APPENDIX C

How Forms Communicate

With a typical static page, the user requests a page and the server gathers it from your folder (directory) and returns it to the user looking exactly as it did when you placed it on the Web server. However, it's possible for any page served to a user to be generated "on the fly" on the server when the page is requested. For example, let's say you have an online bookstore. There's a field on your form from which the user makes a selection. (This is called a s*elect* field.) You want this field's Options to include the books that your store has for sale. If you "hard wire" the titles in, every time the available titles change you have to open the page in GoLive and change the code within the list. Instead of doing this, you can have the server call upon a middleman. That middleman calls for your form page, reads the special tags/commands, learns to query a database and what to seek (a search for all sale titles in this example), merges the results into a sort of template page, and presents the users with an up-to-date list of titles.

The middleman is doing its work on the server so it's called a *server-side processing tool*. With any server-side processing tool, you still have HTML forms, with the usual checkboxes, radio buttons, text areas, submit button, clear button, and so on. And, of course, you can still use GoLive to create these pages. In addition to standard HTML, you have other special tags (commands) that are created specifically for the program that handles your processing on the server and returns a page to your users.

The page you create in GoLive is the page that contains the special tags (commands). Those tags or commands are specific to the technology you use to do the server-side processing. The page that is actually served to the user is a combination of your page, with the query results inserted in place of the tags. When the user views the source of the page he or she sees a regular HTML page because the insertions are complete.

In the following sections, I describe some of the server-side programs and what they do.

How Forms Communicate

Forms can do a lot more than just *collect* information. Forms can add life to a page by adding *interactivity*. When you create a regular Web page, placing your words, pictures, and other content, you may be adding multimedia and animation but you are still creating a static page — that is a page that remains the same every time a user calls it up. A form, however, creates the opportunity for communication between the user and your Web site. Any time you use your visitor's input to help form the page that's served to the user, you're creating a *dynamic* Web page.

Dynamic Web pages are customized for each user or type of user. This customization is based on information either directly provided by the user, such as the user's answers to a form's questions, or indirectly provided by the user, such as the user's environment. The page that is returned to the user is often constructed from data selectively retrieved from a database.

Forms, in conjunction with a *back end* database, can enable a user to search for specific information or automatically provide a user with date-sensitive or event-sensitive information. The form shown in Figure C-1 is a good example of a back-end database. Holli Boyd's database contains her postcards, which are merged into a form generated from user input to create pages on the fly. The system then sends the recipient notification of the card. Discussion forums and chats also use forms to enter and retrieve data from a server. Shopping carts and auctions are yet another use of forms.

It's important to realize that whereas GoLive enables you to design the form (as in Chapter 16), it *doesn't* provide the actual communication between the form on your page and the server-side actions that are required for the interactivity. When a user clicks your Submit button, a command is sent to the Web server, just as when any link is clicked or URL is typed into the address bar. But Web servers are designed to serve pages, not to search a database or phone a credit card company. So a third program, sometimes called *middleware,* must come into play. The commands calling for this program are hidden in the form request that gets sent to the server (because you put them there using the GoLive interface).

Figure C-1: Holli Boyd's electronic postcard (at www.hsbdesigns.com/postcards/cards.html) — a form that collects information, generates a Web page, and then generates and sends an e-mail to inform the recipient

The server recognizes that the hidden commands belong to another application and sends them on their merry way to the *middleware* or *agent* that the form addressed it to. This agent is actually another program. It recognizes these commands because they are written in its own special language, and it acts accordingly. Maybe its job is to send instructions to a database as requested and await a reply or maybe its job is to send a credit card authorization request and await a response. Whatever its task, it waits for a result, and then, using the result, dynamically creates a Web page that incorporates the newly obtained information. Everything that happens on the server, outside of the server's normal actions, is called *server-side processing*.

Several types of agents are available to do server-side processing. Understanding a bit about them helps you make a wise choice of hosts and/or servers. After a brief overview of this additional server-side communication, you'll understand how to build your form.

The CGI — Common Gateway Interface

The original and most common method of communication between the form and the processing that occurs after a submission is contained in a program called a Common Gateway Interface (CGI). The CGI program (or script) provides the rules through which the program communicates with the Web server. Just as several word processing or drawing programs exist in the computer world, many CGI scripts exist as well. Just as each word processor performs different functions beyond the most general task of creating text documents, each CGI script performs different functions, while the base one — communication to and from the browser — remains the same.

CGI scripts vary greatly. A CGI can be a simple text document with only a few commands designed to do one or two specific tasks, or it can be a full service program. For example some enable you to do just about any operation within a database that you'd normally do directly. You can write your own CGI script, or buy one that's commercially marketed.

Exactly how a CGI behaves varies from platform to platform, but the basics are the same. The information sent to the server by the form's submission includes an *action* that contains the name of the CGI script in some manner. It also contains certain information or commands that you embed within your form. Every CGI program knows to take the information it's been given, process it, and then return that information to the server.

How CGI works

Here's how a CGI program works (you can also follow along with Figure C-2):

1. When the user clicks a submit button the browser packs up the form's information and sends it to the Web server. The browser always carries information to the server. In this case it just carries more. The additional item it carries is the *Form Action*.
2. The server receives the information, but notices that the directions aren't intended for its normal function. It sees the Form Action, which includes the name of a CGI (or some kind of CGI ID), so it knows to send the directions to the CGI.
3. The CGI interprets the request and performs your embedded commands. For example, if the command is to search a database for all records containing a specific word, date, or event, the CGI program translates that command into a language that the database understands, and then waits until the search is complete. Armed with the data gained during the search and a special response command, the CGI then packs up that data in a way that the Web server can send it.

4. After doing its special job the CGI returns its information to the Web server.
5. The server sends the information back to the browser.
6. The browser displays the results for you. That little CGI script is a very powerful program.

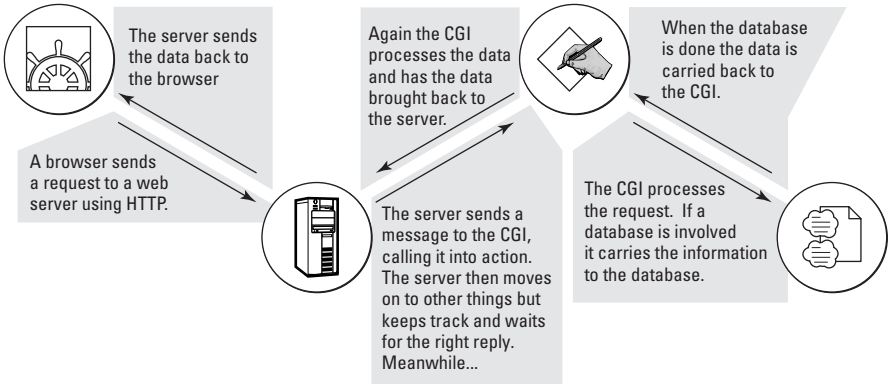


Figure C-2: A visual depiction of how a form communicates with a database and returns a response

Reinventing the wheel?

Many CGIs are available. Some are specifically designed to communicate with a specific type of database. In that case, you are able to have your users send information to a database, similar to when a user signs a guest book. You are also able to take communication between the user and database even further. For example, the user can send information to a database and the CGI would carry a result back from it, such as when a user searches for information from your site, and then receives the desired information. Another common use for a CGI is to process a form and return a confirmation by e-mail. Yet another is an online shopping cart.

Your choice of which CGI script to use depends upon several factors. First is what service you wish to perform and what platform your server runs on. (The functionality of the CGI must be written to run on the platform on which your server is running, such as Windows NT, Unix, or Mac.) Your ISP or site host's parameters (rules) are another major issue. You may be permitted to use any CGI scripts or be limited to a few specific ones. They may come at no charge, or you may be charged for some. If you use a database to hold your collected and distributed information, that's another major consideration because a CGI program must be able to communicate with that type of database.

You can write your own CGI script if you are so inclined, or you can use a commercial CGI. If you're doing only a simple, limited exchange of information it may be easy to write your own or borrow from someone else's. (It's common for people to help each other out by sharing on the GoLive Talk e-mail list. See Chapter 2 for more about such resources.) In cases where you exchange a lot of data or deal with many variables, a commercial CGI may be the better and easier choice. Shopping carts, for example, are usually intricate because they have to track a user's purchases, and then carry out a payment method. Several shopping carts, also called e-commerce solutions, are available today. Some are commercial, while others are freely shared. If you're considering buying one, or working on implementing a free one, investigate carefully. CGI scripts vary in quality. A badly written CGI script can be slow and frustrating and even possibly crash a server, whereas a well-written one performs quickly and efficiently.

To determine which CGI scripts you can use, begin by checking with your ISP/site host for answers to the following questions:

- What is the policy on CGIs?
- Are any provided with your account?
- If so, what does each do?
- Is a fee required to use it or any other limiting factor you should know of?

Plug-ins — closer than CGI integration

You're probably familiar with the concept of plug-ins because Photoshop and programs like it have long been using plug-ins. The key to a plug-in is that rather than act as a totally independent program, it *plugs* into the related program. For example, when you're using Photoshop you access a plug-in from within the application, rather than calling up a separate program.

Likewise, the Web server can have plug-ins, or *modules*, as they are often called. This eliminates one step of communication that would normally have to take place between the Web server and the CGI. Actually, it eliminates two steps — one as the request comes in and the other as the data is returned.

Plug-ins or modules are considerably faster than a CGI. However, they're more complicated to write because they need to be coded per the server's application programming interface (API) which includes properly tying it into the server. A module written for one server is not compatible with another server. Also, because they plug into the server, they must be installed; the server must be restarted because they need to load when the server starts up.

Most Web servers have a plug-in or module type interface. Apache servers have modules. Netscape has the NSAPI. Microsoft IIS has ISAPI, which is popular in the

NT world. Your site host may well be running plug-ins or modules that you're welcome to take advantage of. If you know one, you may even be able to request it specifically.

Due to what's involved you're unlikely to write a plug-in. They're the domain of software developers. That's why CGI scripts remain popular.

One of Apache's modules is "mod_perl," a Perl interpreter embedded in the Apache Web server. This enables you to run Perl scripts directly in the Web server. This is faster than running them via CGI, which is how Perl scripts are traditionally run.

From a form creation point of view, no difference exists between a plug-in (module) and CGI, so I use the term CGI in the rest of this appendix. (A CGI is actually a script, or program, but at times I take the liberty of simply using CGI instead of CGI script or CGI program. All three terms are interchangeable.)

Other Server-Side Technologies

Several more advanced tools exist for creating dynamic Web pages. Some of these tools include ColdFusion, Active Server Pages (ASP), Hypertext Preprocessor (PHP), and Sun's JavaServer Pages (JSP). Some are proprietary, meaning they require the manufacturer's specific server. You'll only use one of those if you definitely intend to host your site on that particular type of server. Others can be used on a variety of servers. As with CGIs, this is beyond the scope of this book. However, I introduce you to a few of the most popular advanced server-side technologies here. You'll need to check with your ISP to learn whether any of these are available for you. Then you can visit the recommended pages to learn more about each option.

ColdFusion

ColdFusion (not to be confused with the putative breakthrough in physics) is a product from Allaire. Using ColdFusion, programmers create *templates,* which are files containing standard HTML tags and ColdFusion tags. You can create these templates in GoLive and keep them in your Site Window with the rest of your site.

The ColdFusion tags, of the form `<CFthecommandname> xyz </CFthecommandname>`, enable database queries, invocation of COM objects, and more. ColdFusion's strength is that it does not require knowledge of a traditional programming language, because its tags look like HTML. In fact, the Allaire calls its tags CFML (ColdFusion Markup Language.) While early versions of ColdFusion worked via CGI, recent versions use the API—a programming protocol provided by various Web browsers such as Apache. ColdFusion runs as a separate application (called the ColdFusion server) but because of the API, it is tightly connected to the server software. It runs on Windows NT and Solaris platforms, and, as of this writing, is due to run on Linux machines.

Out of the box, GoLive doesn't recognize ColdFusion (CF) tags as legitimate tags. You have two easy ways to ensure GoLive will respect your CF code. You can enter the CF codes into GoLive's Web settings so GoLive recognizes the tags as legitimate code, or you can use GoLive's `<noedit>` tag around your CF code. (I show you this in the next sections.)

In ColdFusion you can have a template (page) called ChooseBook.cfm that looks like this:

```
<HTML>
<HEAD>
<CFQUERY NAME="Books" DATASOURCE="BooksDatabase">
select title from BooksTable where InStock='True'
</CFQUERY>
</HEAD>
<BODY>
<FORM ACTION="BuyBook.cfm" METHOD="POST">
<CFSELECT QUERY="Books"></CFSELECT>
<INPUT TYPE="Submit" VALUE="Proceed to checkout">
</FORM>
</BODY>
<HTML>
```

When users browse to ChooseBook.cfm, they see a regular HTML page, with a Select field whose Options are the book titles. If they view the source, all they see is pure HTML, no ColdFusion tags such as CFSELECT, because the server has executed the query and replaced the CF tags with HTML and data. This example pertains to CF, but ASP and PHP operate similarly.

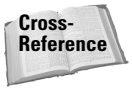
You can learn more about this technology at `www.allaire.com/products/coldfusion`.

ASP — Active Server Pages

ASP, or Active Server Pages, is Microsoft's offering to the world of dynamic Web pages. It is designed to run only on Microsoft IIS Web servers under Windows NT. However, a program called Chili!Soft ASP enables Unix servers and other platforms to run ASP as well. (A Mac flavor does not exist as of this writing.)

Using ASP, programmers create pages with HTML and scripts written in VBScript, JScript, or Perl. ASP tags are of the form `<% xyz %>`. These scripts tell the IIS server what to do. (They'd be useless on any other server because these tags are proprietary to ASP technology.)

You can use GoLive's Dynamic Link feature to create ASP pages.



See Chapter 24 to learn more about dynamic links.

PHP — Hypertext Preprocessor

PHP, or Hypertext Preprocessor, is a server-side HTML-embedded scripting language. It is a free, open source alternative to other commercial middleware. PHP can be used to perform functions such as collect form data, generate dynamic page content, manage user session information, or send and receive cookies.

GoLive 5 has much improved support for PHP code within your HTML documents. Problems in GoLive 4 with quotes and end tags (`</tag>`) within PHP code are now resolved.

Instead of writing a program with lots of commands to output HTML, you write an HTML page with some embedded code to perform a task (in this case, output some text). The following is an example:

```
<html>
<head>
<title>Example</title>
</head>
<body>
<?php echo "Hi, I'm a PHP script!"; ?>
</body>
</html>
```

The PHP code is enclosed in special start and end tags that enable you to jump into and out of PHP mode. It runs on Windows NT, Unix, Mac OS *X*, and Linux platforms. It can be run via CGI or as part of the Apache Web server and more than one million servers with PHP are installed worldwide. (You need to ask your site host if the server supports PHP.) PHP is very adaptable in that it provides four different tags you can use to embed code into a page.

Writing a database-enabled Web page in PHP is incredibly simple and it supports Adabas D, InterBase, Solid, dBase, mSQL, Sybase, Empress, MySQL, Velocis, FilePro, Oracle, Unix, dbm, Informix, and PostgreSQL. Other database systems such as Microsoft SQL Server are supported through ODBC. PHP also has support for talking to other services using protocols such as IMAP, SNMP, NNTP, POP3, or even HTTP. You can also open raw network sockets and interact using other protocols.

PHP can be used for many applications on your Web site such as shopping carts, discussion forums, content management systems, dynamic page creation, personalization and customer logins, security, and many more.

The example similar to the preceding ColdFusion one may look like this in PHP:

```
<HTML>
<HEAD>
<?php mysql_connect("user", "pass", "bookstore")
$query = "select title from BooksTable where InStock='True'" ;
$result = mysql_query("BooksDatabase", "$query"); ?>
</HEAD>
<BODY>
<FORM ACTION="BuyBook.php" METHOD="POST">

<SELECT name="selectName" size="1">
<?php while($row = mysql_fetch_array($result)) {
print '<OPTION value="';
echo $row["ISBN"];
print '">';
echo $row["title"];
print '</OPTION>'; ?>
</SELECT>

<INPUT TYPE="Submit" VALUE="Proceed to checkout">
</FORM>
</BODY>
<HTML>
```

When users browse to ChooseBook.php, they see a regular HTML page, with a Select field whose Options are the book titles. Like with CF, all they see from the source is pure HTML, no PHP code, because the server has executed the query and replaced the PHP tags with HTML and data.

You can learn all about PHP at `www.php.net`. You'll find many tutorials, code examples, and applications that use PHP both on the PHP Web site and on many other support sites linked to from the PHP site.

Choosing a technology

When choosing among these technologies, you need to consider several factors. They include the following:

- **Server platform.** The hardware and operating system. For example: a Pentium II PC running Windows NT, a G4 running Mac OS *X* Server, or a Sun Enterprise Server running Solaris.
- **Web server software.** The software that processes the HTTP requests. For example: Apache, IIS, Zeus, Xitami, AOLServer, or WebSTAR.
- **Cost.** Many factors can influence the cost of using a particular technology including Web hosting costs, development time, product licensing costs, and contracting costs if it is a large site or you are having someone do the programming for you.

- **Ease of programming.** You need to consider your background and the learning curve of using each technology (for example ColdFusion may be easier if you only know HTML, whereas PHP may be a better choice if you can already program in Perl or C). You also need to think about the availability of programmers in your area, whether your client wants to maintain the site themselves, and other issues. Consider maintainability as well as site construction.
- **Portability.** Can you easily move your site onto a different server platform or Web host? How easy is it to find another Web host that supports the technology you are using?
- **Scalability.** If your site gets a lot of visitors, or if it's featured on a major news or portal site (such as Slashdot, Yahoo, NBC, and so on), is the technology you have able to cope with the number of hits you are receiving?
- **Support.** Is support for the technology readily available? Is there a large user community that you can interact with if you have problems?

Using a technology

If you are building a form that uses any server-side processing and requires proprietary tags, I recommend that you design your page and form first. Then, after all of the page and form's visual elements (and standard HTML) are in place, switch to the Source tab and add the technology-specific code.

As you do, refer to the "Dealing with Nonstandard Tags" section in Chapter 16, which covers all of the standard form elements.

Understanding GoLive's Trouble with Non-HTML Tags

Recall that GoLive sometimes has "problems" with some code. In reality, no single page-creation program can know, identify, and make it simple to work with every CGI and server-side processing application. Simply too many of them exist to accommodate and the syntax of their commands is too varied. The primary language of the Web is HTML so priority must be given to HTML code. Extensible Markup Language (XML) is becoming more popular and offers advanced page creation so advanced support for XML is an excellent addition to GoLive. When it comes to supporting anything outside of the HTML-family norm, any program must make a conscious decision to support that code.

Like every other program, GoLive can choose to support some such code, but cannot possibly begin to cover them all. For example, double brackets (<< or >>) are also problematic, as they are not proper HTML syntax. (They are also problematic in Dreamweaver.)

While GoLive cannot be everything to everyone in its visual interface (Layout mode), its Source mode is a full-fledged HTML-editing program that can be used to write any code for the Web. You just may have to do some of that code in the good-old HTML source.

That said, GoLive 5's support for non-HTML tags and code is greatly improved over GoLive 4. The new 360Code feature means that it will be a rare occasion that GoLive will change or have problems with your non-HTML code.