Part One

Setting Up the Shopping Cart

CHAPTER ONE

GETTING, INSTALLING, AND RUNNING THE APPLICATION

Downloading the Script

Although the script is distributed with the CD-ROM accompanying this book, it is recommended that you point your Web browser to "Selena Sol's Script Archive" to get the latest version of this script. The Script Archive is located at the following URL: http://www.eff.org/~erict/Scripts/.

From the Scripts Archive front page, follow the hyperlinks for "Web Store" to the detailed page dedicated to this script. There you will find several working demonstrations representing different configuration options as well as the scripts for review and download. Feel free to click on the hyperlink **Download the scripts as a single TAR file** in order to transfer the archived scripts to your local machine. For Windows NT Server users, there will be a zipped distribution of the scripts that are configured to run on a variety of Windows NT Web Servers.

Alternatively, you may grab the application off the accompanying CD ROM by following the instructions in Appendix A.

Unpacking the Application

For UNIX Web Servers, the Web Store application is distributed in the form of a single TAR file. **TAR** is a UNIX command that allows you to create a single archive file containing many files. Such archiving allows you to maintain directory relationships and facilitates transferring complex programs with many separate but integrated parts which must have their relationships preserved. TAR has a motley of options that allow you to do archiving and dearchiving in many ways. However, for the purpose of unpacking this application from its TAR file, the commands will be fairly simple.

The Windows NT version of the Web Store will be distributed as a ZIP file. Setting up the Web Store on Windows NT or Windows 95 Web servers will be discussed in a separate section of this chapter.

Once you have downloaded the TAR file, transfer it to an executable directory on your Web server and unpack it using the TAR program. On UNIX systems, you may type the following at the command (for this command to work, you must be in the same directory as the TAR file itself resides):

```
tar xvfp web_store.tar
```

If you are using a UNIX server, do not unpack the scripts on a Windows or Mac OS machine and ftp them to the server. Instead, transfer the TAR file to the UNIX machine using binary mode and unpack it there. Otherwise, you may introduce bad characters into the files because Windows and MAC machines typically handle line breaks differently than UNIX machine, inserting **\r\n** for new lines instead of just **\n**.

TAR will go through the archive file and separate each of the individual directories and files, expanding them into their appropriate places beneath the current directory. The **xvfp** letters in the TAR command above are parameters that tell the program to extract the files and directories out of the **.tar** file. Specifically, **x** tells TAR to extract the files. **v** tells TAR to output information about the

status of its extraction while it is performing the work, **f** informs TAR to use the **.tar** filename as the source of the files to be extracted, and **p** notes that the original permissions should be maintained. The reason the **f** parameter has to be used is that TAR, by default, archives files and directories to a tape drive. TAR is actually short for "[T]ape [AR]chive". Figure 1.1 shows the output of the above command.

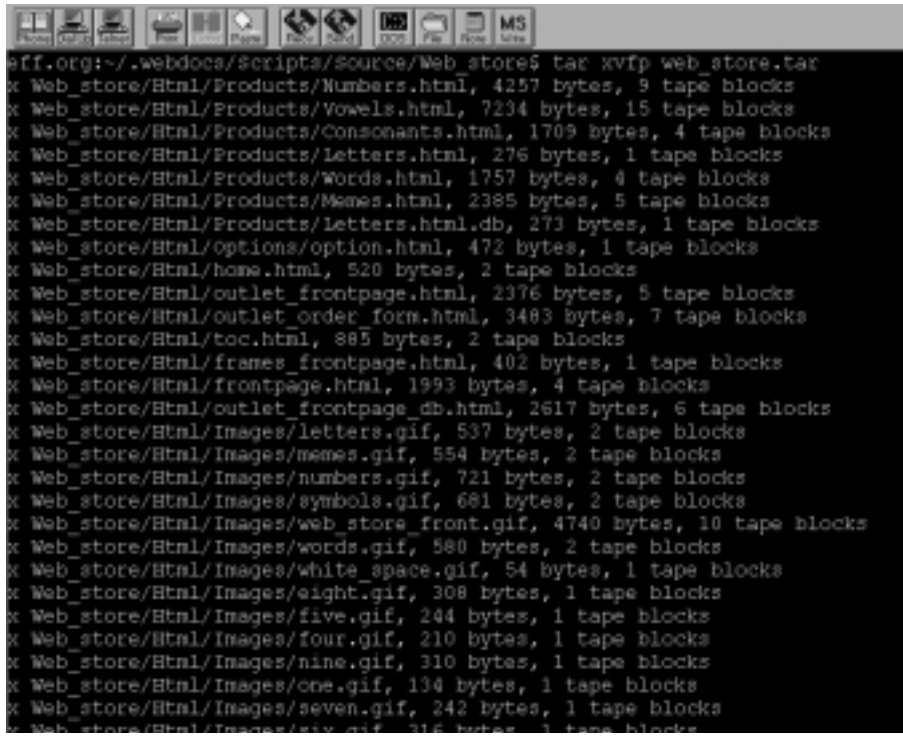

Figure 1.1 Unpacking the Web store using TAR.

If you are using a non-UNIX operating system and Web server, you may download a TAR/UNTAR program by pointing your Web browser to http://www.shareware.com. We suggest using "untar" as your keyword when you search their inventory. In addition, many popular Windows Zip utility programs such as WinZip have inherent functionality that allows them to unpack tar files.

Setting Permissions

Unpacking the files is only one part of the equation of installing the Web Store application and getting it to actually execute. Frequently, the Web server needs to be given special permission to execute your scripts and have the scripts perform their job with the appropriate "rights."

The cardinal rule for setting up Web server software is that the server should be given only minimal capabilities. This definitely rules out the Web server running as the ROOT user (Super-user on UNIX). More often than not, it means that the Web server is run as a user that has no rights to do anything significant—a user "nobody." By default, "nobody" usually does not have permission to read any files in directories that you create. However, when you download scripts, you need to create an environment so that the scripts can be read and executed by the Web server software. In other words, "nobody" has to be able to access the files.

In UNIX, the command for performing this task is **chmod**. The directory structure of the Web store root directory is a good example of how you should use the **chmod** command.

For this example, we will assume that you are sharing a server on an Internet service provider, which is typically the most restrictive situation in terms of your security options, and that all your CGI scripts are located in the default directory called Web_store. The example also assumes that we are in the directory above the one where the CGI scripts actually reside when we execute the chmod command. Finally, this example assumes that the Web server is running as the user "nobody" and does not belong to your user i.d.'s security group. Note that if the Web server is indeed part of your group, you will be able to restrict permissions on the files so that the world cannot do anything with the files, but anyone in your security group can do the minimal operations needed for the script to operate.

There are four different sets of permissions that need to be granted in order for the user "nobody" to have the permissions needed to execute this CGI script. First, the Web_store directory itself must be both readable and executable by the world. The directory is readable because we need to read the entries in the directory, and it is executable so that the Web server can go into the directory and open the files in it. This is done using the following command:

```
chmod 755 Web_store
```

The 755 number tells chmod to make the Web_store directory readable, writable, and executable by the owner of the directory (you) while making it only readable and executable by the world and the group you are in.

How did we come up with the number? Files in UNIX have three types of permissions: User (the owner of the file), Group (the security group you are in), and OTHER (for the world to see). Each digit in the number above corresponds to one of these categories. The first digit is User, the second digit is group, and the final digit is other. Thus in the example above, 7 = USER, 5 = GROUP, and OTHER = 5.

The actual value of the digit determines the permissions granted to that area. Permissions consist of three numbers: 4 for Read, 2 for Write, and 1 for Execute access. By adding these numbers together, you form the permissions that make up one digit. For example, 4 + 2 + 1 = 7, which grants read, write, and execute permissions; 4 + 1 = 5, which grants only read and execute permissions. Thus, 755 grants 7 (read, write, execute) to the owner of the file, and 5 (read and execute) to the group the file is in and to the world. Table 1.1 can be used as a quick reference

Table 1.1 Common Permissions Settings for CGI Applications

Permission			Command
User	**Group**	**World**	
rwx	rwx	rwx	chmod 777 filename
rwx	rwx	r-x	chmod 775 filename
rwx	r-x	r-x	chmod 755 filename
rw-	rw-	rw-	chmod 666 filename
rw-	rw-	r- -	chmod 664 filename
rw-	r- -	r- -	chmod 644 filename

r = Readable; w = writable; x = executable; – = no permission

The files within the Web_store directory need to be readable and executable since the Web server will be running the CGI scripts in it. To set those permissions, we use the following command:

```
chmod 755 Web_store/*.*
```

This command operates in the same way as the one above, except that it changes all the files (*.* pattern matches everything) inside the Web_store directory to be readable and executable by everyone. These scripts themselves should not be writable!

Nor should the Web_store directory be writable, since that would allow other users on the system place scripts there. The Admin_files directory is another issue. However, since it contains log files which the Web server must write to, the Admin_files directory must be made writable so that the script can write to the directory. In addition, the files generated in that directory should also be readable in case the Web Store script needs to access them. To change the directory so that it is writable as well as having read and execute access, use the following command:

```
chmod 777 Web_store/Admin_files
```

This will change the Admin_files directory under the Web_store directory to be readable, writable, and executable by the world.


WARNING

You may be tempted to simply use chmod 777 on all the files and directories since that assures the Web server can do anything with the files. However, it is strongly advised that you do not leave the files in this state. It is considered a major security risk to leave your scripts open to changes by the Web server instead of being read-only. Anyone on the server could use another rogue CGI script to write over your scripts and make them do something completely different. There is still a risk involved in making the Admin_files and User_carts directory writable, but at least if someone is going to be messing with your area, they will only destroy a bit of data and not your main programs. It is OK to set the scripts to 777 if you are troubleshooting a problem and want to rule out permissions entirely, but do not leave the scripts like this.

On another security note, if you are really concerned with the security of your data, do not use a shared server where other people can write CGI scripts using the same Web server configuration. It is much better to use your own server soft ware or purchase space on a virtual server that may be shared, but is set up in such a way that each user's scripts are shielded from each other.

Not setting your permissions correctly is the chief cause installation failure. Take the time to get this right. The Web store setup checking script described in Chapter 2 can aid you in this endeavor.

Finally, most other files should be simply readable to the Web server. For example, consider the files in the Library subdirectory. Since the script never needs write access to these files, it is best guard them by the strictest permissions possible, making them writable only by you. This will protect those files against malicious hackers or accidental modifications by others in your group (see Figure 1.2).

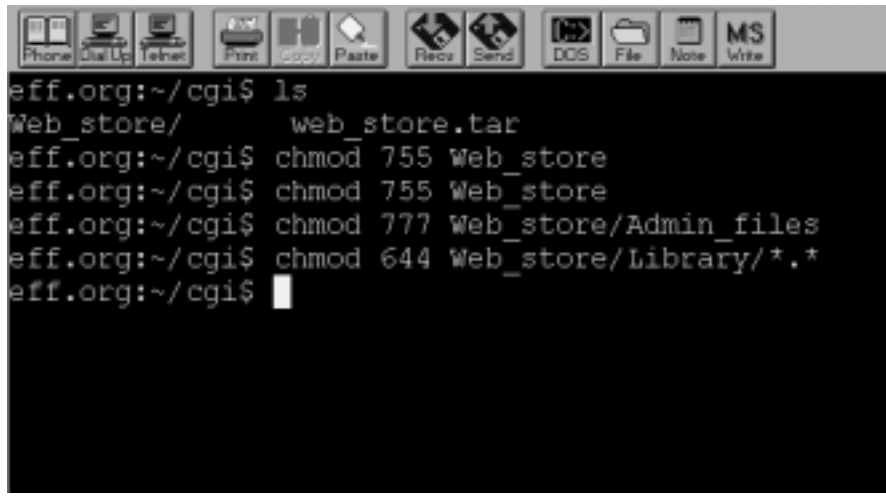

Figure 1.2 Setting permissions.

To set permissions for these files, use the following syntax:

```
chmod 644 Web_store/Library/*
```

The actual permissions required for the subdirectories and files used by this application are listed in the next section.

Understanding Web Store Files and Directories

The TAR file expands into a root directory called *Web_store* that contains several subdirectories and several files. Figure 1.3 depicts the directory structure as well as the permissions that must be applied to the files and sub-directories used by the application.

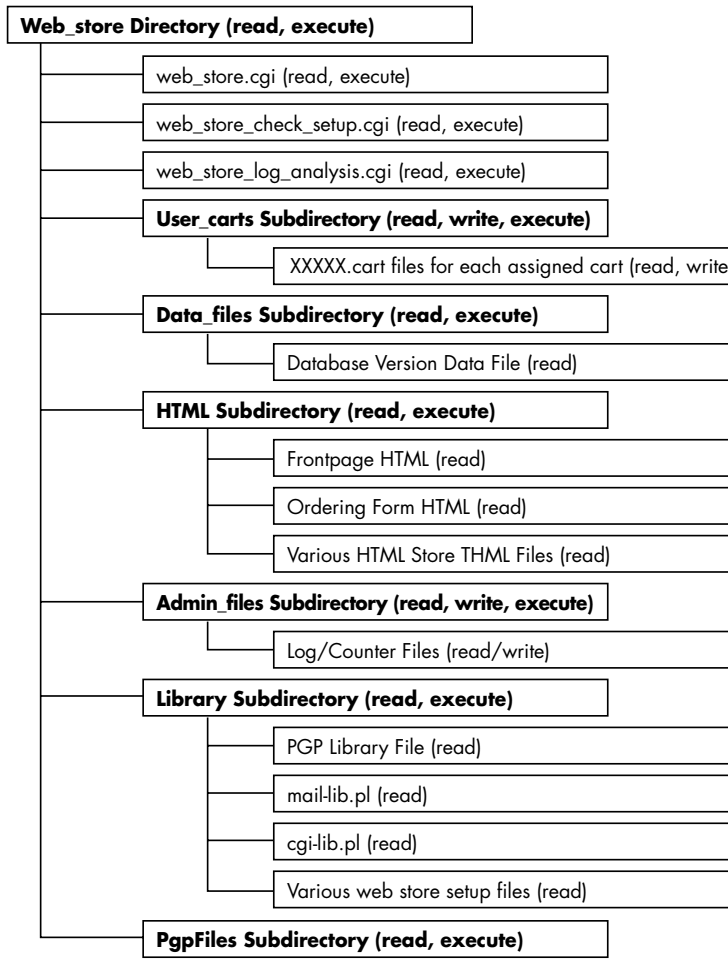

Figure 1.3 Web store directory structure with permission settings.

Web_store is the application's root directory. The Web server must have read and execute privileges for this directory.

The **Admin_files** subdirectory is used to hold various files that the Web server must have permission to read and write to. Within the directory are several logs used to keep track of store usage as well as the counter file. Each of these files must be readable by and writable to the Web server. The subdirectory itself must be readable, writable, and executable by the Web server.

access.log contains information about customers who access the store frontpage. It will contain the values of any environment variables known to your server as well as the date of access in pipe-delimited (|) database rows separated by the newline character.

counter.file is used to generate unique i.d. numbers for every item in customers' carts. Each item that the customer orders must be uniquely identifiable if the script is to modify or delete it.

error.log contains information about errors that have occurred during use of the script. Each pipe-delimited row of the log file contains information about what error has occurred, the line number of the error and any environment variables known by the Web server. Each row is separated by a newline character.

order.log contains a log of all the orders that have been processed by web_store.cgi. Each individual order is separated by a line of dashes (- -).

The **Data_files** subdirectory contains the data file used to generate the product pages in a database-based store or to check against in an HTML-based store with database validation. The directory itself must be readable and executable by the Web server. The data file within it must be readable:

data.file is the file that contains all the store products. Essentially, this is a simple, flat-file, ASCII, pipe-delimited database. It is crucial that you define the structure of the database in the setup file as each column must be defined in several index variables and arrays. The data file and the index variables will be discussed in greater detail in Chapter 2 and Chapter 4.

The **Html** subdirectory holds various HTML documents used by both the HTML-based and database-based versions of web_store.cgi to provide navigation-like functions such as the front page, order form, and list of product pages.

However, in the case of the HTML-based store, it is responsible for containing all of the product pages as well as the navigational pages. The directory and its subdirectories must be readable and executable by the Web server, and the files within the directories must be readable.

Images is a subdirectory containing images used by the default example store in the Web Store distribution.


NOTE

If you are running this script on a server that does not allow you to reference images within the CGI directory, you may have to move this directory elsewhere in order to actually see the images. If you are getting broken images but all else works correctly, this is likely the cause. It is important to note that if you do move this directory, all links hardcoded in the example HTML pages as well as those in data.file must be changed to reflect the new location.

Options is a subdirectory containing any option files that you create. Option files are used in the database-based version of the store when you want many items to have the same option HTML code, but you do not want to write that code in every single database row. Option files will be discussed in greater detail in Chapter 4. This directory must be readable and executable and every option file must be readable.

Products This subdirectory contains predesigned HTML pages that the store may be asked to display during shopping. There are two types of pages stored here. First, product pages are stored here. *Product pages* are simply HTML forms written with a specific format (using special tags discussed in Chapter 3) that allow customers to buy specific store items (see Figure 1.4). These pages are used for HTML-based stores which cannot generate product pages dynamically from a data file.

Secondly, this subdirectory contains *list of product* pages, which are simply HTML pages that contain hyperlinks to product pages (see Figure 1.5). These lists of product pages are used for both the HTML and database versions of the store. The coding of both product pages and list of product pages will be discussed later.

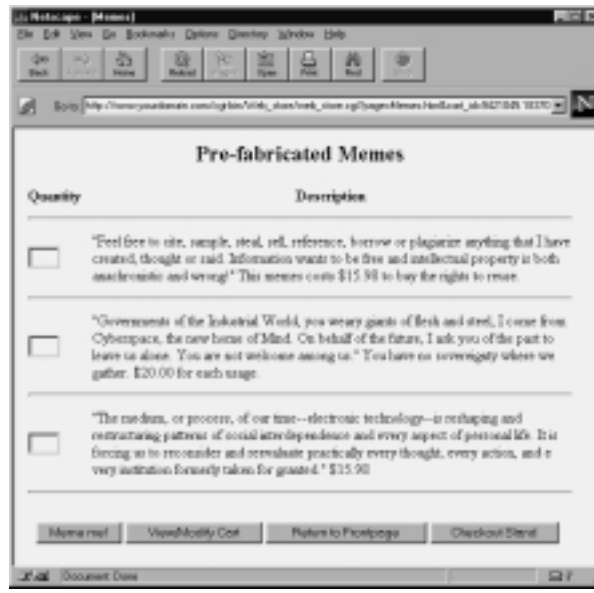

Figure 1.4 Memes.html—An example product page.

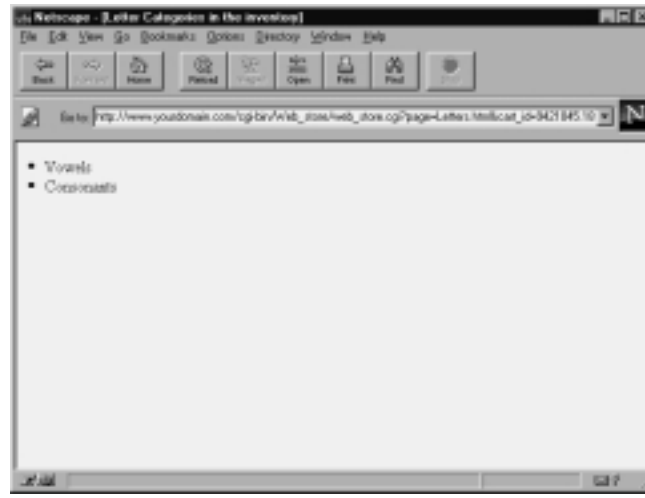

Figure 1.5 Letters.html—An example list of products page.

Letters.html (HTML-based store-specific) and **Letters.html.db** (Database-based store-specific) are examples of a list of product pages, and **Memes.html** is an example of a product page. Other product pages include **Vowels.html**, **Consonants.html**, **Words.html,** and **Numbers.html**.

frames_frontpage.html, home.html, toc.html and **frontpage.html** are all used in the Frames example of the store. **frames_frontpage.html** defines three frames that point to the other three. These will be discussed more thoroughly in Chapter 6.

outlet_frontpage.html, outlet_frontpage_query.html and **outlet_frontpage_db.html** are the frontpage examples for the Html-based store, Query-driven store, and the database-based stores, respectively.

outlet_order_form.html and outlet_order_form_with_shipping.html are two examples of how you might create your own order form. **outlet_order_form.html** is a basic example. **outlet_order_form_with_shipping.html** contains the HTML necessary to calculate sample shipping logic.

The **Library** subdirectory is used to hold the associated library files used by this application. The directory itself must be readable and executable by the Web server. The files within it must be readable.

cgi-lib.pl is used to read and parse form data as well as providing a convenient way to exit with explanation in case of a problem in opening a file.

mail-lib.pl is used to send unencrypted email to the store administrator.

pgp-lib.pl is used to send encrypted email to the store administrator.

web_store.setup.* are the six sample setup files distributed by default. These setup files exemplify several methods for defining the interface to the store. The differences among these setup files, and the actual meaning of setup variables will be discussed in Chapter 2, which goes into the details of creating the generic **web_store.setup** .

web_store.setup.db is the basic database-based store with a non–table-based product display. (In use, the setup file creates product displays that mirror the HTML-based store). **web_store.setup.db.table** displays products tabularly; Less visually effective but very efficient, it is used in the distribution to demonstrate query-based searching. Chapter 4 gives the changes needed to make the **web_store.setup.db** operate off of a database. Chapter 5 gives the changes needed to transform the regular database store into a query-driven store.

web_store.setup.frames gives an example of a frames-based store. By default, it uses the nontabular database-based logic for display of products. However, it could just as easily use the tabular-database or HTML logic. Chapter 6 goes into the details of how the frames-based store works.

web_store.setup.frames.javascript takes frames one step further by integrating a JavaScript routine to calculate order totals before customers add items to their carts. **web_store.setup.frames.vbscript** does the same thing, but in VBScript. Chapter 7 illustrates how JavaScript and VBScript were integrated with the Web store.

web_store.setup.html shows an example of using an HTML-based Web store interface. Chapter 3 outlines how the HTML Web store is configured.

web_store_db_lib.pl is a library of subroutines used to search the database.

web_store_html_lib.pl is a library of subroutines used to generate most of the customizable HTML. If you want to change the look and feel of your store, you will probably modify the code in this library. The specifics of the library will be discussed later.

web_store_html_search.pl defines the search routines for the HTML-based store that cannot search a database for keywords but must read through product pages individually searching for hits.

web_store_order_lib.pl defines the routines necessary to process orders.

Pgpfiles is the subdirectory used to store files necessary for processing orders using PGP encryption. The directory itself must be readable and executable by the Web server and files within it must be readable.

PGP is a third-party encryption tool written by Phil Zimmerman for the purpose of encrypting and decrypting files. You have to know how to use PGP in order to use this feature of the cart. By default, we turn off PGP encryption. Basically, you need to obtain PGP, generate a public and private key for your Web Store and then store the public key configuration files inside this directory. The details of this process are discussed in Chapter 9.

The **User_carts** subdirectory is used to store the actual shopping carts used by clients browsing your store. Initially, this subdirectory will be empty. As customers enter your store, the script will automatically fill the directory with new carts and prune the directory of old carts. The directory must be readable, writable, and executable by the Web server; the cart files created

within it must be readable and writable. The script will generally create and delete carts on its own.

web_store.cgi is the main script that generates the GUI for the online store. It must be readable and executable by the Web server.

web_store_check_setup.cgi is a script you can use to help with installation. This script will go through and check the permissions and path values you have defined in your setup file and report any problems it finds. The Web server must be able to read and execute this file.

web_store_log_analysis.cgi is a script used to browse your log files. It offers a simple keyword-search box and asks which log file you would like to review. It must be readable and executable by the Web server.

Installing on Windows 95/NT Web Servers

Installing CGI scripts on a Windows NT– or a Windows 95–based Web server is generally more of a challenge than installing them on a UNIX server. This is because, although UNIX servers have standardized the way they handle CGI scripts, 32-bit Windows servers have not. They all have slightly different behaviors when interpreting CGI. To top it off, there is a CGI standard specifically made for Windows and called *WinCGI*. In this section, we will focus on tips and techniques that will help you run the Web store on some of the more popular Windows NT Web servers.


NOTE

The various changes discussed below have already been implemented and are distributed on the CD-ROM in order to allow you to have an easier time of making your scripts work on Windows NT or Windows 95 Web servers. Appendix A covers the CD-ROM distribution in detail.

General Tasks for All NT Servers

The main concern for all NT-based Web servers is to make sure that anything that seems to be UNIX specific either has an equivalent feature on NT or can be substituted by another utility. This generally involves obtaining Perl for NT

and make sure that any UNIX specific features on the Web Store are translated into their Windows NT equivalents.

The first step is to obtain Perl for Windows NT/Windows 95. This can be done by doing a simple Web search on **perl** and **nt**. While there are several Perl ports for NT, we currently recommend using the port from Hip Communications. It is Perl 5–compatible and appears to be the port most compatible with UNIX-based Perl at the moment. Installing Perl is easy. Simply unzip it into a directory on your NT server and run the **install.bat** program that exists in the new Perl subdirectory that you have created.

Next, **mail-lib.pl** needs to be replaced with **smtpmail-lib.pl**. In addition, changes need to be done to the **smtpmail-lib.pl** file to make it recognize that it is running on NT. This is discussed in detail at the end of Chapter 8 in the section that discusses configuring **mail-lib.pl**. **Mail-lib.pl** is the only UNIX-specific library in the entire Web store. Similarly, you may need to modify **pgp-lib.pl**. The only reason this library may not work on NT is if the interface to the DOS version of PGP is different from the interface to the UNIX version of PGP. Every other library and program should work as-is on Windows NT.

Tasks Required to Make the Web Store Run on Netscape Commerce Server for NT

The only thing that must be done in addition to the tasks above for Netscape Commerce Server version 1.x (and Netscape Communications Server v1.x) is that the script must be encapsulated inside a Batch file. The v1.x Netscape servers were unable to recognize CGI scripts directly. Thus, you need to wrap the **web_store.cgi** script inside a batch file to make it run. Then, all references to **web_store.cgi** need to be changed to **web_store.bat** in the various HTML files as well as the setup file. In the setup file, there are two url variables that need to be changed to reflect the new batch file name: **$sc_main_script_url** and **$sc_order_script_url**.

The contents of the batch file is easy. The first line consists of **@echo off** so that the DOS-related information does not print out to the Web browser and the second line actually calls the script using Perl.

In order for this to work, Perl must be installed so that it is in the path of the machine you are running it on. If it is not in the path of the machine you are running it on, you will need to alter the batch file definition so that Perl is referenced with an absolute path to its location.

```
@echo off
perl web_store.cgi
```

Running scripts as encapsulated batch files can open up a can of security worms. To obtain a full appreciation of the problems that may arise, you should read the CGI programming security FAQ located at the following URL: http://www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html.

Tasks Required to Make the Web Store Run on Netscape Enterprise/FastTrack Server for NT

The Netscape Enterprise or FastTrack server has the capability of recognizing and running scripts with the extension of **.cgi** if you have previously associated the Perl executable with **.cgi** extensions in your file manager or Windows explorer. Thus, you do not need to encapsulate the scripts inside a batch file. The bad news is that at the time of this writing, the default behavior of the new Netscape servers is to run the scripts as if they are operating relative to where Perl is located.

This is bad because all the setup script variables are generally configured so that they are expected to be relative paths from the directory where **web_store.cgi** resides. However, your **perl.exe** executable is usually in a totally different subdirectory such as **c:\perl5** or **d:\perl5**. There are two solutions to this problem. First, you could set all the paths in the setup file to be absolute paths instead of relative to the current working directory. However, then, you also have to modify the **web_store.cgi** file so that when the setup file is "required," it is required with the absolute path as well.

An alternative, and probably easier, approach is to add a line to **web_store.cgi** script instead of altering all the setup file options. For this example, we will assume the the real path where **web_store.cgi** is located on

your server is **d:\netscape\server\cgi-bin\Web_store\.** Before the line that contains the words **&require_supporting_libraries**, insert the following:

```
chdir(d:\netscape\server\cgi-bin\Web_store);
```

Adding this line will make the Web server reference the web script in the correct subdirectory for the duration of the running of that script. Remember, you must use two backslashes for every backslash in a normal MS-DOS path because backslashes need to be escaped in Perl.

Tasks Required to Make the Web Store Run on the WebSite Server for NT

As of this writing, WebSite has a problem almost identical to that of the Netscape Enterprise/FastTrack servers. Whenever a script is run on the WebSite server, it runs as if it is relative to the current working directory of the WebSite server executable itself. You can fix this problem in the same way we fixed the problem with Netscape Enterprise/FastTrack server.


NOTE

In order for **web_store.cgi** to run on a WebSite server, you need to associate the **.cgi** extension with the **perl.exe** executable just as you do with Netscape Enterprise/FastTrack server.

Tasks Required to Make the Web Store Run on Internet Information Server (IIS) For NT

Microsoft Internet Information Server version 1.0 and 2.0 both have a problem similar to that of the Netscape Enterprise/FastTrack and WebSite Web servers. Of course, just as these two Web servers have their own twist on the problem, IIS has its own peculiarity. Whenever you set up a directory alias on IIS that points to a real subdirectory on the NT server, any scripts that run in subdirectories underneath the original alias will act as if their current working directory is actually the root where the alias started. Thus, if the alias Scripts is configured to look at **d:\inetsrv\scripts**, and you try to run **web_store.cgi**

inside a Web_store subdirectory under the d:directory, the script will act as if it is running inside d:cripts. There are two ways around this problem.

First, you can solve it the same way the WebSite and the Netscape Enterprise/FastTrack server problems were solved and use the **chdir** command inside the **web_store.cgi script**. However, IIS offers you an alternative way to solve the problem. You can simply make a new alias! That is, if you want the script to act as though it is running in its own subdirectory, then all you have to do is set up an alias that points directly to the subdirectory where **web_store.cgi** is located and then call **web_store.cgi** using the new alias.

In order for **.cgi** extension scripts to run at all on IIS v1.0 and v2.0, you need to set up an association with the **.cgi** extension and the **perl.exe** executable. However, IIS does not recognize the normal NT-wide associations set up in File manager or Windows explorer. You will need to set up these associations manually in the registry. The instructions for doing this are included with the on-line help for IIS. Basically, you need to run **regedt32.exe** and then open the **HKEY_LOCAL_MACHINESControlSetregistry** entry. Then, from the edit menu, select **Add Value** of type **REG_SZ**. Enter the filename extension **.cgi**. Then, in the string editor, type the full path to the **perl.exe** interpreter. Finally, close down the registry editor and restart the IIS server.

Finding Perl

On UNIX, all the CGI scripts written in Perl have a line in them that expects Perl to be located in a particular directory. In addition, some CGI applications may expect external programs such as sendmail to be located in a certain location on the server. For example, the **mail-lib.pl** that is distributed with the UNIX version of the Web store expects Sendmail to be located in the **/usr/lib** directory by default.

The Web store script also has a header line telling the Web server where Perl is located. Most of the time, the references to these locations will be correct since the majority of servers are set up in a standard way. However, you may run across a situation where the programs that the scripts use are not where the script was configured to think they are. Thus, one of the last steps in setting up the Web store to actually run is to figure out where these files are

located so that the scripts can be changed to reflect the new file locations on your local server.

The classic example of a reference to an absolute path on a CGI script is the first line of the Perl code:

```
#!/usr/local/bin/perl
```

This line instructs the server to execute the following script through the Perl interpreter and indicates where to find the Perl interpreter. The Perl interpreter is a program that reads your script and translates it into a form that your server can execute. In the example above, the server will know that it can find the Perl interpreter in the directory **/usr/local/bin**.

While many servers may contain Perl in **/usr/local/bin**, there may be others that have installed them in other areas, such as **/usr/bin** or **/opt/bin**. If the location of Perl is not in **/usr/local/bin**, the first bit of customizing you will have to do is to find out where your local Perl interpreter is and change this line to reference the correct location. There are several ways of finding files on your system and not all of them work on every server, so be prepared to experiment with the following techniques.

The first command to try is **which**. At the command prompt of your UNIX server, you would type **which perl** and receive back the following reply:

```
$ which perl
/bin/perl
```

In other words, Perl is located in **/bin** on this system. Thus, you will need to change the first line of your script to

```
#!/bin/perl
```

If that does not work, you might also try the **whereis** command. This command could give you the following output:

```
$ whereis perl
perl: /usr/bin/perl /usr/local/bin/perl /usr/local/bin/perl4.036
/usr/local/bin/perl5.002
```

This gives us a little more information than the **which** command. It gives us information about all the Perl interpreters contained in the system. In other words, there are several versions of Perl installed on this server including 4.036 and 5.002! Since there are several versions, you could choose whichever one you wish to reference in the CGI script.

If those two commands failed, the next step would be to try **whence**. **Whence** is a command which is more specific to the Korn Shell, so you should first change to the Korn Shell if you can. To do this, simply type **ksh** at the command line of your UNIX system. Now, you should be free to use **whence** and get output that looks similar to the following:

```
$ ksh
$ whence perl
/usr/local/bin/perl
```

If all else fails, you might try the **find** command, but that is really pulling out all the guns when a simple email to your systems administrator might suffice. The syntax for find would be

```
find / -name perl
```

Running the Script

This script should be executed by pointing your browser to **web_store.cgi**. For example, you might use a URL such as:

```
http://www.yourdomain.com/cgi-bin/Web_store/web_store.cgi
```

The log analyzer can be accessed with the following URL:

```
http://www.yourdomain.com/cgi-bin/Web_store/web_store_log_analysis.cgi
```

The installation helper script can be executed with the following URL:

```
http://www.yourdomain.com/cgi-bin/Web_store/web_store_check_setup.cgi
```

As a final word of advice, *never* point a customer directly to an HTML document under the Web store. In order for the customer carts to be created and remembered by the application, **web_store.cgi** must filter *all* HTML pages before they are sent to the customer. Thus, you will never point to anything but **web_store.cgi** for every link having anything to do with the store. The moment users link to anything but **web_store.cgi**, they will lose their carts.

Some companies that have customized earlier versions of this script created a buffer page between the store and any outside site. This buffer page would contain a warning that explained to the client that if they followed this link they would be losing their cart and that if they wished to return to shopping, they should copy down their cart_id number in the form of a URL back to the script. Thus, anytime a link appeared within the store to an outside location, the actual URL would point to the buffer page, which would contain a second level link plus the warning.

If you are experiencing a situation in which customers are either losing their cart contents or having their cart contents merged with the cart contents of other customers, it means that you are probably providing a link to something other than **web_store.cgi**, or you have added a link without the proper state variables contained in the URL string. If you must add links within product pages to other locations within your store, always follow the format exemplified below. This hyperlink reference is taken from the list of product page **Letters.html**:

```
<A HREF = "web_store.cgi?page=Vowels.html&&cart_id=">
Vowels</A>
```

Notice that the **page** and **cart_id** state variables must be added to the call to the Web store script so that filtering can occur. This concept is discussed in greater depth in Chapter 2 and Chapter 3.

Using the Web Store

Once you have begun running the Web store script, a discussion of how to navigate through the Web store is in order. Basically, there are three main

areas of the store to be concerned about: browsing the catalog of items and purchasing the ones you want, viewing and managing your cart contents, and finally, actually sending in the order.

Browsing the Online Catalog

When you first start viewing the Web Store, a frontpage is presented. This front page typically has a menu of general categories of items to view plus a minimum of a keyword search engine to allow you to find the specific items you want more quickly. Figure 1.6 contains a sample front page from the HTML Web store.

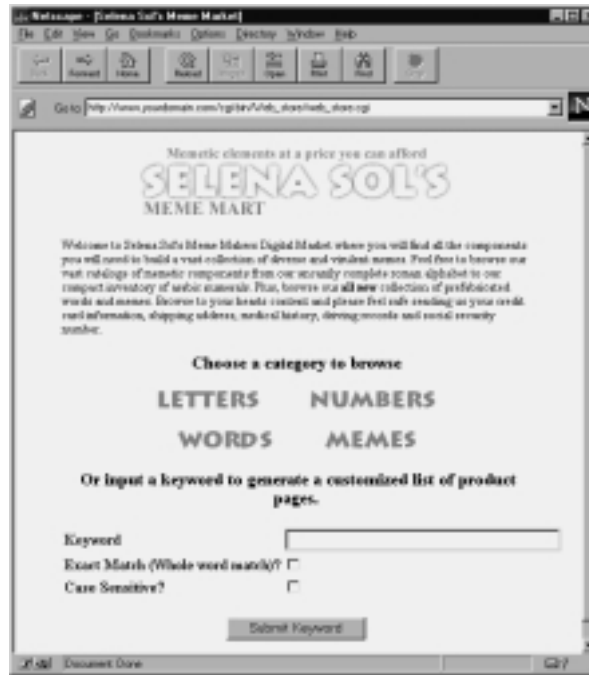

Figure 1.6 Frontpage from the HTML Web store.

The menu of general categories is typically called a list of products page because there is a list of product categories to choose from. If you click on a particular product hyperlink, then the Web Store will take you to an actual product page

that contains the products related to the selected category. Alternatively, you can do a keyword search and pull up a list of pages of products that correspond to your keyword search. Figure 1.7 shows a sample product page. Notice that there are text input fields next to each product. You can enter the quantity of the item that you want to purchase in these text boxes. Then, all you have to do is click on the **Add Items To My Cart** button to actually place them in your cart. The **View/Modify Cart** button lets you manage your cart contents, while the **Checkout Stand** button will bring you to the ordering form.

Figure 1.7 Sample product page.

Managing Your Cart Contents

If you click on the **View/Modify Cart** button, you will be brought to a screen that displays your cart contents and gives you options for changing quantities of items you have placed in the cart or removing them entirely. In addition, there are buttons that allow you to continue shopping and take you back where you were, or go to the checkout stand to order the items that you placed in the cart. Figure 1.8 shows an example of the cart page.

Figure 1.8 The View/Modify Cart page.

Sending in the Order

If you click on the **Checkout Stand** button, you will be brought to the order form. The order form displays your cart again, along with subtotals and any calculations that can be performed without such information about you as discounts based on high-volume purchasing. The order form has standard fields on it which ask for your name, mailing address, phone number, method of payment, and more. A sample order form is shown in Figure 1.9.

Once you submit the order form, the cart is again displayed to you along with any calculations that need to be performed. If have entered your state and shipping type (for example, UPS or FedEx), your sales tax and shipping costs can be calculated. The order is sent transparently to the store owner for processing and your trip through the store is finished. A sample Order Processed screen appears in Figure 1.10.

Figure 1.9 Standard order form.

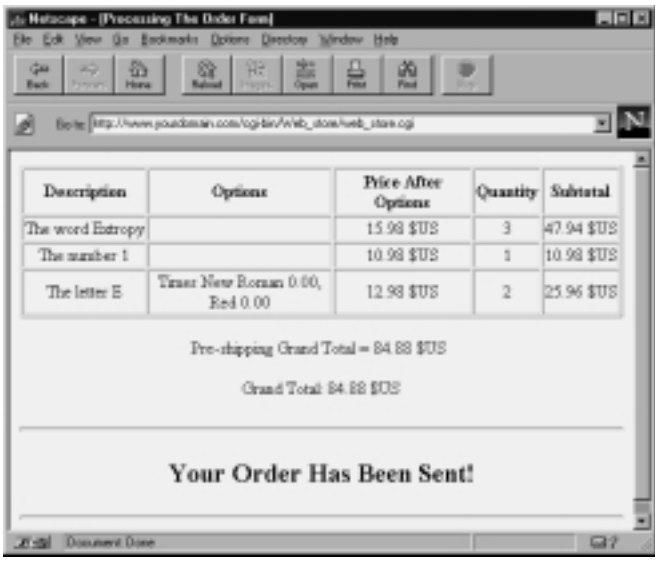

Figure 1.10 Standard order processed page.

Summary

Now that you have been able to successfully run and navigate through the Web Store, you are ready to start setting it up to cover your needs. Chapter 2 will cover the general Setup options. Subsequently, the remaining chapters in Part I will cover the specifics behind setting up different types of Web stores and configuring specific areas such as security and order processing.