INTRODUCTION

The Web Store

Shopping cart applications are some of the most popular, interesting to customize, and technically demanding CGI applications currently in use on the Web. Such applications allow companies to display their inventories online so that clients can quickly and easily browse through and order items of interest (Figure I.1).

Some of the more famous and groundbreaking examples include Virtual Vineyards (http://www.virtualvin.com/) and CD-Now (http://www.cdnow.com/). Both of these sites generate large sums of money from virtual clients through their user-friendly interfaces and sleek design. Because the warehouses are virtual and the sales staff are automated, the virtual store is a profitable option for many small and midsize businesses that want to gain instant access to a global market. Web stores can also provide larger, established companies with alternative outlets for their products, as well as a continued presence in emerging markets.

Figure I.1 Americal Web store customized by IDEA PRO Online Marketing at http://www.silicon.net/~tbond.

In addition, the customers receive the benefits of dynamic, customizable browsing tools. Rather than deal with mail-order catalogs, 1-800 numbers, or fax (or worse yet, snail mail–based) order forms, a customer need only navigate over to their favorite store on the World Wide Web, click to their section of interest, add some items to their virtual shopping cart, type in some shipping information, and submit the information over a secure channel of the Internet. As David Cook writes in *Launching a Business on the Web*, "The idea of a virtual shopping system is to make the job of picking products as painless as possible. In essence, we want the user to do no more than simply point to a picture or word to purchase an item."

And yet, however invisible and seamless the application is to the customer, an online store is a demanding application to program, customize, and install because it integrates many Web/CGI functions into one system. Consider the complexity for a programmer. A shopping cart application must be able to manage *every* function of shopping, not only from the perspective of the customer but from that of the vendor as well.

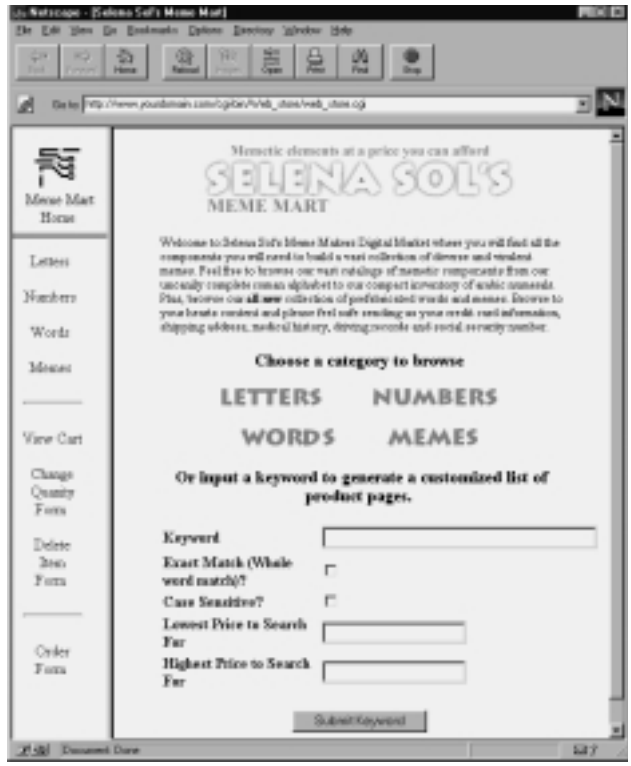

Figure I-2 Image of the Web store.

Store Management from the Vendor's Perspective

For one thing, the application must be able to manage and display the stores inventory. That is, the application must be able to keep track of every item in the store and to present those items in an efficient and sensible way. Think of this function as that of the store employee who must continually restock the shelves of the store with new items, change some prices here, remove some items there, introduce "specials," and generally make sure that aisles are clean, current, and organized for the convenience of customers.

In terms of the Web Store application, the program must be able to generate and display HTML "product pages" that can be browsed according to the needs of the client. There are many different ways these product pages can be displayed to the user. For example, they might be organized hierarchically, according to categories and subcategories of products or they might be generated dynamically based on the customer's search criteria. Further, depending on the desires of the vendor and the demands of the inventory itself, the inventory might be physically stored as a simple ASCII text flat file, in an SQL database, or as a series of predesigned HTML pages. The shopping cart system must be flexible and intelligent enough to produce a consistent look and feel while the specific display of items is changing based on the customer's desires.

Figure I.3 Product page from the Smokin' Joe Web store.

A further complication lies in the fact that every item must be uniquely identified and may have unique qualities that differentiate it from other items, even within the same category. One good example of this is the application of options. Many items, even within the same category may have different options available. For example, in an online music shop, certain albums might come in CD format only, in both tape and CD format, or even LP only. The Web store application must have a means for interpreting such unique qualities for every item and incorporating them into the logic of the process. In a store with 1000 items, there may be up to a 1000 special cases.

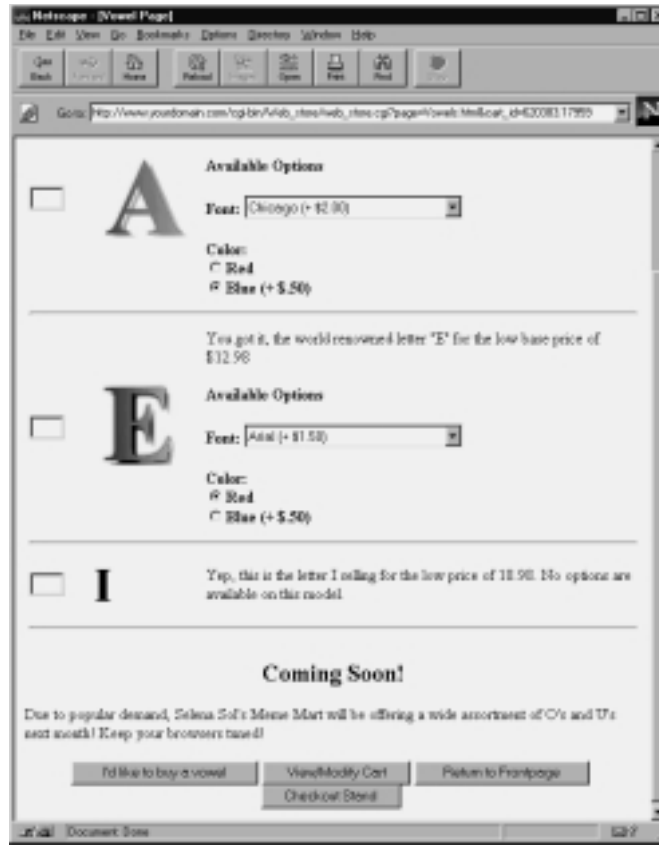

Figure I.4 Vowels.html page with options.

The Web store must also be able to keep track of many customers simultaneously, all of whom are at different points in the shopping process. Some of those clients might be just entering the store, some might be at the virtual cash register, and others might be browsing the shelves casually or with a specific product in mind. The Web store must be able to handle each of these situations so seamlessly that each customer feels as if he or she is the only customer in the store. Speed and consistency are paramount.

Keeping track of visitors also involves administrative functions. For example, just as someone must constantly keep track of used shopping carts in the parking lot of your local supermarket, so must the Web store application deal with its own old, used shopping carts. After all, it would not do to continually save all the carts used by all the customers. If all goes well and your products attract droves of browsing clients, you might generate hundreds of carts per day. Though each cart will occupy only a small amount of space on your server, when combined, the set of all carts could cause your server's hard disk to fill up quickly. Thus, the application must be responsible for pruning old carts at a regular interval.

Another administrative function is that of logging accesses and errors. Every new client will bring a set of valuable data that if gathered and interpreted wisely, could yield insights crucial to your continued success. How many of your clients are international? How many are repeat visitors? Which pages do clients most often request? Which items do they most often buy? What percent of your visitors use Netscape? When are your daily peak hours? These are the kinds of data that should be available in your access logs for analysis.

Similarly, you should have a complete error log. Have there been any attempted hacks? Are there errors in your configuration that slipped by you? Has some environment variable on your server changed that demands your attention, such as a change in permissions? The error log can be used to quickly diagnose problems with the application as it evolves. In some cases, a well-analyzed error log can save you from having to hire a programming consultant to do troubleshooting.

Finally, the Web store must be your cashier, totaling up shopping carts full of items and accepting payment from customers. In doing so, it must be able to handle your local taxes, shipping costs, discounts, specials, and any number of surprise price modifications on the fly for every customer.

More crucially, the Web store must do this flawlessly and securely. Not only must the connection between the browser and server be secure (an issue between you and your ISP), but orders must then be sent securely from the server to the person handling order processing. What good is a powerful and expensive SSL account if the customer's credit card and personal information are sent via unprotected email to the store owner? A secure store is only as secure as its weakest link. The Web store must be able to support SSL technology and provide a secure method of getting that information from the secure server to the store owner or administrator for processing.

Whatever the case, the application must be able to quickly provide a virtual store environment with seamless access to products. At this infant stage of the Web, there are already enough barriers between the clients and vendors: speed, ignorance, unfamiliarity, and more; the Web store technology must not add another barrier. It must be an interface that makes the user want to come back.

Shopping from the Customer's Perspective

The Web store is also responsible for handling the needs of the customer. It must follow each client through the store, even if there are dozens of clients shopping simultaneously, and act as his or her personal shopping servant. When the customer says "grab this item," the Web store must do so. It must also be able to get more than one item or to put all the items back on the virtual shelves if so instructed. The shopping cart itself is actually a database file that is built and modified on the fly based on the client's needs. The maintenance of the cart, therefore, requires a database management system capable of handling the addition, modification, and deletion of items in the cart.

However, the script must do more than just display and modify the shopping cart. It must also manipulate the database fields to perform price calculations, such as calculating a subtotal for each item purchased and generating a grand total from the subtotals in a cart. Furthermore, the Web store must do this for every customer independently using a technique to maintain the state of where each customer is in his or her shopping process.

Before we discuss how the script maintains this "state," a short discussion of client/server technology is in order. When you type a URL into the location window of your favorite browser and hit **Enter**, you are becoming a client to some server from which you are requesting a service. Typically, you will be asking the server to send you a document formatted with HyperText Markup Language (HTML). In the case of CGI, such as a Web store, you ask the server to execute a program and return the results of the processing. Unfortunately, that is all there is to the Web. There is no continuous discussion, dialog, or connection. There is only a simple one-time query and response each time a document or CGI script is requested.

One of the facts of life of client/server architecture is that each request sent from a client and processed by a server is considered a new and unique one. That is, there is only a series of unrelated queries and responses. The server maintains no link with its clients. Instead, the server simply, blindly, and automatically waits for a query, answers the query, and settles back down to wait for the next query to arrive. You may ask it for a second document by clicking another hyperlink or a **Submit** button, but the server will treat you as the unknown stranger that you are. It will not "remember" anything about you or your past interactions with it.

So, how do you create a complex relationship between a customer and a vendor using a Web script? For example, in order for each client to maintain a unique set of shopping cart items, the application must keep track of each client and each clients virtual cart. *Maintaining state*, as this is called, is difficult because HTTP, as we have mentioned, is a "connectionless" protocol. Every time a Web browser requests the attention of your server, it is considered a new request, unrelated to any other requests fulfilled by the server. Because each request is considered independently of others, the server has no way to keep track of what clients have added to their carts in the past.

It is possible for the browser software to be made responsible for remembering this information using technology like Netscape Cookies. However, using cookies is not the best solution for an online store that hopes to serve customers using a wide assortment of browsers that may not adhere to Netscape's proprietary cookie standard. Furthermore, even the customers who do use Netscape may have the cookies feature turned off for security reasons.

To solve this problem, the application becomes a self-referential script. It continuously calls itself for every request made by the client. However, when it calls itself, it passes to itself information about how to process the new request and information specific to the history of the relationship with the current client. In the case of the clients shopping cart items, the script will pass to itself the location of the clients cart on the server. This cart file will keep track of the items previously purchased. By providing the location in each call, the script continues to keep track of each client and cart.

The Web store does all this while still performing all the usual CGI functions, including reading and parsing incoming form data, checking that form data against bad input, emailing orders, and communicating with the browser via HTTP.

How This Book Is Structured

As you can see, the Web store application has a lot on its plate. The rest of this book will delve into the complex methodology used to solve these problems and provide a manageable framework for you to install and customize the routines for your own unique needs.

The book is divided into three sections.

Part One focuses on the process of downloading, customizing, and running the Web store application. Contained in 10 chapters, Part One goes through several common types of installations, discussing the many facets of customization from configuring your GUI to handling specific order logic ranging from discounts to shipping costs to zip code calculations. This part concludes with a discussion of methods of log analysis and secure shopping with which you can streamline and enhance the functionality and security of your store.

Part Two takes a more in depth look at the scripts that make up the application themselves. This part looks at the actual code of the application, stripping each routine down to its bare algorithms to explain the deep logic behind the store. This application is very powerful because, due to its modularity, you can change not only the look and feel quite a bit, but also the logic and capabilities. After all, you may find down the road that there are features you would like to add that are

unique to your own installation or inventory that are not necessarily appropriate for, or coded within, the generic store. These chapters should give you a firm grasp of how to add or remove from the programming of the Web store's core capabilities.

Finally, two appendices close the book. Appendix A discusses the format and usage of the CD-ROM and Appendix B reviews some of the basic tenets of Perl CGI.

Who This Book Is For

This book was primarily written for Web/CGI programmers who wish to set up online stores and programmers who are interested in examining a real-world CGI application that uses nearly every available feature in the CGI/Perl programmer arsenal. Secondly, content providers who are interested in expanding their services beyond simple HTML pages will benefit from the book as a primer on how to set up an online store.

All levels of programmers can benefit from this book. For example, the beginning CGI developer can use the book as a straightforward, example-based way of learning CGI while obtaining information on how to set up a useful real-world script. The script is explained simply enough that we hope it will be accessible to all, and we have taken great pains to isolate the basic installation and usage issues into their own part so that nonprogrammers do not need to spend much time deciphering the code if all they want to do is run the scripts.

Additionally, this book is for advanced CGI developers who are looking for a good, well-documented script that they can sink their teeth into to learn more about how a large, complex CGI application fits together. The discussion in Part Two about the details of the code behind the Web store application will help advanced developers to actually go in and customize the source code to the needs of their more demanding customers.