Web Communications: Internet Protocols and HTTP:

1. Internet protocols define the format for all Internet communications between computers. What this means is that for your computer to talk to another computer across the Internet, both must be speaking the same language. For file transfers, FTP (File Transfer Protocol) is used, and for Web communications, HTTP (HyperText Transfer Protocol) is used.

TCP/IP:

1. The Internet is designed to provide communications between its many interconnected nodes. Every computer or device that has an IP address (that set of four numbers connected by dots, such as 64.71.134.49) is a node on the Internet.
2. The Internet is designed to provide communications between its many interconnected nodes. Every computer or device that has an IP address (that set of four numbers connected by dots, such as 64.71.134.49) is a node on the Internet. The main protocol (actually a suite of networking protocols) used to format data for transit is TCP/IP (Transmission Control Protocol over Internet Protocol.).
3. Before the packets of data are sent out across the network, they need to be addressed (they should include a source address and a destination address in the form of an IP address). So a second protocol called HyperText Transfer Protocol (or HTTP) puts an address label on them, so that TCP/IP knows where to direct the information. HTTP is the protocol used by the World Wide Web in the transfer of data from one machine to another—when you see a URL prefixed with http://, you know that the internet protocol being used is HTTP. You can think of TCP/IP as the postal service that does the routing and transfer, although HTTP is the stamp and address on the letter (data) to ensure it gets there.
4. The message passed from the browser to the Web server is known as an HTTP request. When the Web server receives this request (the request is actually a request for a Web page or file), it checks its stores to find the appropriate page. If it finds the page, it parcels up the HTML contained within (using TCP), addresses these packets to the browser (using HTTP), and sends them back across the network. If the Web server cannot find the requested page, it issues a page containing an error message (in this case, the dreaded Error 404: Page Not Found), it parcels up, and dispatches that page to the browser. The message sent from the Web server to the browser is called the HTTP response.

**Web Communications: Internet Protocols and HTTP**

Internet protocols define the format for all Internet communications between computers. What this means is that for your computer to talk to another computer across the Internet, both must be speaking the same language. For file transfers, FTP (File Transfer Protocol) is used, and for Web communications, HTTP (HyperText Transfer Protocol) is used.

The next few sections introduce the Internet protocols and discuss how they facilitate communications on the Web. A good understanding of what's going on between the browser and the server is essential for PHP programming, because within the requests and responses flying back and forth from client to server is a wealth of data you can tap into and use.

**TCP/IP**

The Internet is designed to provide communications between its many interconnected nodes. Every computer or device that has an IP address (that set of four numbers connected by dots, such as 64.71.134.49) is a node on the Internet. The main protocol (actually a suite of networking protocols) used to format data for transit is TCP/IP (Transmission Control Protocol over Internet Protocol.). TCP/IP is simply a method of describing *information packets* (the packages of bits that are individually transmitted across a network) so that they can be sent down your telephone, cable, or T1-line from node to node, until they reach their intended destination.

One advantage of the TCP/IP protocol is that it can reroute information very quickly if a particular node or route is broken or slow. When the user tells the browser to fetch a Web page, the browser parcels up (turns into packets) this instruction using TCP. TCP is a transport protocol, which provides a reliable transmission format for the instruction. It ensures that the entire message is taken apart and packaged up correctly for transmission (and also that it is correctly unpacked and put back together after it reaches its destination).

Before the packets of data are sent out across the network, they need to be addressed (they should include a source address and a destination address in the form of an IP address). So a second protocol called HyperText Transfer Protocol (or HTTP) puts an address label on them, so that TCP/IP knows where to direct the information. HTTP is the protocol used by the World Wide Web in the transfer of data from one machine to another—when you see a URL prefixed with http://, you know that the internet protocol being used is HTTP. You can think of TCP/IP as the postal service that does the routing and transfer, although HTTP is the stamp and address on the letter (data) to ensure it gets there.

The message passed from the browser to the Web server is known as an *HTTP request.* When the Web server receives this request (the request is actually a request for a Web page or file), it checks its stores to find the appropriate page. If it finds the page, it parcels up the HTML contained within (using TCP), addresses these packets to the browser (using HTTP), and sends them back across the network. If the Web server cannot find the requested page, it issues a page containing an error message (in this case, the dreaded Error 404: Page Not Found), it parcels up, and dispatches that page to the browser. The message sent from the Web server to the browser is called the *HTTP response.*

**The HTTP Protocol**

There's quite a bit more technical detail to all of this, let's look more closely at exactly how HTTP works. When a request for a Web page is sent to the server, it contains more than just the desired URL. There is a lot of extra information that is sent as part of the request. This is also true of the response—the server sends extra information back to the browser. You'll explore these different types of information shortly.

A lot of the information that's passed within the HTTP message is generated automatically, and the user doesn't have to deal with it directly, so you don't need to worry about transmitting such information yourself. Although you don't have to worry about creating this information yourself, you should be aware that this extra information is being passed between machines as part of the HTTP request and HTTP response because the PHP script that you write can enable you to have a direct effect on the exact content of this information.

Whether it's a client request or a server response, every HTTP message has the same format, which breaks down into three sections: the request/response line, the HTTP header, and the HTTP body. The content of these three sections is dependent on whether the message is a request or a response, so you'll examine these two cases separately.

**The HTTP Request**

The HTTP request that the browser sends to the Web server contains a request line, a header, and a body. Here's an example of the request line and header:

GET /testpage.htm HTTP/1.1

Accept: \*/\*

Accept-Language: en-us

Connection: Keep-Alive

Host: www.wrox.com

Referer: http://webdev.wrox.co.uk/books/SampleList.php?bookcode=3730

User-Agent: Mozilla (X11; I; Linux 2.0.32 i586)

**The Request Line**

The first line of every HTTP request is the *request line,* which contains three pieces of information:

* An HTTP command known as a method (such as GET and POST)
* The path from the server to the resource that the client is requesting
* The version number of HTTP (such as HTTP 1.1)

Here's an example:

GET /testpage.htm HTTP/1.1

The method is used to tell the server how to handle the request. The following table describes three of the most common methods that appear in this field.

| **Method** | **Description** |
| --- | --- |
| GET | A request for information residing at a particular URL. The majority of HTTP requests made on the Internet are GET requests (when you click a link, a GET request is made). The information required by the request can be anything from an HTML or PHP page, to the output of a JavaScript or PerlScript program, or some other executable. You can send some limited data to the browser, in the form of an extension to the URL |
| HEAD | The same as the GET method, except that it indicates a request for the HTTP header only and no data |
| POST | Indicates that data will be sent to the server as part of the HTTP body (from form fields, for example). This data is then transferred to a data-handling program 011 the Web server |

HTTP supports a number of other methods, including PUT, DELETE, TRACE, CONNECT, and OPTIONS. As a rule, you'll find that these are less common; they are therefore beyond the scope of this discussion. If you want to know more about these, take a look at RFC 2068, which you can find at [www.rfc.net.](http://www.rfc.net.)

**The HTTP Request Header**

The next bit of information sent is the HTTP header. This contains details of what document types the client will accept back from the server, including the type of browser that has requested the page, the date, and general configuration information. The HTTP request's header contains information that falls into three different categories:

* General: Information about either the client or server, but not specific to one or the other
* Entity: Information about the data being sent between the client and server
* Request: Information about the client configuration and different types of acceptable documents

Here's an example of a request header:

Accept: \*/\*

Accept-Language: en-us

Connection: Keep-Alive

Host: www.wrox.com

Referer: http://webdev.wrox.co.uk/books/SampleList.php?bookcode=3730

User-Agent: Mozilla (X11; I; Linux 2.0.32 i586)

As you can see, the HTTP header is composed of a number of lines; each line contains the description of a piece of HTTP header information, and its value.

There are many different lines that can comprise a HTTP header, and most of them are optional, so HTTP has to indicate when it has finished transmitting the header information. To do this, a blank line is used.

**The HTTP Request Body**

If the POST method is used in the HTTP request line, then the HTTP request body contains any data that is being sent to the server—for example, data that the user typed into an HTML form (you'll see examples of this later in the book). Otherwise, the HTTP request body is empty, as it is in the example.

**The HTTP Response**

The HTTP response is sent by the server back to the client browser, and contains a response line, a header, and a body. Here's an example of the response line and header:

HTTP/1.1 200 OK //the status line

Date: Fri, 31st Oct 2003, 18:14:33 GMT //the general header

Server: Apache/1.3.12 (Unix) (SUSE/Linux) PHP/4.0.2 //the response header

Last-modified: Fri, 29th Oct 2003, 14:09:03 GMT //the entity header

//blank line (header

complete)

**The Response Line**

The response line contains only two bits of information:

* The HTTP version number
* An HTTP request code that reports the success or failure of the request

The example response line,

HTTP/1.1 200 OK

returns HTTP status code 200, which represents the message OK, denoting the success of the request, and that the response contains the required page or data from the server. If the response line contains HTTP status code 404 (mentioned earlier in the chapter), then the Web server failed to find the requested resource. Error code values are three-digit numbers, where the first digit indicates the class of the response. There are five classes of response, as shown in the following table.

| **Code class** | **Description** |
| --- | --- |
| 100–199 | Informational; indicate that the request is currently being processed |
| 200–299 | Denote success (that the Web server received and carried out the request successfully) |
| 300–399 | Indicate that the request hasn't been performed because the information required has been moved |
| 400–499 | Denote a client error (that the request was incomplete, incorrect, or impossible) |
| 500–599 | Denote a server error (that the request appeared to be valid, but that the server failed to carry it out) |

**The Response Header**

The HTTP response header is similar to the preceding request header. In the HTTP response, the header information again falls into three types:

* General: contains information about either the client or server, but not specific to one or the other
* Entity: contains information about the data being sent between the client and the server
* Response: contains information about the server sending the response and how it can deal with the response

Once again, the header consists of a number of lines, and uses a blank line to indicate that the header information is complete. Here's an example header, with the name of each line commented at the end:

Date: Fri, 31st Oct 2003, 18:14:33 GMT //the general header

Server: Apache/1.3.12 (Unix) (SUSE/Linux) PHP/4.0.2 //the response header

Last-modified: Fri, 29th Oct 2003, 14:09:03 GMT //the entity header

//blank line (header

complete)

The first line is self-explanatory. On the second line, Server, indicates the type of software the Web server is running. Because this example is requesting a file somewhere on the Web server, the information on the third line refers to the last time the requested page was modified.

The header can contain much more information than this, or different information, depending on what is requested. If you want to know more about the different types of information, you'll find them listed in RFC 2068 (Sections 4.5, 7.1 and 7.2).

**The Response Body**

If the request was successful, the HTTP response body contains the HTML code (together with any script that is to be executed by the browser), ready for the browser's interpretation. If unsuccessful, a failure code is sent.

**Running PHP Scripts via an HTTP Request**

Actually, any client application (not just a browser) that can send an HTTP request to a Web Server can activate and run a PHP program. In fact, it's not a requirement that the file display anything to the user (meaning you don't have to embed your code in a Web page). If a properly formatted HTTP request is sent to the Web server, asking for a file containing PHP code, and the file has the appropriate filename extension, the PHP program will run.

**The Web Server**

If you (or your system administrator) have properly set up the Web server software for the OS it's running on and for PHP, you can expect that HTTP requests for files containing PHP code will be properly handled and that your PHP programs will run.