web/util subprogram starts for the given routine.
required_ argument	is a string or numeric variable name. For each entry point of the utility, there are different argument lists. See the chart below for details.

Description:

CALL the Webserver's *web/util program to simplify your coding when you build your HTML pages in ProvideX applications. The *web/util program contains miscellaneous routines to put various HTML codes into the %print_fn data channel for you. This is a CALLed program with various entry points and various ENTER argument lists. See the chart below for a list of Formats for the various entry points, with arguments and descriptions of the codes that are added to %print_fn.

See also **%print_fn, p. 31**.

To Add to $PRINT_FN:	***web/util CALL EntryPoints**
1. Begin HTML	`call "*web/util;BEGIN_HTML",title$` • Adds HTML code starting with the <html><header> tag and going through to the </header><body> tags. • Adds title$ value as a page title.
2. Closing Sequence	`call "*web/util;END_HTML"` • Adds the closing sequences </body></html>. • Auto-closes any of the following: • links set byBEGIN_LINK • paragraphs set by NEW_PARA • forms set by BEGIN_FORM
3. Begin Link	`call "*web/util;BEGIN_LINK",url$` Puts in a link and anchor tag for the given url$
4. End Link	`call "*web/util;END_LINK"` Adds closing the anchor tag
5. Begin Font	`call "*web/util;BEGIN_FONT",name$,pt_size[,bold]` Adds the begin_new_font tag for your given font name and point size. Bold is a Boolean value: 0 (zero) if omitted, 1 if included.
6. End Font	`call "*web/util;END_FONT"` Adds the closing tags to end the font tag created by BEGIN_FONT.
7. Begin Form	`call "*web/util;BEGIN_FORM",form_name$,url$` • Closes a previous paragraph and form (as necessary), before creating a new form. • Creates a new form tag, with the given form name, links by POST to the given url$

To Add to $PRINT_FN:	*web/util CALL EntryPoints
8. End Form	**`call "*web/util;END_FORM"`** Adds the closing tags for paragraph and form (as necessary).
9. Input Field	**`call "*web/util;INPUT_FIELD",field_name$,size`** Creates an input field ("TEXT" type) with field_name$ and width.
10. Input Required	**`call "*web/util;INPUT_REQD_FIELD",field_name$,size`** Same as INPUT_FIELD, but puts in validation code for field_name$!=null
11. Input Area	**`call "*web/util;INPUT_AREA",field_name$,cols,lines`** Creates an input field ("TEXTAREA" multiline type) with the given number of columns and rows.
12. Submit Button	**`call "*web/util;SUBMIT_BUTTON",text$`** Creates a submit_action button for a form. The text$ is the button's description.
13. Reset Button	**`call "*web/util;RESET_BUTTON",text$`** Create a reset_action button for a form. The text$ is the button's description.
14. New Paragraph	**`call "*web/util;NEW_PARA"`** Creates a new paragraph tag (and closes a pre-existing one as necessary).
15. Program List	**`call "*web/util;LIST"`** The utility lists itself to the %print_fn channel.

DireXions Examples (Appendix)

DireXions Examples

Description:

The ProvideX programs and Frontpage HTML web page codes in the examples here were created for the ProvideX ***DireXions*** Technical Conference in March, 1999. The programs and HTML page applications LISTed in this appendix are illustrations for those who don't have access to the DireXions CD. For more information on creating HTML, refer to your HTML editor's manual (e.g Frontpage, Word, WordPerfect, etc.).

DireXions attendees can run the Orders Demonstration from the ProvideX 4.10 DireXions CD by launching it through **`http://127.0.0.1:4000/orders/`**. Programs and web pages from the demonstration are included here as illustrations for those without access to the CD.

Orders Program 1: home.htm

DireXions Example **orders\home.htm**

Description:

This is listing of the ProvideX program that redirects the user to the LOGIN page in the DireXions Order Demonstration.

See also: **Orders Web Page 1: login.htm, p. 52**

```
0010 ! home.html - A misdirection
0020 LET MESS1$="Welcome",MESS2$="Please sign on",USERID$="",USERPASSWD$=""
0030 LET HTML_FILE$="orders/login.htm"
0040 RUN "*web/merge"
```

NOTE: **Just because a file has a name that ends in .HTML does not mean that the Web Server will have to treat it as such. In the above example, the file named HOME.HTML is, in reality, a ProvideX program which would be RUN.**

Orders Program 2: webord

DireXions Example **orders\webord**

Description:

This ProvideX program deals with the user LOGIN process. It jumps to the Profile page in the case of a new user. When a user signs on, a cookie is set so that order pages will identify the user.

```
0010 ! webord - Web Based Order Status
0020 ! by Gord Davey - ProvideX Technologies
0030 !
0040 LOGON:
0050 LET EXIT$=STP(EXIT$,2),NEWUSER$=STP(NEWUSER$,2),SIGNON$=STP(SIGNON$,2)
0060 LET USERID$=STP(USERID$,2),USERPASSWD$=STP(USERPASSWD$,2)
0070 !
0080 ! The following line happens when they hit the exit button
0090 ! The value of exit$ is > null when they do
0100 ! So, we can set an %exit_code to 301 which is a URL redirection
0110 ! This causes the browser to jump to the new location given.
0120 ! but because its a pvx app, the redirection will expire
0130 ! right away, so the browser will let them come back here later.
0140 !
0150 IF EXIT$>"" THEN LET %EXIT_CODE=301; LET %NEW_LOCATION$="../"; EXIT
0160 !
0170 IF SIGNON$>"" AND (USERID$="" OR USERPASSWD$="") THEN GOTO NOT_ENOUGH_INFO
0180 !
0190 IF NEWUSER$>"" THEN RUN "orders/profile;new_user"
0200 !
0210 VALIDATE_USERID:
0220 LET USERFIL=HFN; OPEN (USERFIL,ERR=USERFILE_NOT_FOUND)"orders/userfil.dat"
0230 READ (USERFIL,KEY=USERID$,ERR=USER_NOF)*,PASSWD$
0240 IF USERPASSWD$<>PASSWD$ THEN GOTO BAD_PASSWD
0250 !
0260 ! user id and passwd are on file
0270 ! Set cookie for 1 Day, so they don't have to enter it anymore
0280 CALL "*web/cookie;set","order-userid",USERID$,"/orders/","+1D","",0
0290 RUN "orders/ordstat"
0300 !
0310 USERFILE_NOT_FOUND:
0320 LET MESS1$="Internal Error",MESS2$="User ID file not available. Please con
0320:tact the webmaster"; GOTO SEND_BACK
0330 !
0340 NOT_ENOUGH_INFO:
0350 LET MESS1$="Invalid",MESS2$="Please enter both User ID and Password"
0360 GOTO SEND_BACK
0370 !
0380 USER_NOF:
0390 LET MESS1$="Invalid",MESS2$="User ID is not correct."
0400 GOTO SEND_BACK
0410 !
0420 BAD_PASSWD:
0430 LET MESS1$="Invalid",MESS2$="Incorrect password."
0440 GOTO SEND_BACK
0450 !
0460 SEND_BACK:
0470 IF USERFIL THEN CLOSE (USERFIL)
0480 LET HTML_FILE$="orders/login.htm"
0490 RUN "*web/merge"
```

Orders Program 3: ordstat

DireXions Example **orders\ordstat**

Description:

This is a listing of the ProvideX program that sets up a simple order status information page in the DireXions application, based on the UserID retrieved from the cookie. It also illustrates coding to jump from page to page.

```
0010 ! ordstat - Order Status Page
0020 !
0030 LET EXIT$=STP(EXIT$,2),GETPROFILE$=STP(GETPROFILE$,2)
0040 IF EXIT$>"" THEN RUN "orders/home.htm"
0050 IF GETPROFILE$>"" THEN RUN "orders/profile;current_user"
0060 !
0070 GOSUB CHECK_COOKIE
0080 IF USERID$="" THEN GOTO USER_NOT_FOUND
0090 LET PROF=HFN; OPEN (PROF,IOL=*)"orders/profile.dat"
0100 READ (PROF,KEY=USERID$,ERR=USER_NOT_FOUND)
0110 !
0120 ! got shipto info, we would look up the orders here
0130 LET HTML_FILE$="orders/ordstat.htm"
0140 RUN "*web/merge"
0150 !
0160 USER_NOT_FOUND:
0170 RUN "orders/home.htm"
0180 !
0190 CHECK_COOKIE:
0200 FOR X=1 TO %COOKIE.RCV
0210 CALL "*web/cookie;get",X,VAR$,VALUE$
0220 IF VAR$="order-userid" THEN LET USERID$=VALUE$
0230 NEXT X
0240 RETURN
```

Orders Program 4: profile

DireXions Example **orders\profile**

Description:

This application submits the Profile page and provides an information Lookup in cases where the user is jumping to the page from somewhere else. It's also an example of how to look for a cookie from the user's browser and display the user information.

```
0010 ! profile - Web Based Profile Entry
0020 !
0030 PROFILE_SUBMIT:
0040 LET OK$=STP(OK$,2),CANCEL$=STP(CANCEL$,2)
0050 IF CANCEL$>"" THEN GOTO USER_NOT_FOUND
0060 LET PROF=HFN; OPEN (PROF,IOL=*)"orders/profile.dat"
0070 LET USERFIL=HFN; OPEN (USERFIL)"orders/userfil.dat"
0080 IF PASSWD1$<>PASSWD2$ THEN LET MESS1$="Passwords do NOT match.",HTML_FILE$
0080:="orders/profile.htm"; CLOSE (PROF); CLOSE (USERFIL); RUN "*web/merge"
0090 REMOVE (PROF,KEY=USERID$,ERR=*NEXT)
0100 REMOVE (USERFIL,KEY=USERID$,ERR=*NEXT)
0110 WRITE (PROF)
0120 WRITE (USERFIL,KEY=USERID$)USERID$,PASSWD1$
0130 CLOSE (PROF); CLOSE (USERFIL)
0140 RUN "orders/home.htm"
0150 !
0160 NEW_USER:
0170 LET USERID$="",HTML_FILE$="orders/profile.htm"; RUN "*web/merge"
0180 !
0190 CURRENT_USER:
0200 GOSUB CHECK_COOKIE
0210 IF USERID$="" THEN GOTO USER_NOT_FOUND
0220 LET PROF=HFN; OPEN (PROF,IOL=*)"orders/profile.dat"
0230 LET USERFIL=HFN; OPEN (USERFIL)"orders/userfil.dat"
0240 READ (PROF,KEY=USERID$,ERR=USER_NOT_FOUND)
0250 READ (USERFIL,KEY=USERID$)*,PASSWD$
0260 LET PASSWD1$=PASSWD$,PASSWD2$=PASSWD$
0270 LET HTML_FILE$="orders/profile.htm"
0280 RUN "*web/merge"
0290 !
0300 USER_NOT_FOUND:
0310 RUN "orders/home.htm"
0320 !
0330 CHECK_COOKIE:
0340 FOR X=1 TO %COOKIE.RCV
0350 CALL "*web/cookie;get",X,VAR$,VALUE$
0360 IF VAR$="order-userid" THEN LET USERID$=VALUE$
0370 NEXT X
0380 RETURN
```

Orders Web Page 1: login.htm

DireXions Example **orders\login.htm**

Description:

This is a listing of the HTML for the DireXions **`login.htm`** web page application:

```
<html>

<head>
<title>ProvideX - Order System - Login</title>
</head>

<body background="../images/whttxtr1.jpg">
<div align="left">

<table border="0" cellpadding="0" width="737" cellspacing="0">
  <tr>
      <td><img src="../images/pvxbird2.jpg" alt="pvxbird2.jpg (13155 bytes)
      "WIDTH="737" HEIGHT="90"></td>
  </tr>
  <tr>
      <td height="50" bgcolor="#000080"><p align="center"><strong><big><big><big><font
      color="#80FFFF" face="Tahoma">On-Line Order System</font></big></big></big><stro
      ng></td>
  </tr>
  <tr>
      <td height="25"><p align="center"><br>
      <font color="#FF0000" face="Arial"><big><strong><big>~~mess1$~~ - </
      big></strong>~~mess2$~~</big><br></font></td>
  </tr>
  <tr>
    <td><form method="POST" action="/orders/webord;logon">
     <div align="center"><center><table border="0" cellpadding="3" width="50%"
      cellspacing="1">
        <tr>
            <td width="50%" valign="middle" align="right" nowrap><font face
            ="Arial" size="5"><strong>UserID:</strong></font></td>
            <td width="50%" valign="middle" align="left"><input type="text"
            name="userid" size="20" value="~~userid$~~"></td>
        </tr>
        <tr>
            <td width="50%" valign="middle" align="right" nowrap><font face=
            "Arial" size="5"><strong>Password:</strong></font></td>
            <td width="50%" valign="middle" align="left"><input type="passwo
            rd" name="userpasswd" size="20" value="~~userpasswd$~~"></td>
        </tr>
        <tr>
          <td width="50%" valign="middle" align="right" nowrap>&nbsp; </td>
          <td width="50%" valign="middle" align="left">&nbsp; </td>
        </tr>
        <tr>
            <td width="100%" colspan="2" valign="middle" align="center"><st
            rong><font face="Arial" size="3"><input type="submit" value="
            Sign On                        " name="signon" style="font-family:
            sans-serif, serif; font-size: 14pt; font-weight: bold"></font></
            strong></td>
        </tr>
        <tr>
            <td width="50%" valign="middle" align="right"><strong><font
            face="Arial" size="3"><input type="submit" value="      New User
            " name="newuser" style="font-family: sans-serif, serif; font-size:
            14pt; font-weight: bold"></font></strong></td>
            <td width="50%" valign="middle" align="left"><strong><font
            face="Arial" size="3"><input type="submit" value="
            Exit             " name="Exit" style="font-family: sans-serif,
            serif; font-size: 14pt; font-weight: bold"></font></strong></td>
        </tr>
      </table>
      </center></div>
    </form>
    </td>
  </tr>
  <tr>
    <td height="25"></td>
  </tr>
  <tr>
      <td height="50" bgcolor="#000080"><p align="center"><font color="#80F
      FFF" face="Tahoma"><strong><big><big><big>Please Sign-On</big></big></
      big></strong></font></td>
  </tr>
</table>
</div>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>
</body>
</html>
```

Orders Web Page 2: profile.htm

DireXions Example **orders\profile.htm**

Description:

The **`profile.htm`** web page application illustrates a variety of HTML codes.

```
<html>

<head>
<title>ProvideX - Order System - Personal Profile</title>
</head>

<body background="../images/whttxtr1.jpg">
<div align="left">

<table border="0" cellpadding="0" width="737" cellspacing="0">
  <tr>
    <td><img src="../images/pvxbird2.jpg" alt="pvxbird2.jpg (13155 bytes)" WI
    DTH="737" HEIGHT="90"></td>
  </tr>
  <tr>
    <td height="50" bgcolor="#000080"><p align="center"><font color="#80FFFF"
    face="Tahoma"><strong><big><big><big>Personal Profile</big></big></big></
    strong></font></td>
  </tr>
  <tr>
    <td height="25"><p align="right"><font face="Tahoma"><small>&nbsp;</small>
    <img src="../images/blueball.gif" alt="blueball.gif (262 bytes)" WIDTH="1
    5" HEIGHT="15">&nbsp; <small>Denotes a Mandatory Field</small></font></td>
  </tr>
  <tr>
    <td><!--webbot BOT="GeneratedScript" PREVIEW=" " startspan --><script Lang
    uage="JavaScript"><!--function FrontPage_Form1_Validator(theForm)
{

  if (theForm.UserID.value.length < 4)
  {
    alert("Please enter at least 4 characters in the \"User ID\" field.");
    theForm.UserID.focus();
    return (false);
  }

  if (theForm.UserID.value.length > 10)
  {
    alert("Please enter at most 10 characters in the \"User ID\" field.");
    theForm.UserID.focus();
    return (false);
  }

  var checkOK = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzƒŠŒŽš
  œžŸÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþ01234567
  89-";
  var checkStr = theForm.UserID.value;
  var allValid = true;
  for (i = 0;  i < checkStr.length;  i++)
  {
```

```
  ch = checkStr.charAt(i);
  for (j = 0;  j < checkOK.length;  j++)
    if (ch == checkOK.charAt(j))
      break;
  if (j == checkOK.length)
  {
    allValid = false;
    break;
  }
}
if (!allValid)
{
  alert("Please enter only letter and digit characters in the \"User ID\"
  field.");
  theForm.UserID.focus();
  return (false);
}

if (theForm.passwd1.value.length < 4)
{
  alert("Please enter at least 4 characters in the \"Password\" field.");
  theForm.passwd1.focus();
  return (false);
}

if (theForm.passwd1.value.length > 10)
{
  alert("Please enter at most 10 characters in the \"Password\" field.");
  theForm.passwd1.focus();
  return (false);
}

var checkOK = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzƒŠŒŽš
œžŸÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþ01234567
89-";
var checkStr = theForm.passwd1.value;
var allValid = true;
for (i = 0;  i < checkStr.length;  i++)
{
  ch = checkStr.charAt(i);
  for (j = 0;  j < checkOK.length;  j++)
    if (ch == checkOK.charAt(j))
      break;
  if (j == checkOK.length)
  {
    allValid = false;
    break;
  }
}
if (!allValid)
{
  alert("Please enter only letter and digit characters in the \"Password\"
  field.");
  theForm.passwd1.focus();
  return (false);
}

if (theForm.passwd2.value.length < 4)
```

```
{
  alert("Please enter at least 4 characters in the \"Confirmation Password\"
  field.");
  theForm.passwd2.focus();
  return (false);
}

if (theForm.passwd2.value.length > 10)
{
  alert("Please enter at most 10 characters in the \"Confirmation Password\"
  field.");
  theForm.passwd2.focus();
  return (false);
}

var checkOK = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzƒŠŒŽš
œžŸÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþ01234567
89-";
var checkStr = theForm.passwd2.value;
var allValid = true;
for (i = 0;  i < checkStr.length;  i++)
{
  ch = checkStr.charAt(i);
  for (j = 0;  j < checkOK.length;  j++)
    if (ch == checkOK.charAt(j))
      break;
  if (j == checkOK.length)
  {
    allValid = false;
    break;
  }
}
if (!allValid)
{
  alert("Please enter only letter and digit characters in the \"Confirmation
  Password\" field.");
  theForm.passwd2.focus();
  return (false);
}

if (theForm.FirstName.value == "")
{
  alert("Please enter a value for the \"First Name\" field.");
  theForm.FirstName.focus();
  return (false);
}

if (theForm.FirstName.value.length < 1)
{
  alert("Please enter at least 1 characters in the \"First Name\" field.");
  theForm.FirstName.focus();
  return (false);
}

if (theForm.LastName.value == "")
{
  alert("Please enter a value for the \"Last Name\" field.");
  theForm.LastName.focus();
```

```
    return (false);
  }

  if (theForm.LastName.value.length < 1)
  {
    alert("Please enter at least 1 characters in the \"Last Name\" field.");
    theForm.LastName.focus();
    return (false);
  }

  if (theForm.CompName.value == "")
  {
    alert("Please enter a value for the \"Company Name\" field.");
    theForm.CompName.focus();
    return (false);
  }

  if (theForm.CompName.value.length < 1)
  {
    alert("Please enter at least 1 characters in the \"Company Name\" field.");
    theForm.CompName.focus();
    return (false);
  }

  if (theForm.Addr1.value == "")
  {
    alert("Please enter a value for the \"Address\" field.");
    theForm.Addr1.focus();
    return (false);
  }

  if (theForm.Addr1.value.length < 1)
  {
    alert("Please enter at least 1 characters in the \"Address\" field.");
    theForm.Addr1.focus();
    return (false);
  }

  if (theForm.City.value == "")
  {
    alert("Please enter a value for the \"City\" field.");
    theForm.City.focus();
    return (false);
  }

  if (theForm.City.value.length < 1)
  {
    alert("Please enter at least 1 characters in the \"City\" field.");
    theForm.City.focus();
    return (false);
  }

  if (theForm.Province.value.length < 1)
  {
    alert("Please enter at least 1 characters in the \"Province\" field.");
    theForm.Province.focus();
    return (false);
  }
```

```
if (theForm.Country.value == "")
{
  alert("Please enter a value for the \"Country\" field.");
  theForm.Country.focus();
  return (false);
}

if (theForm.Country.value.length < 1)
{
  alert("Please enter at least 1 characters in the \"Country\" field.");
  theForm.Country.focus();
  return (false);
}

if (theForm.Telephone.value == "")
{
  alert("Please enter a value for the \"Telephone Number\" field.");
  theForm.Telephone.focus();
  return (false);
}

if (theForm.Telephone.value.length < 7)
{
  alert("Please enter at least 7 characters in the \"Telephone Number\"field
  .");
  theForm.Telephone.focus();
  return (false);
}

var checkOK = "0123456789-+-() \t\r\n\f";
var checkStr = theForm.Telephone.value;
var allValid = true;
for (i = 0;  i < checkStr.length;  i++)
{
  ch = checkStr.charAt(i);
  for (j = 0;  j < checkOK.length;  j++)
    if (ch == checkOK.charAt(j))
      break;
  if (j == checkOK.length)
  {
    allValid = false;
    break;
  }
}
if (!allValid)
{
  alert("Please enter only digit, whitespace and \"+-()\" characters in the
  \"Telephone Number\" field.");
  theForm.Telephone.focus();
  return (false);
}

var checkOK = "0123456789-+-() \t\r\n\f";
var checkStr = theForm.Fax.value;
var allValid = true;
for (i = 0;  i < checkStr.length;  i++)
{
  ch = checkStr.charAt(i);
  for (j = 0;  j < checkOK.length;  j++)
    if (ch == checkOK.charAt(j))
      break;
  if (j == checkOK.length)
  {
    allValid = false;
    break;
  }
}
if (!allValid)
{
  alert("Please enter only digit, whitespace and \"+-()\" characters in the
  \"Fax Number\" field.");
  theForm.Fax.focus();
  return (false);
}

if (theForm.Email.value == "")
{
  alert("Please enter a value for the \"E-Mail Address\" field.");
  theForm.Email.focus();
  return (false);
}

if (theForm.Email.value.length < 3)
{
  alert("Please enter at least 3 characters in the \"E-Mail Address\" field
  .");
  theForm.Email.focus();
  return (false);
}

var checkOK = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzƒŠŒŽš
œžŸÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþ01234567
89-.@-_,";
var checkStr = theForm.Email.value;
var allValid = true;
for (i = 0;  i < checkStr.length;  i++)
{
  ch = checkStr.charAt(i);
  for (j = 0;  j < checkOK.length;  j++)
    if (ch == checkOK.charAt(j))
      break;
  if (j == checkOK.length)
  {
    allValid = false;
    break;
  }
}
if (!allValid)
{
  alert("Please enter only letter, digit and \".@-_,\" characters in the \"E-
  Mail Address\" field.");
  theForm.Email.focus();
  return (false);
}
return (true);
```

```
}
//--></script><!--webbot BOT="GeneratedScript" endspan --><form method="POST"
action="/orders/profile;profile_submit" onsubmit="return FrontPage_Form1_Vali
dator(this)" name="FrontPage_Form1">
      <div align="center"><center><table border="0" cellpadding="
      3" width="100%" cellspacing="1">
        <tr>
          <td width="100%" valign="middle" align="center" nowrap colspan="
          4"><font color="#FF0000" face="Arial"><strong><big><big>~~mess1$~~
          </big></big> <big>- ~~mess2$~~</big></strong></font></td>
        </tr>
        <tr>
          <td width="15%" valign="middle" align="right" nowrap></td>
          <td width="85%" valign="middle" align="left" colspan="3" nowrap>
          </td>
        </tr>
        <tr>
          <td width="15%" valign="middle" align="right" nowrap><strong><font
          face="Arial">User ID:</font></strong></td>
          <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
          -webbot bot="Validation" S-Display-Name="User ID" S-Data-Type=
          "String" B-Allow-Letters="TRUE" B-Allow-Digits="TRUE" I-Minimum-
          Length="4" I-Maximum-Length="10" --><input type="text" name=
          "UserID" size="20" tabindex="1" maxlength="10" value="~~userid$~~">
          <font face="Tahoma"><imgsrc="../images/blueball.gif" alt="blue
          ball.gif (262 bytes)" WIDTH="15" HEIGHT="15"></font></td>
        </tr>
        <tr>
          <td width="15%" valign="middle" align="right" nowrap><strong><font
          face="Arial">Password:</font></strong></td>
          <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
          -webbot bot="Validation" S-Display-Name="Password" S-Data-Type=
          "String" B-Allow-Letters="TRUE" B-Allow-Digits="TRUE" I-Minimum-
          Length="4" I-Maximum-Length="10" --><input type="password" name=
          "passwd1" size="20" tabindex="1" value="~~passwd1$~~" maxlength=
          "10"><font face="Tahoma"> <img src="../images/blueball.gif" alt=
          "blueball.gif (262 bytes)" WIDTH="15" HEIGHT="15"></font></td>
        </tr>
        <tr>
          <td width="15%" valign="middle" align="right" nowrap><strong><font
          face="Arial">ConfirmPassword:</font></strong></td>
          <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
          -webbot bot="Validation" S-Display-Name="Confirmation Password" S-
          Data-Type="String" B-Allow-Letters="TRUE" B-Allow-Digits="TRUE" I-
          Minimum-Length="4" I-Maximum-Length="10" --><input type="password"
          name="passwd2" size="20" tabindex="1" value="~~passwd2$~~"
          maxlength="10"><font face="Tahoma"> <img src="../images/blueball.
          gif" alt="blueball.gif (262 bytes)" WIDTH="15" HEIGHT="15">
          </font></td>
        </tr>
        <tr>
          <td width="15%" valign="middle" align="right" nowrap></td>
          <td width="85%" valign="middle" align="left" colspan="3" nowrap>
          </td>
        </tr>
        <tr>
          <td width="15%" valign="middle" align="right" nowrap><strong><font
          face="Arial">FirstName:</font></strong></td>
```

```
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="First Name" B-Value-
    Required="TRUE" I-Minimum-Length="1" --><input type="text" name=
    "FirstName" size="20" tabindex="2" value="~~firstname$~~"><font
    face="Tahoma"> <img src="../images/blueball.gif" alt="blueball.gif
    (262 bytes)" WIDTH="15" HEIGHT="15"></font></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">Last Name:</font></strong></td
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="Last Name" B-Value-
    Required="TRUE" I-Minimum-Length="1" --><input type="text" name=
    "LastName" size="20" tabindex="3" value="~~lastname$~~"><font
    face="Tahoma"> <img src="../images/blueball.gif" alt="blueball.gif
    (262 bytes)" WIDTH="15" HEIGHT="15"></font></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap height="20">
    </td>
    <td width="85%" valign="middle" align="left" colspan="3" height=
    "20" nowrap></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">CompanyName:</font></strong></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="Company Name" B-Value-
    Required="TRUE" I-Minimum-Length="1" --><input type="text" name=
    "CompName" size="50" tabindex="4" value="~~compname$~~"><font face=
    "Tahoma"> <img src="../images/blueball.gif" alt="blueball.gif (262
    bytes)" WIDTH="15" HEIGHT="15"></font></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">Address:</font></strong></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="Address" B-Value-Required=
    "TRUE" I-Minimum-Length="1" --><input type="text" name="Addr1"
    size="50" tabindex="5" value="~~addr1$~~"><font face="Tahoma">
    <img src="../images/blueball.gif" alt="blueball.gif (262 bytes)"
    WIDTH="15" HEIGHT="15"></font></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap>
    <input type="text" name="Addr2" size="50" tabindex="6" value="~~add
    r2$~~"></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap>
    <input type="text" name="Addr3" size="50" tabindex="7" value=
    "~~addr3$~~"></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">City:</font></strong></td>
    <td width="34%" valign="middle" align="left" nowrap><!--webbot bot=
    "Validation" S-Display-Name="City" B-Value-Required="TRUE" I-
    Minimum-Length="1" --><input type="text" name="City" size="20"
    tabindex="8" value="~~city$~~"><font face="Tahoma"> <img src="../
    images/blueball.gif" alt="blueball.gif (262 bytes)" WIDTH="15"
    HEIGHT="15"></font></td>
    <td width="17%" valign="middle" align="right" nowrap><strong><font
    face="Arial">State/Prov:</font></strong></td>
    <td width="34%" valign="middle" align="left" nowrap><!--webbot
    bot="Validation" I-Minimum-Length="1" --><input type="text" name=
    "Province" size="10" tabindex="9" value="~~province$~~"></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">Country:</font></strong></td>
    <td width="34%" valign="middle" align="left" nowrap><!--webbot bot=
    "Validation" S-Display-Name="Country" B-Value-Required="TRUE" I-
    Minimum-Length="1" --><input type="text" name="Country" size="20"
    tabindex="10" value="~~country$~~"><font face="Tahoma"> <img src=
    "../images/blueball.gif" alt="blueball.gif (262 bytes)" WIDTH="15"
    HEIGHT="15"></font></td>
    <td width="17%" valign="middle" align="right" nowrap><strong><font
    face="Arial">Zip/Postal:</font></strong></td>
    <td width="34%" valign="middle" align="left" nowrap><input type=
    "text" name="PostalCode" size="10" tabindex="11" value="~~postal
    code$~~"></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap>
    </td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">Tel:</font></strong></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="Telephone Number" S-Data-
    Type="String" B-Allow-Digits="TRUE" B-Allow-WhiteSpace="TRUE"
    S-Allow-Other-Chars="+-()" B-Value-Required="TRUE" I-Minimum-
    Length="7" --><input type="text" name="Telephone" size="30"
    tabindex="12" value="~~telephone$~~"><font face="Tahoma"> <img src=
    "../images/blueball.gif" alt="blueball.gif (262 bytes)" WIDTH="15"
    HEIGHT="15"></font></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">Fax:</font></strong></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="Fax Number" S-Data-Type=
    "String" B-Allow-Digits="TRUE" B-Allow-WhiteSpace="TRUE" S-Allow-
    Other-Chars="+-()" --><input type="text" name="Fax" size="30" tab
    index="13" value="~~fax$~~"></td>
  </tr>
  <tr>
    <td width="15%" valign="middle" align="right" nowrap><strong><font
    face="Arial">E-Mail:</font></strong></td>
    <td width="85%" valign="middle" align="left" colspan="3" nowrap><!-
    -webbot bot="Validation" S-Display-Name="E-Mail Address" S-Data-
```

```
            Type="String" B-Allow-Letters="TRUE" B-Allow-Digits="TRUE" S-Allow-
            Other-Chars=".@-_," B-Value-Required="TRUE" I-Minimum-Length="3"
            --><input type="text" name="Email" size="40" tabindex="14" value=
            "~~email$~~"><font face="Tahoma"> <img src="../images/blueball.gif"
            alt="blueball.gif (262 bytes)" WIDTH="15" HEIGHT="15"></font></td>
          </tr>
          <tr>
            <td width="100%" colspan="4" valign="middle" align="center" nowrap>
            </td>
          </tr>
          <tr>
            <td width="50%" valign="middle" align="right"><strong><font face=
            "Arial" size="3"><input type="submit" value="         OK        "
            name="ok" style="font-family: sans-serif, serif; font-size: 14pt;
            font-weight: bold" tabindex="15"></font></strong></td>
            <td width="50%" valign="middle" align="left" colspan="3"><strong>
            <font face="Arial" size="3"><input type="reset" value="     Reset
            " name="Reset" style="font-family: sans-serif, serif; font-size:
            14pt; font-weight: bold" tabindex="16"></font></strong></td>
          </tr>
        </table>
        </center></div>
      </form>
      </td>
  </tr>
  <tr>
    <td height="25"></td>
  </tr>
  <tr>
    <td height="50" bgcolor="#000080"><p align="center"><small><font color=
    "#80FFFF" face="Tahoma">This information will be kept confidential.
    </font></small></td>
  </tr>
</table>
</div>
</body>
</html>
```

Orders Web Page 3: ordstat.htm

DireXions Example **orders\ordstat.htm**

Description:

This is a listing of the HTML for the **`ordstat.htm`** web page:

Example

```
<html>

<head>
<title>Order Status Page</title>
</head>


<body background="../images/whttxtr1.jpg">
<div align="left">

<table border="0" cellpadding="0" width="737" cellspacing="0">
  <tr>
    <td><img src="../images/pvxbird2.jpg" alt="pvxbird2.jpg (13155 bytes)"
    WIDTH="737" HEIGHT="90"></td>
  </tr>
  <tr>
    <td height="50" bgcolor="#000080"><p align="center"><strong><big><big>
    <big><font color="#80FFFF" face="Tahoma">On-Line Order System</font>
    </big></big></big></strong></td>
   </tr>
</table>
</div>

<form method="POST" action="ordstat">
  <p><input type="submit" value="     Exit     " name="Exit"><input type=
  "submit" value="Change Profile" name="getprofile"></p>
</form>

<p><font   face="Arial"><font   color="#0000FF"><strong><big>Welcome:</big></
strong></font>~~firstname$~~ ~~lastname$~~</font></p>

<p><font  face="Arial"><strong><big>Your  Current  ShipTo  address  is:</big></
strong><br>
&nbsp;&nbsp;&nbsp; ~~compname$~~<br>
&nbsp;&nbsp;&nbsp; ~~addr1$~~<br>
&nbsp;&nbsp;&nbsp; ~~addr2$~~<br>
&nbsp;&nbsp;&nbsp; ~~addr3$~~<br>
&nbsp;&nbsp;&nbsp;     ~~city$~~,      ~~province$~~,      ~~country$~~&nbsp;
~~postalcode$~~</font></p>

<p><font face="Tahoma">Number of Orders on File: ~~numorders~~</font></p>
<div align="left">

<table border="1" cellpadding="3" width="737">
  <tr>
    <td width="16%" align="center" bgcolor="#C0C0C0"><strong><font face=
    "Tahoma" color="#000000">Order Number</font></strong></td>
    <td width="16%" align="center" bgcolor="#C0C0C0"><strong><font face=
    "Tahoma" color="#000000">Product #</font></strong></td>
    <td width="17%" align="center" bgcolor="#C0C0C0"><strong><font face=
    "Tahoma" color="#000000">Qty</font></strong></td>
    <td width="17%" align="center" bgcolor="#C0C0C0"><strong><font face=
    "Tahoma" color="#000000">Cost</font></strong></td>
    <td width="17%" align="center" bgcolor="#C0C0C0"><strong><font face=
    "Tahoma" color="#000000">Status</font></strong></td>
  </tr>
  <tr>
    <td width="16%" align="center">&nbsp;</td>
    <td width="16%" align="center">&nbsp;</td>
    <td width="17%" align="center">&nbsp;</td>
    <td width="17%" align="center">&nbsp;</td>
    <td width="17%" align="center">&nbsp;</td>
  </tr>
</table>
</div>
</body>
</html>
```