RUNNING A SUCCESSFUL SITE

Three features of successful websites

All websites make sense to the people who create them; their purpose is obvious, their content is laid out clearly, and navigation is immediately comprehensible. The people who use them, however, do not always find sites to be so transparent. They cast around for clues about what services a site offers and how to use the site.

Although sites can succeed with greatly different site designs, Web managers must keep in mind that, inevitably, users will become lost, follow paths the designers did not anticipate, or follow a broken link to a dead end. Something always goes wrong, and one measure of a successful site is how well the Web manager supports users when something does go wrong.

Usability expert Jakob Nielsen has done research showing that user behavior falls into three broad categories (Nielsen, 2000, p. 224-225). About one-fifth of all users are *link-dominant*, preferring to find even known pages through links provided as part of a site's navigation. About half of all users are *search-dominant*, preferring to use search engines and search results to find what they are looking for. The remaining users switch between these two modes, most often following links until they become lost, and only then resorting to search behavior.

Users who alternate usage behavior are frustrated, often angry, and unlikely to give the site many more chances. To take advantage of that last chance, successful sites provide links on every page to a site index for link-dominant users and a search engine for search-dominant users. They also provide carefully crafted error messages for inevitable occasions when users follow links to nonexistent pages or pages they have no permission to access.

Site index

A site index is a list of a site's available resources in alphabetical order. Some sites choose to make separate alphabetical lists broken down by major area of the site's structure. For the majority of sites, maintaining the index manually is not difficult, although some larger sites use software such as Site Expert to generate indexes automatically.

Site Expert,
www.xtreeme.com/sitexpert

One concern with site indexes, especially in their role as a tool of last resort, is that the alphabetical entries must use all variants of a term a user is likely to look for. Links to the library catalog, for example, should appear under both "catalog" and "library catalog," as well as any other name commonly used for the catalog.

Site search

Compared with a site index, a site search engine is a substantially more complex service to maintain. Two main options are available; some sites

install and run search engine software on their own servers, and others link to a customized search service hosted on an external site. Hosted search engines, using services like Google or Picosearch, require substantially less work to set up.

Typically, the hosted service needs to know the site's homepage URL. It then occasionally runs a crawler program to read and index each page on the site. One drawback to this arrangement is that the crawler program runs on its own schedule; sites that move pages or revise content may be unable to update the site search in a timely way. Another drawback for some sites is that hosted services typically split their brand-name identities between the library and the search service. Hosted services also may include advertisements for other services, especially on their free versions.

Locally installed search engines, such as ht://Dig and SWISH-E, require software to be installed on the local server that includes both a program to compile the index and a CGI application for actual searching. Some local search engines index files by viewing files and directories on the server. Others retrieve files via HTTP commands, acting as a small-scale indexing crawler. One advantage to this second approach is that the indexer sees the output of dynamic pages in addition to static HTML pages. Another advantage is that well-behaved crawlers automatically skip any parts of a website that have been identified to Internet search engines as excluded from indexing.

The Standard for Robot Exclusion (SRE) describes the most widely recognized way to inform crawlers what parts of a website to ignore, using a file named robots.txt in the server's root document directory. The following robots.txt file would place a directory named *staffonly* off limits for all compliant indexers, except for any with *htdig* in their User-Agent header:

The Standard for Robot Exclusion (SRE), www.robotstxt.org/wc/exclusion.html

```
User-agent: *
Disallow: /staffonly

User-agent: htdig
Disallow:
```

Since few authors have access to the document root directory, managers should encourage use of a newer, slightly less widely supported standard that uses a meta element in HTML markup. This markup advises compliant indexers not to index the current page and not to follow any of its links for further pages to index:

A thorough review of site search tools is available at www.searchtools.com.

```
<meta name="robots" content="noindex, nofollow">
```

The art of the error message

On most if not all websites, the homepage is the most important page. It is certainly the most visible page; the one responsible for defining what the site is and does; the one whose URL is publicized; and the one that often receives the most hits. For these reasons, the greatest amount of design effort goes into the homepage, even to the point where second- or third-level pages may receive little more than cursory consideration.

Unfortunately, another important page, the error page, is so little considered that it is not even explicitly written, but left for the Web server to provide from a minimal template. A good error page has an ability to

assist, inform, and explain that may be second only to the homepage in importance.

The error message by definition alerts the user when something has gone wrong. It always appears at a teachable moment; one of the most important lessons it can teach is that the problem is probably not the user's fault. A well-designed page explains the most likely causes of the error and suggests the most likely solutions, putting users back on track rather than stopping them at a dead end.

In this context, the errors under discussion are specific HTTP errors, rather than problems such as database searches that retrieve no results. Those problems also need thoughtful error handling, but the error pages sent by a Web server are generated in response to problems that prevent the site from displaying the requested page.

The HTTP errors a user is most likely to face, and which a server should provide assistance with, are:

- **404 Not Found.** The user has requested a URL that is not on the server.
- **403 Forbidden.** The server's configuration forbids it from providing the requested URL. Users receive a 403 error if the URL cannot be sent to the IP address or host domain, or if file permissions on the server prohibit the Web server from reading it.
- **401 Unauthorized.** The requested page requires authentication credentials—a username and password—and the request supplied either no credentials or incorrect credentials. Browsers typically present the user with a login prompt one or more times in response to this error, then display the server's 401 message.
- **500 Internal Server Error.** In trying to provide the requested page, something on the server has gone wrong. When server-side scripts or dynamic pages break or fail to run completely, they often general 500 errors.

Other status codes indicate less common errors. A complete list explained in detail is in the HTTP 1.1 specification.

HTTP 1.1 specification, www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

By default, Web servers provide messages in response to each of these errors, although often not in an especially helpful way. A page stating simply 404 Not Found does not provide users with helpful information.

Taking control from your server (and Internet Explorer)

Most Web servers, including both Apache and Microsoft's Internet Information Server, provide ways to customize what the server shows users in error situations. This customization allows Web designers to decide what options to offer users and to keep error messages within the common look and feel of the site.

Apache. As with all its configuration options, Apache controls its error messages through directives, or one-line commands in one of its configuration files. In this case, the relevant directives may be established for an entire site in the main configuration file or only for specific directories in a .htaccess file.

Apache provides two error-related directives. The first, ErrorMessage, followed by the error status to be replaced, changes Apache's brief description

of the error to another brief text message. The following directives replace two common error messages:

```
ErrorMessage 404 This page does not exist on the server.
ErrorMessage 403 This page is only available from on-
campus locations.
```

The ErrorMessage directive is, at best, a partial solution. Simply replacing one message for another cannot add much functionality. A more complete solution is to write a page that says and does everything desired in an error situation. This page can be static HTML, a CGI script, a PHP or ASP page, or any other format. In general, any dynamic technology is better able to customize itself to the specific error facing the user.

Once the page is created and available on the Web server, Apache can use it instead of writing any textual message through the ErrorDocument directive:

```
ErrorDocument 404 /errors/404handler.php
ErrorDocument 403 /errors/403handler.html
```

As a general reminder, any changes made to Apache's main, site-wide configuration files go into effect only after Apache is stopped and restarted.

IIS. In some ways, customizing error messages in Microsoft's Internet Information Server (IIS) is easier than in Apache. Fine-tuning those errors messages to respond, for example, to different reasons why a document was not found is possible. The trade-off is that more files must be edited.

IIS always generates error messages from a collection of HTML or ASP files, located by default in the Windows help\iishelp\common directory, so no equivalent exists for Apache's ErrorMessage directive. IIS provides several default HTML files for some common errors.

There are separate files for 403 errors due to the user's IP address being disallowed and 403 errors due to the server having too many simultaneous users. This level of detail benefits the user but adds to the job of customizing the complete set of error messages.

Internet Explorer's friendly error messages. Since Version 4.0, Internet Explorer has had a setting called "Show friendly HTTP error messages." This setting is enabled by default and when it is enabled Explorer replaces many server-generated error messages with its own version of, for example, a 404 Not Found message.

Reasonable arguments exist for and against this practice, but it does show that Microsoft recognizes the great number of servers that never do anything to provide useful information in error messages. Explorer's friendly messages are more useful than nothing, but less useful than a carefully tailored page with information and navigational help specific to the site.

Fortunately for sites that do customize their error messages, Explorer can be forced to display the server's message rather than its own. Explorer uses its internal error messages only if they are enabled *and* if the server's error message has a total length, including HTML and images, of less than 512 bytes. If the server's message is longer than 512 bytes, Explorer always displays the server's error message.

What makes a good error message?

The most common error users encounter is 404 Not Found, and in many ways this error is the most confusing. Explaining why a user's address is denied access or why a username and password fail is easy, explaining why a link from some other site, or a URL typed in by the user, fails to find the requested document is usually difficult.

Some common reasons exist for a 404 error, however, and a 404 error page should provide solutions to them. Users entering URLs manually are prone to typographical errors. A static URL page can direct users to re-edit the URL in the browser's address bar; a dynamic page can create a form, present the current incorrect URL in an editable input box, let the user make corrections, and jump to the corrected URL. Some sites even have scripts that make guesses about likely candidates based on the incorrect URL—pages whose URL differs only in capitalization, *.html vs. *.htm differences, and so forth—and present those variations as links.

Careful log analysis may suggest patterns in 404 errors, such as hits on one server for a page that actually resides on another server, or hits in a directory that was renamed at some point in the server's history. A static HTML page can describe this problem in general, and a dynamic page can do some guesswork about which such problem applies. If a pattern of 404 errors is sufficiently predictable, it can be fixed once and for all by configuring the server to redirect users from the common incorrect URL to the correct one.

If a site has IP- or password-protected pages, it needs well-crafted messages for the 401 Unauthorized and 403 Forbidden errors. The user needs to know why a requested page has restricted access and to whom it is available. If different parts of the site are protected with different passwords, the message should make clear which password is needed for the requested page.

Sites that rely heavily on dynamic pages eventually generate 500 errors. Although bugs related to broken scripts are most common in debugging new scripts, they can happen in production when problems arise. For example, a script that tries to access a database on a server that is down may die before generating any necessary HTTP headers.

The script should be written to handle this problem gracefully, informing the user of the problem and exiting without a server error. For this error page, rather than asking users to correct their URLs, an error message needs to assure users they have found the correct URL and explain that the problem is at the server's end, not theirs.

Regardless of the specific error, error messages also should give users the option to start navigating the site from scratch. Links to the homepage, top-level pages, and any site index or site search should be prominent. A search box for the site search is helpful.

Things to avoid

Send the right status—don't cheat. As with other aspects of website management, the end result of writing error messages has to make sense to people and computers (browsers, proxies, link checkers).

Some sites display human-intelligible error messages by redirecting the user from the requested URL to another page. This approach creates two problems: it loses the requested URL, so there can be no option for the user to make minor typing corrections to go to the right page; and it does not actually return an error status code but instead sends an HTTP redirection status.

A true error status causes link checkers, Web crawlers, and other automated agents to mark that URL as nonexistent or inaccessible; a redirection status causes them either to keep that URL marked as good, or actually update bookmarks and links to go directly to the error page.

Contact referring sites—don't ask users to do your work. Any dynamic page can easily determine the URL from which the user linked—the referring page in HTTP's terminology—so making a link to that page is easy—just add a message to the user, "Go back where you came from and tell them to fix their links." This approach might work if the user wants to spend time solving someone else's problem. Most users, though, just want to move forward and find what they were looking for in the first place.

The proper person to take up this problem with the referring site is the Web manager. A routine scan of log files for 404 errors will generate a list of referring URLs. Many of those pages provide contact information for an author or Web manager in a position to correct the bad links. Most Web managers appreciate knowing about problems on their own sites.

Dynamic pages

HTML is at heart a language for describing the structure of documents, and a characteristic of documents is that they do not change. Many Web designers, however, discover needs for services that do change, depending on a variety of circumstances, such as:

- The user's IP address. Is the user on-campus or off-campus? In the library building or elsewhere?
- The URL from which the user followed a link to the current page. Did the user follow a link from the main library's list of databases or a branch library's list?

The URL of this "referer" [sic] is provided by the browser as part of the information sent to request a page. It is an optional header, and some browsers or firewalls are configured not to send it. The HTTP specification also prohibits sending it in cases where the user follows a bookmark or types the page's address manually. In addition, a sufficiently motivated hacker can send a fake referer header. So scripts should be built to keep working if there is no referer, and any security based on the user coming to a page with a specific referrer is weak, at best.

- The time of day. Is the circulation desk staff currently answering the telephone?
- The value of an HTTP cookie previously set by the user. Did this user set a preference of full displays or brief displays when viewing database search results?

Referer was misspelled in the original HTTP specification and continues to be misspelled for compatibility with older scripts.

Cookies are optional. A growing number of users take advantage of options offered in newer browsers, or in third-party software utilities, to eliminate or restrict the cookie information they send to Web server.

- The user's responses to a form. Did the user ask to view the list of available electronic journals by subject or by name?
- The contents of a database or other file that the Web server can access. What entries in the library's database of Internet resources relate to environmental impact statements?

Many other pieces of information about the user's environment are available to pages on a Web server. Pages that are capable of responding to a user's input or environment are called dynamic pages. Many ways exist for creating them. Dynamic page technologies are divided into two main categories: Server-side technologies and Common Gateway Interface (CGI).

The terms **dynamic pages** and **Dynamic HTML** are confusingly similar. The broader term, dynamic page, applies to any method of creating pages that provides different content under different conditions, or that determines a page's content only when the page is requested by the user. Dynamic HTML refers to several techniques that use client-side scripting (usually JavaScript) to interact with or modify the content of an HTML page, especially by altering a page's style sheet after it is loaded.

Server-side technologies

Server-side technologies require data to be sent to the server, where any necessary computation takes place, giving the server access to any user input. Because all the work is done on the server, server-side technologies avoid any problems with browsers that have disabled client-side scripting or have incompatible versions of client-side scripting languages. But most server-side technologies require careful configuration of the Web server and if used carelessly can create security problems for the server. Because of this risk, system administrators often enable these technologies only for certain directories or accounts on their server.

Server-side includes (SSI)

Many Web servers provide a feature called server-side includes (SSI). As indicated by the name, SSIs include the contents of one file within a second file at the time the server sends the second file to a user.

The most common use of SSI is to gather the markup for document sections, such as navigation links, common to many or all of a site's pages, and write it only once in a separate file. All other pages can then include that file instead of writing their own copies of the links. If a site needs to change its common links, the links only need to be edited in one place and the change appear in all pages.

For example, a site's design might require the following markup on all pages:

```
…<body>
<p><a href="/index.html">Home</a> -
<a href="/search.cgi">Site Search</a> -
<a href="messageform.cgi">Send Us a Message</a>
</p>
<h1>…
```

Constantly reiterating this markup can be handled in a server-side include by moving the repeating text to a separate file—in this case "/includes/nav-tabs.htinc"—and referencing that file as an include:

```
…<body>
<!--include virtual="/includes/nav-tabs.htinc" -->
<h1>…
```

The include is a specially formatted HTML comment, whose first character is a hash mark (#) followed by the include command. In this case, the "virtual=…"

option specifies what file to include, relative to the Web server's top-level document directory, also called the document root.

In addition to this basic function of including another file, most implementations of SSI provide ways to include values of some variables, like the date a file was most recently edited, or to include different files or blocks of text depending on aspects of user environments, such as IP addresses. This example includes one file if the user's IP address is in a specified range and another if it is not:

```
<!--#if expr="${REMOTE_ADDR} = /^234.56.78/" -->
 <!--#include virtual="on-campus.htinc" -->
<!--#else -->
 <!--#include virtual="off-campus.htinc" -->
<!--#endif -->
```

Many Web managers choose not to enable SSI support. The reasons are twofold:

- Transmitting a page with SSI requires the server to work harder than if it is transmitting static HTML. Instead of simply sending a file, the server must open each page, scan it for SSI markup, find and open any included files, and then send the processed result. This extra work, though, is not a concern for any but the most overloaded Web servers, and SSI can usually be enabled for all HTML files without noticeable performance problems.
- SSI, like all dynamic server-side technologies, introduces potential security risks. A careless or ill-intentioned SSI can allow a server to send sensitive files, such as lists of accounts and encrypted passwords, which would normally not be made available to the general Web public. Because of these risks, many sites enable SSIs only for certain directories that can be closely monitored by the Web manager.

SSI is enabled by default in IIS. To enable it in Apache for all HTML files, add the following lines to the Apache configuration files:

```
AddType text/html .html
AddHandler server-parsed .html
```

Common Gateway Interface (CGI)

The Common Gateway Interface (CGI) is the oldest technology for generating dynamic pages and processing online form submissions, and it is still one of the most common. CGI is not a programming language, but a description of how a program can interact with users through a Web server.

This specification establishes basic programming structures: how to read input, how to write output, and what environment variables to expect. Since almost every programming language uses these three structures, writing CGI applications in any language an author already knows is possible. This flexibility allows programmers with experience in other areas to be up to speed quickly in a CGI environment.

In practice, the most common language for writing CGI applications is Perl. Perl's strengths in managing strings of text are well-suited to the CGI environment, where all program input and output is made up of text strings. Its use also is because Perl's popularity in the programming community rose at the same time CGI applications became common features on websites.

A CGI script is an executable program whose output must create the entire source of the document sent to the user and at least some of the HTTP headers. This output is usually a complete HTML document, although CGI can be used to generate any type of document. CGI scripts can easily output any type of text document and with the necessary code can be used to create images from scratch.

Apache identifies files as CGI scripts in two ways:

- The first is to associate a file extension, usually .cgi, with CGI scripts:

  ```
  application/x-www-form-urlencoded
  ```

- The second is to establish one or more directories where all files are to be executed as scripts, using the ScriptAlias directive:

  ```
  ScriptAlias /cgi-bin/ /web/apache/cgi-bin/
  ```

To enable CGI scripts in a directory under IIS, view the properties dialog box for the directory and set its execute permissions to "Scripts and Executables."

Most Web servers require the CGI script to provide the HTTP Content-type header, identifying what MIME type the program output comprises. In most cases, this type is HTML; HTTP headers are separated from their accompanying content by an empty line ended with a carriage return/line feed combination. A typical beginning for Perl CGI scripts is to print out Content-type: text/html\r\n\r\n.

Middleware

Many programs and programming languages make creating dynamic pages easy. In many cases they emphasize communications among Web servers and database servers. These programs are often collectively referred to as Web application platforms or Web middleware.

In general, any of these programs can successfully handle the typical tasks of dynamic pages. Choosing one may involve any program's performance or capabilities but more often involves issues of available support, peaceful coexistence with other Web servers in an organization, common pairings between certain specific Web servers and specific middleware applications, or simply personal preference.

If CGI scripts must generate the entire output of a page, pages that use middleware technologies can be partly or mostly HTML markup, interspersed with markup that is interpreted and acted on by the middleware program when the server sends the page to the user. For example, a page that uses PHP to display the user's IP address can include:

```
<p>Many library databases are only available from library
IP addresses. Your address is

<?php
 print $REMOTE_ADDR;
?>

We are unable to provide access to that address.</p>
```

Common middleware technologies

PHP is a programming language designed to work on Web servers from code included in Web pages. It is both open-source and free, and it communicates easily with database programs, including the open-source MySQL database. This combination of features has made PHP a common inclusion with Linux server software, although a Windows version also is available. PHP also can be run with either IIS or Apache for Windows.

PHP, www.php.net

Microsoft's Active Server Pages (ASP) technology is not a language itself, but a mechanism for incorporating programs written in other languages into Web pages themselves. In practice, ASP is used almost exclusively with IIS servers, although Apache can be compiled with an extra module to support it, and most ASP scripts are written in Microsoft's Visual Basic Script.

Microsoft's Active Server Pages (ASP), http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000522

Cold Fusion is a language created by Allaire (now owned by Macromedia) that implements a set of tags similar to HTML tags, with an emphasis on database interaction and ease of learning.

Cold Fusion, www.macromedia.com/software

Java Server Pages (JSP) is an open-source technology for writing server-side scripts with the Java programming language. Using Java as the underlying language makes JSP an attractive choice for organizations doing other development work in Java.

Java Server Pages (JSP), http://java.sun.com/products/jsp

In general, all these languages are used in similar ways for similar tasks. The most common task for any dynamic page is to process the results of HTML forms. Another is to access discrete elements of site content stored in a database—MySQL, Microsoft SQL Server, Oracle, or other database programs that communicate via SQL. Scripts in dynamic pages then execute database queries, retrieve records needed for a page's content, and format it with HTML markup. This separation of content from markup allows Web managers to update content and revise page designs as separate tasks on separate schedules.

SQL: Structured Query Language

Client-side technologies

Client-side technologies run on the user's workstation, often within the Web browser itself, and can function without needing to communicate with the server. They cannot, however, send data to the server for processing by themselves.

Java applets

Sun Microsystems released the Java programming language and demonstrated a browser running Java applets in 1995. This release galvanized the Web development community, generating tremendous interest in the possibilities of using the Web to deliver not just static pages, but services that were more like small software applications (hence the name applets).

Early expectations for Java applets turned out to be exaggerated, and much of what was expected from them has been developed instead in JavaScript. Java has instead become a more common choice for large-scale application development on servers and for server-side scripting of Web pages, but Java applets remain an important way of distributing application-like services over the Web.

Despite the similarity of their names, Java and JavaScript are not related. Java is a general purpose programming language that programmers can use to make both applets to run in browsers and complete, stand-alone applications to run either on servers or on user workstations. JavaScript is more specifically a scripting language for manipulating data in Web pages from within a browser.

JavaScript (ECMAScript)

In 1995, as part of its development on Version 2.0 of its browser, Netscape introduced a scripting language called LiveScript; to emphasize its role as a complement to Java applets, Netscape changed the language's name to JavaScript before its official release. Netscape subsequently submitted the language to the European Computer Manufacturers Association (ECMA) for approval as a standard. The ECMA formally adopted it under the name ECMAScript, but it remains JavaScript in the minds of most Web managers.

As a client-side language, the source code for a JavaScript application is either contained within HTML pages or is delivered to a browser in a separate file and is the interpreted and executed by the browser. Running a script within the browser has the advantage of putting no burden on the server, and including it within HTML markup allows any author with permission to create HTML files the opportunity to provide scripts.

By contrast, many Web managers restrict use of server-side scripting technologies to the accounts of trusted authors or to document directories with access limited to specific authors.

A drawback to the client-side scripting model, however, is that the Web manager cannot know all the details of the user's JavaScript support. Over the years, different versions of different browsers have offered widely varying support for specific functions and procedures in the language; adding to the confusion is the fact that users can choose to disable client-side scripting altogether. The availability and exact functionality of server-side languages, on the other hand, is always knowable.

The company thecounter.com sells usage statistics software to many websites. In addition to generating statistics for individual sites, thecounter.com gathers and publishes aggregate statistics for many aspects of use across all its sites. According to its global statistics, the number of Web users with JavaScript disabled may be as high as 10%, although completely reliable numbers are difficult to determine.

Users with certain cognitive disabilities are more likely than others to disable JavaScript. In addition, users with some motor disabilities may be unable to execute scripts that require mouse movement. For these reasons, accessibility guidelines usually require that pages with scripts remain usable when scripting is unavailable.

The Document Object Model (DOM)

One increasingly important role for client-side scripts is to interact with and manipulate the structure of documents through a programming interface called the Document Object Model (DOM). This interface is a generalized method for describing a document as a set of objects, each with relationships to other objects and with properties of its own.

In HTML and XML, objects correspond closely to elements, and an object's properties correspond to its attributes and style rules. Properties for a paragraph object, for example, might include a unique ID value, the size of its margins, the color of its background, and its content. Properties for an image object might include the location of the image file, its dimensions, and its alt text and title attribute. The document's object model allows a script to determine that the image object is located inside the paragraph object.

Armed with this information, scripts can move objects within the document's structure and manipulate the values of properties, such as changing one image file for another or revising the style properties of objects based on user actions or on environment data the script can access.

A document can include lists of psychology databases, education databases, and sociology databases. JavaScript code can prevent all three lists from displaying until the user's mouse passes over the item—or the user

selects the item with the keyboard. The script then switches that list's visibility property from hidden to visible. When the user selects another database, the first list's visibility property switches back to hidden, and the new list appears in its place.

The greatest barrier to widespread use of the DOM in scripting is because different browsers have created different internal models of document structure. With Netscape 4, Internet Explorer 5 for Windows, and the World Wide Web Consortium establishing three incompatible models, authors have typically had to include three separate behaviors in each script, plus a fourth display option for users with scripting disabled.

More recently, however, Internet Explorer 6 for Windows and Mozilla-based browsers (including Netscape 6 and 7) have adopted the W3C standard. This standardization make simplifying scripts possible with only a single DOM.

Understanding Web logs

Web servers typically record every interaction, or transaction, in one or more log files. Access to these logs is one of the most important tools a Web manager can use to measure how and how much the server is used, diagnose problem reports, and discover other problems, such as broken incoming links.

Combined, these log files represent an accurate picture of what the server is doing, although the logs may not reflect exactly the same as what users are doing. Many users may be inside a company firewall or behind an Internet service provider's (ISP) proxy server that maintains a cache of previously retrieved documents. In those cases, there will be times when users access a page more than once, but only one retrieval is logged at the original server. Subsequent hits are made on the intermediate cache, preventing server logs from providing an accurate record of hit counts.

At the same time, some log analysis software attempt to measure the number of visits made to a website. A visit is one user retrieving one or more pages during a single browser session. This software may define a visit in the log file as hits coming from a single IP address within a given period of time. Multiple users coming through a proxy server all appear to have the same IP address, however, and some ISPs reassign a user's IP address within a single user session, so counting visits in this manner provides a rough estimate at best.

Log analyzers also may expect servers to assign a particular cookie to each new visitor and count visits by cookie assignment, but users have a growing number of options for managing and eliminating cookies in their browsers, so that count also is less than reliable.

If log files cannot count either hits or visitors with total accuracy, what good are they? They do record usage trends reliably. If the number of hits logged for a page, or entire server, is 50% higher in January 2003 than in January 2002, a Web manager can trust that actual use really has increased about 50% in one year, although the specific numbers involved may be inexact.

Likewise, if a list of databases by subject receives twice as many hits as a list of databases by title, then—whatever the recorded numbers—users are relying on subject browsing of those resources substantially more than title browsing.

What the server logs

Typically, a Web server maintains two log files, although this number can vary. Servers may maintain separate sets of log files for each virtual host they maintain or for separate file directories. Both Apache and IIS allow Web managers to create custom log files, recording data other than what is normally found in the logs.

By default, both Apache and IIS maintain one log with a record of each request handled by the server. In most cases, this log is a plain text file with one line per transaction. By default, Apache's transaction log is named access_log and is located in the Apache logs directory. IIS logs are placed in the Windows system32\logfiles directory and default to different names based on the log format chosen and the date.

The first widely available Web server programs settled on a single format for their log files. This format can be used by either Apache or IIS, and in fact by most other servers, and is now referred to as the common log format. It records the domain name or IP address of the user's computer (or proxy), the user name in use if any, the date and time of the request, the actual request made to the server, the status code of the server's HTTP response, and the number of bytes sent by the server. For example:

```
staff-pc-118.foo.org - - [11/Nov/2002:18:32:48 –0500]
"GET /info/library-hours.html HTTP/1.1" 200 8240
```

In this case, an incoming request was received from the machine staff-pc-118.foo.org on Nov. 11, 2002, at 18:32:48 (like most software, Web servers use 24-hour times). The two dashes show that no user name was provided for either of the authentication schemes logged.

The common format logs names from the seldom-used identd authentication system defined by Internet RFC 1413 and by the standard Basic Authentication are supported by almost all Web servers. The -0500 identifies the difference between the server's time zone and Greenwich Mean Time; -0500 is the U.S. Eastern time zone, during standard time (Eastern Daylight Time is –0400).

The HTTP status code is a three-digit number indicating either that the server provided the data requested or what else happened instead. All codes are listed in Appendix A. For studying logs, the most important are usually Code 200 (OK: data was sent successfully), 301 and 302 (permanent and temporary redirections), 401 (Forbidden: the user has no permission to receive the data requested), and 404 (Not Found: the server cannot locate the file or script requested).

The actual HTTP request asked to use the GET method to retrieve the URI "info/library-hours.html" using HTTP version 1.1. The server responded with a status code of 200, meaning the file was found and could be sent with no problems, and actually sent out 8,240 bytes of data.

The earliest Web servers tended to store the common log format in one file and to record other information separately. Most often, they recorded a separate referer log that recorded URLs providing links to requested pages on the server, and an agent log that recorded the names and versions of browsers making requests.

The more common practice today is to record all this information together in what is called the extended, or combined, log format. For example:

```
staff-pc-118.foo.org - - [11/Nov/2002:18:33:17 –0500]
“GET /info/ill-policy.html HTTP/1.1” 200 14316
“http://www.ourlibrary.org/circulation.html” “Opera
6.05”
```

In addition to the data seen in the common format, the combined format shows the user requested this page by following a link on the circulation.html page, and that the user’s browser is Opera version 6.05.

The URL of the refering page and the browser’s identity are both sent by the browser. Both are optional, and browsers may allow users to send either no value or a false value. The ability of users to fake these headers is an important reason why websites cannot reliably use referer values to determine whether a user is authorized to view a page, and why Web designers should not create different versions of a page based on the user agent string sent by the browser.

These two log formats are used by most Apache servers. IIS servers often use one of these, or may use an IIS-specific format:

```
193.154.31.119, -, 11/11/2002, 18:34:15, W3SVC1,
LIBWINBOX,
192.35.130.181, 3, 120, 12427, 200, 0, GET,
/libraryhours/default.htm
```

The IIS format records the user’s IP address, user name, date, and time. It shows the Windows service that handled the request (W3SVC1), the Windows domain name of the server (LIBWINBOX), and the server’s IP address. Then it records the number of seconds needed to process the request (3), the number of bytes sent by the user to the server (120), the number of bytes sent in response, and the HTTP status. Finally, it records the Windows status code for processing this request (0, meaning the process ran correctly), the HTTP request method, and the URI requested.

In addition to these widely used log formats, both Apache and IIS provide a way to create custom log formats with these or some other elements. Apache provides a syntax for creating a format in its configuration files. This format consists of a percent sign, optionally followed by a list of HTTP status codes to log on or not log on, and a one-character code for the item to log.

For example, a log that recorded the user’s IP address would include “%a”, and a log that included the user’s IP address only when the server returns a 401 or 404 status would include “%401,404a”. A log that recorded the user’s IP address only when the status was not 200 would include “%!200a”. The full set of data points Apache can log is:

a: the remote user’s IP address

A: the server’s IP address

b: bytes sent, not including HTTP headers

B: bytes sent, including HTTP headers

{…}C: the contents of a specific cookie sent. To log a cookie named SessionID use “%{SessionID}C”.

D: the time taken to send the request, in microseconds

{…}e: the contents of a specific environment variable. To keep a log of query strings to scripts, use “%{QUERY_STRING}e”. Many standard environment variables are directly loggable, but Apache has mechanisms for assigning new environment variables to specific requests.

f: filename

h: user’s hostname if available, or IP address if not

H: the request protocol

{…}i: the contents of a specific HTTP header sent by the user. To log the User-agent, use "%{User-agent}e".

l: the remote user's logname from the RFC 921 identd system

m: the request method

{…}n: the contents of a specific note generated by another Apache module

{…}o: the contents of a specific HTTP header sent by the server back to the user

p: the server's port number

P: the operating system's process ID number for the process that handled the request

q: the query string sent to any script, including the question mark

r: the first line of the HTTP request. This designation should always be the request method, requested URI, and HTTP version.

s: the HTTP status code. Where Apache internally redirects a request, this code is for the original request. More common is the log "%>s", which is the code of the final request.

{…}t: the time and date of the request. If no qualifier is present, "%t" uses the time format shown in the Common Log Format example. Other time formats can be specified using the strftime syntax used by some programming languages and the Unix date command.

T: time taken to serve the request, in seconds

u: remote user name, if any

U: the URI requested, not including any query string

v: the name of the server handling the request

V: the name of the server as set in the Apache configuration file

X: the HTTP connection when the response is completed. May be "X" to indicate the connection was aborted before complete, "+" to indicate a keep-alive status for subsequent requests, or "-" for a normally closed connection.

To put these log elements together and create, for example, a log of requests resulting in 404 Not Found errors, first define and name the format in the Apache configuration file and then create the log and identify its location:

```
LogFormat "%404h %t \"%404r\" %404{Referer}i" notfound
CustomLog logs/not_found_log notfound
```

Custom log formats in IIS are slightly less flexible but arguably easier to create. From the dialog box for a website's properties, the Logging section's Properties button leads to a scrollable list of loggable fields. Each can be checked on or off to determine if it will be used in IIS's W3C Extended Log format. The main Web-related options available are:

- Date
- Time
- User's IP address
- User name, if any

- Server's name
- Server's IP address
- Port used for the request
- HTTP request method
- URI requested, not including query strings
- Query string, if any
- HTTP status code
- Number of bytes sent
- Number of bytes received
- Time taken, in seconds
- Protocol used
- User agent
- Cookie
- Referer

The IIS help files cover additional options relating to Windows process management and server performance.

Log rotation

Log files quickly grow in size, and can easily fill all the available disk space. To prevent logs from growing out of control, most Web managers periodically move their servers' log files to a new location and start a new log file from scratch. Both IIS and Apache include scripts for automatically rotating logs this way; IIS's Extended Logging Properties dialog box provides options to start new logs every month, week, day, or hour, or whenever the log file reaches a specified size.

Apache's rotatelogs script automatically rotates logs after a regular period of time or when they reach a specified size. To use it, edit the Apache configuration file to redirect a transaction log to the input of rotatelogs rather than to a filename.

The syntax for this redirection is the Unix convention of using a pipe character to indicate passing output from one process to the input of another:

```
# This will rotate the "combined format" logs/access_log
# file every week, or 604,800 seconds:
CustomLog "|bin/rotatelogs logs/access_log 604800"
combined

# This will do the same whenever the log reaches 50
megabytes
# in size:
CustomLog "|bin/rotatelogs logs/access_log 50M"
```

Analyzing log data

By themselves, log files are of limited use. They provide too much data to assimilate, so few Web managers spend much time reading raw files. Managers more commonly process log files with software designed to identify trends and patterns in the site usage recorded.

Programs such as Webtrends, 123LogAnalyzer, and Analog generate reports that summarize data such as a site's total number of hits, total number of visitors or sessions, site usage by hour of the day or day of the week, or the most referring pages linking into a site. None of these numbers can be taken as an absolute, but they are all useful measures of a site's use when compared with similar measurements over time.

Many sites can make do without installing specialized log analyzers. Most log formats are space-delimited text files that can be imported into desktop database or spreadsheet applications without difficulty. Microsoft Access or even Excel is capable of organizing the data in a small log file and letting users explore patterns in the data.

At the other extreme of specialization, some uses of log data may be too specialized even for dedicated log analysis software. Diagnosing individual problem reports, for example, may require finding only the lines in a log file corresponding to hits from a certain IP address on a certain day requesting files from a certain directory. Such specialized projects are best carried out with custom scripts using tools that process text flexibly.

Although many programs and languages provide tools for finding specific text, few can match Perl's text filtering capabilities. A sample script for parsing log files in Perl is included in Appendix B.