CHAPTER 47

HTTP Fundamentals

Acknowledgement

Included with permission from *The 4D Web Companion* by David Adams, published by 4D Press. For more information visit:

```
http://www.4dcompanion.com/products/webcompanion/
http://4dpress.4d.com/
```

Introduction

The HyperText Transfer Protocol (HTTP) defines how Web browsers, Web proxies, and Web servers should behave and interact. A little bit of knowledge of HTTP goes a long way when building 4D Web applications. This chapter looks under the surface at what goes into HTTP requests and responses, to help with tasks like these:

- ✓ Debugging Web-database sessions.
- ✓ Understanding how to design and troubleshoot Web applications.
- ✓ Sending special response codes, like standard Web password challenges.
- ✓ Reading and writing header fields, like cookies, user names, and passwords.
- ✓ Writing custom Web serving or Web browsing code with 4D Internet Commands, or ITK.

The *Parsing Requests* chapter, which starts on page 377, describes how browsers and servers pack, transmit, and unpack HTTP messages in detail. The *Debugging HTTP* chapter, which starts on page 347, explains how to capture HTTP data for review.

The ***Web Core*** *demonstration includes code for reading and writing HTTP data.*

See the **Web Processing Steps and 4D Features** *chapter, which starts on page 103, for a summary of which 4D commands and features are designed for each step of the Web-serving process.*

General Features of HTTP

HTTP Is Text Based

Programs can send binary data—like PDF files and applications—through HTTP, but HTTP is itself a plain text protocol, therefore it is easy to read and write. There is generally no need to translate or encode HTTP information in any way. Below is a typical HTTP request for a Web page:

```
GET /bookreviews/webbooks.html HTTP/1.1
```

This is a typical first line of a successful response from a Web server:

```
HTTP/1.1 200 OK
```

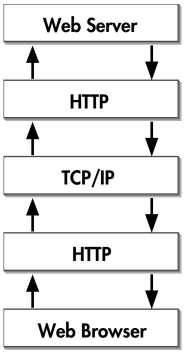
These messages are easy to read because they use descriptive text. The disadvantage of a text-based protocol is that it is not compact. This feature contributes to the overall poor performance of the Web.

HTTP Is a Request/Response Protocol

In HTTP the client, typically a Web browser, sends a request to a server, and the server responds. HTTP defines which requests can be sent, which responses can be returned, and how the request and response messages are formatted. HTTP doesn't define what is requested or sent in a reply, or how the request and response are sent over the network.

HTTP Uses TCP/IP

As far as the HTTP protocol is concerned, request and responses are sent and received magically. Rather than inventing a new low-level protocol, the HTTP specification presumes that TCP/IP is the underlying transportation mechanism. This feature makes HTTP easy to implement in high-level languages, and an in-depth knowledge of TCP/IP is not necessary. Here is a conceptual model of Web communication:



See "Anatomy of HTTP Requests and Responses", starting on page 321 of this chapter, for a detailed discussion of HTTP requests and responses. The ***Web Core*** *database includes code that parses incoming requests into an easy-to-use format.*

HTTP Is Stateless

A protocol can be stateless or stateful. A stateless protocol like HTTP doesn't retain any information about clients between requests. A stateful protocol saves information about clients between requests. 4D Client/Server, for example, remembers current records, current selections, locked records, the current user's name, and operations open in a transaction. These items are properties of the client's current state. HTTP doesn't remember anything between requests. HTTP's statelessness makes it, by nature, poorly suited to building database applications.

Dealing with statelessness is discussed in more detail in the **Managing State** *chapter, which starts on page 405. The* **Contextual Mode Overview** *chapter, which starts on page 195, discussed 4D's attempt to automatically add state management to 4D Web systems.*

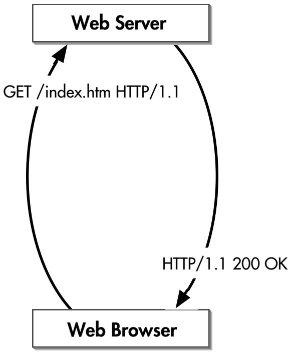
HTTP Is Platform Neutral

HTTP is truly platform-independent. In fact, it's fair to think of HTTP as a platform. Web browsers, proxies, and servers are built for individual platforms, but the protocol itself is the same on all platforms. Web servers exist for every commercial platform in use today, even mainframes. Web browsers exist for PC operating systems, PDAs, and cell phones.

Anatomy of HTTP Requests and Responses

Introduction

Web communication consists of requests from Web clients and responses from Web servers. The most common request is for a page or other document specified in a URL. The Web browser sends a request, and the Web server replies:



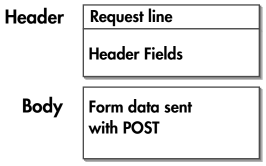
A simple HTTP request-response exchange.

This section explains this process in detail with examples.

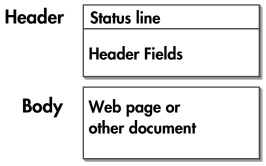
HTTP Message Format

Each HTTP request and response has a header and an optional body. The header includes the request or response and optional header fields. The body of a request may include form field values and uploaded documents. The body of a response may include a Web page, PDF file, program, or other document. HTTP can transfer files of any type, as you will have noticed if you have ever downloaded an application or installer through a Web browser. These diagrams illustrate the general message structure.

HTTP Request

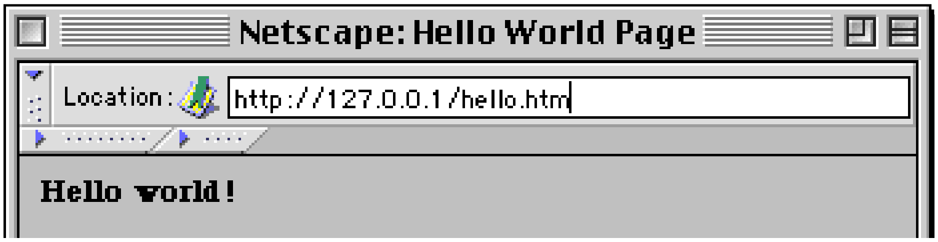


HTTP Response



Example Web Page

It’s easier to explain the details of HTTP headers and responses with examples. Here is a simple Web page shown in a browser:



http://127.0.0.1/hello.htm

Hello world!

Web pages are often slow and disappointing. This one is only disappointing.

This is the HTML for the Web page:

```
<html>
<head>
<title>Hello World Page</title>
</head>

<body>

<b>Hello world!</b>

</body>
</html>
```

The HTML controls what the user sees in the browser. The browser and Web server use HTTP to send the HTML. Here is an example of what the browser sends to request the page shown above:

```
GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.7 (Macintosh; I; PPC)
Host: 198.2.64.127
Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1
```

And this is an example of what the Web server sends to the browser in response:

```
HTTP/1.0 200 OK
Server: 4D_WebStar_D/6.71
Date: Tue, 27 Feb 2001 04:58:21 GMT
Last-Modified: Tue, 27 Feb 2001 04:58:21 GMT
Content-type: text/html;Charset=ISO-8859-1
Content-Length: 102

<html>
<head>
<title>Hello World Page</title>
</head>

<body>

<b>Hello world!</b>

</body>
</html>
```

The HTTP, both in the request and in the response, is invisible to end users. When a browser submits a URL and receives back a Web page, the underlying HTTP is handled internally by the browser and the Web server.

Viewing HTTP

In a native 4D Web application, the HTTP data is not always accessible or easily seen in the 4D Debugger, so other tools are needed. A stream capture program can copy the data moving over HTTP through the operating system. Additionally, a packet capture program can copy all data passing through the computer's network interface (typically a modem or Ethernet card.) See the *Debugging HTTP* chapter, which starts on page 347, for more information on stream and packet capture, and analysis tools.

HTTP Requests

Requesting a Web Page

The simple Web page request header shown above has two parts: the request line and the header fields. This section explains how the data is formatted and its use.

```
Request line  ——  GET /hello.htm HTTP/1.0
                ┌ User-Agent: Mozilla/4.7 (Macintosh; I; PPC)
                │ Host: 198.2.64.127
Header fields ——┤ Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */*
                │ Accept-Encoding: gzip
                │ Accept-Language: en
                └ Accept-Charset: iso-8859-1,*,utf-8
```

The HTTP Request Line

The request line includes three pieces of information:

HTTP request line

Name	Example	Description
Request type	`GET`	Instructs the server how to process the request.
URI	`hello.htm`	Specifies what the browser wants.
HTTP version	`HTTP/1.0`	Indicates the latest version of HTTP the browser supports.

The Web server—4D in this case—reads the request line to determine how to handle the request. Depending on which features a Web browser and Web server implement, there are several possible request types. The table below lists the defined request types and shows whether 4D supports them.

HTTP request types

Type	Description	4D Support
`GET`	A straightforward request for a page or insecure form submission.	Supported.
`POST`	A block of extra data may be sent to the server for processing.	Supported.
`HEAD`	Exactly like `GET` except that no message body is returned. This request type is used by proxies and browsers to check if a document has changed.	Not implemented completely.
`PUT`	A resource is uploaded to the server for storage.	Not implemented.
`DELETE`	A resource is deleted on the server.	Not implemented.

The table below shows the methods used by various 4D features:

HTTP methods supported by 4D commands and features

Request	Type	Supports GET	Supports POST	Notes
Good URL	Link	✓		
Bad URL	Link	✓		
Magic URL	Link	✓		
`4DACTION`	Link	✓		
	Form	✓	✓	
	Image	✓		
	Semi-dynamiccallback	`N/A`	`N/A`	Processed in 4D.

HTTP methods supported by 4D commands and features (continued)

Request	Type	Supports GET	Supports POST	Notes
`4DCGI`	Link	✓		
	Form	✓	✓	
	Image	✓		
`4DMETHOD`	Link	✓		
	Form	✓	✓	
	Image	✓		
`4DHTMLVAR`	Semi-dynamic tag	`N/A`	`N/A`	Processed in 4D.
`4DIF`	Semi-dynamic tag	`N/A`	`N/A`	Processed in 4D.
`4DINCLUDE`	Semi-dynamic tag	`N/A`	`N/A`	Processed in 4D.
`4DLOOP`	Semi-dynamic tag	`N/A`	`N/A`	Processed in 4D.
`4DSCRIPT`	Semi-dynamiccallback	`N/A`	`N/A`	Processed in 4D.
`4DVAR`	Semi-dynamic tag	`N/A`	`N/A`	Processed in 4D.

Semi-dynamic tags are processing instructions embedded within response pages. 4D resolves these tags and replaces their contents with HTML before serving the result page. Therefore, no HTTP request is involved directly with these tags.

In 4D 6.7 and later always use `4DSCRIPT`*, instead of* `4DACTION`*, for semi-dynamic method callbacks.* `4DACTION` *may still be used in links, and image requests, and for form processing.*

HTTP Header Fields

HTTP header fields follow a regular format:

```
Field name+Colon+Space+Field value+Carriage Return+Line Feed
```

The header ends with a blank line:

```
Carriage Return+Line Feed+Carriage Return+Line Feed
```

The extra blank line separates the header and body sections of the HTTP message. Browsers use this break to determine where a result document begins. There are roughly fifty standard HTTP header fields defined in the *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1* specification. Apart from the double carriage return and line feed that end the header, anything can be included in an HTTP header. Cookies, for example, are common header items *not* defined in the HTTP specification.

4D Commands to Read Header Fields

4D 6.7 and later include two commands that read HTTP data directly:

HTTP reading commands

Command	Description
GET HTTP HEADER	Parses incoming HTTP headers into name and value arrays.
GET WEB FORM VARIABLES	Parses incoming form variables out of URL or `POST` section of HTTP header.

The contents of the HTTP request are also present in $2 of the **On Web Authentication** and **On Web Connection** database methods.

See the **Parsing Requests** *chapter, which starts on page 377, for more information about parsing incoming requests.*

HTTP Header Fields Example

The HTTP request can include any number of header fields after the request line, like the ones shown here:

```
User-Agent: Mozilla/4.7 (Macintosh; I; PPC)
Host: 198.2.64.127
Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1
```

Header fields are used to send information which the server and the browser need in order to manage the request. Here is an explanation of the fields shown above:

Field Name	**Field Value**	**Description**
`User-Agent`	`Mozilla/4.7 Macintosh; I; PPC)`	The HTTP specification employs the term "user-agents" instead of browsers because Web clients don't need to be traditional browsers. Mozilla is the internal code name for Netscape browsers. Many non-Netscape browsers identify themselves as Mozilla for compatibility.
`Host`	`198.2.64.127`	The address of the host the request was sent to.
`Accept`	`image/gif, image/x-xbitmap, image/jpeg, image/png, */*`	The document types the browser will accept in response to this request. These are MIME (Multipart Internet Mail Extension) file type codes.
`Accept-Encoding`	`gzip`	The encoding methods the browser will accept in response to this request. Browsers and servers can optimize performance by encoding images and other files in a compressed format.
`Accept-Language`	`en`	The natural languages the browser will accept in response to this request. The value "en" stands for English.
`Accept-Charset`	`iso-8859-1`	The character sets the browser will accept in response to this request. The Latin-1 character set is called ISO-8859-1 within an HTTP request.

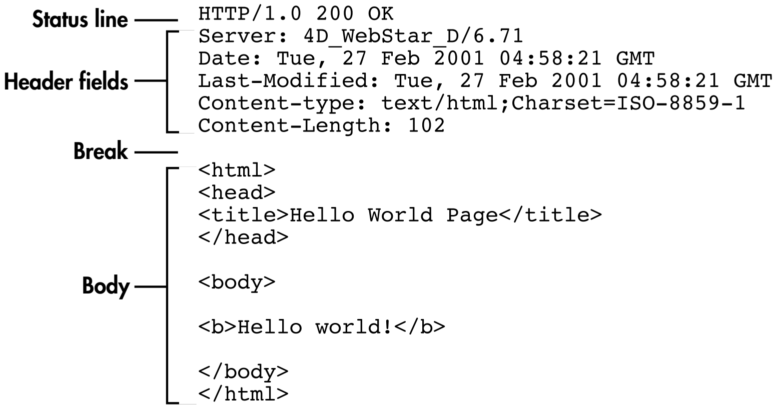
For more information on MIME types, see the **MIME** *chapter, which starts on page 315. For more information on the host tag, see "The Host Header Tag", starting on page 312 of the* **Working with Paths** *chapter.*

Different browsers include a variety of header fields and different contents for the same fields. Don't assume that a specific field is included without testing. The ***Web Core*** database includes routines to test if specific header fields or cookies are included in the current request.

Responding to a Web Page Request

Responding with a Simple Page

The simple Web page request shown above provokes this response:



```
HTTP/1.0 200 OK
Server: 4D_WebStar_D/6.71
Date: Tue, 27 Feb 2001 04:58:21 GMT
Last-Modified: Tue, 27 Feb 2001 04:58:21 GMT
Content-type: text/html;Charset=ISO-8859-1
Content-Length: 102

<html>
<head>
<title>Hello World Page</title>
</head>

<body>

<b>Hello world!</b>

</body>
</html>
```

The response has two parts: the HTTP header and the HTML body. The HTTP header also has two parts: a status line followed by any number of HTTP headers. The HTTP header and HTML response are separated by a break. Below we look at the contents of these parts.

Status Line

All HTTP responses start with a status line that includes the HTTP version used in the response and a status code describing the results of the request. The response code is the most useful piece of information on this line. Web servers use status codes to ask the browser to request a user name and password, send the user to a different page, confirm that the request was understood, and report problems. There are five categories of HTTP response codes:

100 Range: Informational

The request was received by the server, and the browser can continue processing.

200 Range: Success

The request was successfully received, understood, and accepted. The most common responses are `200 OK` and `204 No Content`.

300 Range: Redirection

Further action must be taken in order to complete the request. Redirection is used to send users an updated URL or a private URL after authentication.

400 Range: Client Error

The request contains bad syntax or can't be fulfilled. Common responses in this category include `401 Unauthorized` (a password challenge) and `404 Not Found`.

500 Range: Server Error

The server failed to fulfill an apparently valid request. This situation occurs, for example, when a CGI program crashes or times out.

The table below summarizes the response codes defined in *RFC 2616 Hypertext Transfer Protocol -- HTTP 1.1*:

HTTP 1.1 status codes

Category	Code	Description	Notes
Informational	100	Continue	
	101	Switching Protocols	
Success	200	OK	
	201	Created	
	202	Accepted	
	203	Non-Authoritative Information	
	204	No Content	
	205	Reset Content	
	206	Partial Content	
Redirection	300	Multiple Choices	
	301	Moved Permanently	This header lets you redirect a request from an old address to a new location or to a secure location after custom verification.
	302	Found	
	303	See Other	
	304	Not Modified	
	305	Use Proxy	
	307	Temporary Redirect	
Client Error	400	Bad Request	
	401	Unauthorized	When a Web server, including 4D, sends this header to a browser, the browser asks the user for a user name and password.
	402	Payment Required	
	403	Forbidden	
	404	Not Found	
	405	Method Not Allowed	
	406	Not Acceptable	
	407	Proxy Authentication Required	
	408	Request Time-out	
	409	Conflict	
	410	Gone	
	411	Length Required	
	412	Precondition Failed	
	413	Request Entity Too Large	
	414	Request-URI Too Large	
	415	Unsupported Media Type	
	416	Requested range not satisfiable	
	417	Expectation Failed	
Server Error	500	Internal Server Error	
	501	Not Implemented	
	502	Bad Gateway	
	503	Service Unavailable	
	504	Gateway Time-out	
	505	HTTP Version not supported	

Header Fields

The HTTP response includes any number of header fields. These fields contain information that the browser may need to interpret the response. For example, the `Content-Type` field tells the browser what the response contains, and the `Content-Length` field informs the browser how much data is in the body of the response.

Break

The HTTP response and the response body are divided by a break. Since HTTP is a plain text protocol, the internal delimiters are nothing more than regular text. The break is the character sequence `Carriage Return+Line Feed+Carriage Return+Line Feed`. If you are composing your own HTTP responses and headers—including cookies—make sure not to include extra `Carriage Return+Line Feed` strings.

Body

HTTP can deliver many kinds of documents, including HTML, graphics, and PDF. The exact contents and length of the body are described in the HTTP response header. The Web browser, however, handles storing, displaying, or passing the response body to another program.

4D Commands and HTTP Responses4D includes several commands that read or change HTTP headers and response codes:

4D commands and HTTP responses

Command	Description
SEND HTML BLOB	Sets all required content type and length fields and attaches the BLOB to the message body.
SEND HTML FILE	Sets all required content type and length fields and attaches the document to the message body.
SEND HTML TEXT	Sets all required content type and length fields and attaches the text to the message body.
SEND HTTP REDIRECT	Returns the HTTP `302 Moved temporarily` status code instructing the browser to open a different URL.
On Web Authentication returns **False**	Sends HTTP `401 Unauthorized` status code prompting the browser to ask for a user name and password.
SET HTTP HEADER	Adds HTTP header fields to the response.