Note that the basic authentication protocol does not make use of the Authentication-Info header we showed in Table 12-1.

Base-64 Username/Password Encoding

HTTP basic authentication packs the username and password together (separated by a colon), and encodes them using the base-64 encoding method. If you don't know what base-64 encoding is, don't worry. You don't need to know much about it, and if you are curious, you can read all about it in Appendix E. In a nutshell, base-64 encoding takes a sequence of 8-bit bytes and breaks the sequence of bits into 6-bit chunks. Each 6-bit piece is used to pick a character in a special 64-character alphabet, consisting mostly of letters and numbers.

Figure 12-4 shows an example of using base-64 encoding for basic authentication. Here, the username is "brian-totty" and the password is "Ow!". The browser joins the username and password with a colon, yielding the packed string "brian-totty:Ow!". This string is then base 64–encoded into this mouthful: "YnJpYW4tdG90dHk6T3ch".

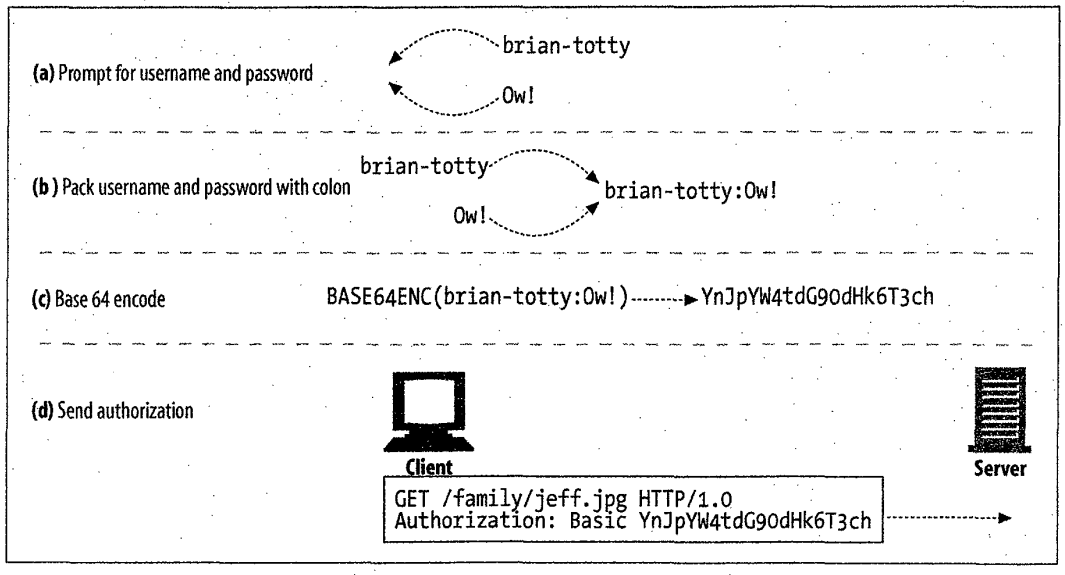


Figure 12-4. Generating a basic Authorization header from username and password

Base-64 encoding was invented to take strings of binary, text, and international character data (which caused problems on some systems) and convert them temporarily into a portable alphabet for transmission. The original strings could then be decoded on the remote end without fear of transmission corruption.

Base-64 encoding can be useful for usernames and passwords that contain international characters or other characters that are illegal in HTTP headers (such as quotation marks, colons, and carriage returns). Also, because base-64 encoding trivially scrambles the username and password, it can help prevent administrators

from accidentally viewing usernames and passwords while administering servers and networks.

Proxy Authentication

Authentication also can be done by intermediary proxy servers. Some organizations use proxy servers to authenticate users before letting them access servers, LANs, or wireless networks. Proxy servers can be a convenient way to provide unified access control across an organization's resources, because access policies can be centrally administered on the proxy server. The first step in this process is to establish the identity via *proxy authentication*.

The steps involved in proxy authentication are identical to that of web server identification. However, the headers and status codes are different. Table 12-3 contrasts the status codes and headers used in web server and proxy authentication.

Table 12-3. Web server versus proxy authentication

Web server	Proxy server
Unauthorized status code: 401	Unauthorized status code: 407
WWW-Authenticate	Proxy-Authenticate
Authorization	Proxy-Authorization
Authentication-Info	Proxy-Authentication-Info

The Security Flaws of Basic Authentication

Basic authentication is simple and convenient, but it is not secure. It should only be used to prevent unintentional access from nonmalicious parties or used in combination with an encryption technology such as SSL.

Consider the following security flaws:

1. Basic authentication sends the username and password across the network in a form that can trivially be decoded. In effect, the secret password is sent in the clear, for anyone to read and capture. Base-64 encoding obscures the username and password, making it less likely that friendly parties will glean passwords by accidental network observation. However, given a base 64–encoded username and password, the decoding can be performed trivially by reversing the encoding process. Decoding can even be done in seconds, by hand, with pencil and paper! Base 64–encoded passwords are effectively sent "in the clear." Assume that motivated third parties will intercept usernames and passwords sent by basic authentication. If this is a concern, send all your HTTP transactions over SSL encrypted channels, or use a more secure authentication protocol, such as digest authentication.

2. Even if the secret password were encoded in a scheme that was more complicated to decode, a third party could still capture the garbled username and password and replay the garbled information to origin servers over and over again to gain access. No effort is made to prevent these replay attacks.
3. Even if basic authentication is used for noncritical applications, such as corporate intranet access control or personalized content, social behavior makes this dangerous. Many users, overwhelmed by a multitude of password-protected services, share usernames and passwords. A clever, malicious party may capture a username and password in the clear from a free Internet email site, for example, and find that the same username and password allow access to critical online banking sites!
4. Basic authentication offers no protection against proxies or intermediaries that act as middlemen, leaving authentication headers intact but modifying the rest of the message to dramatically change the nature of the transaction.
5. Basic authentication is vulnerable to spoofing by counterfeit servers. If a user can be led to believe that he is connecting to a valid host protected by basic authentication when, in fact, he is connecting to a hostile server or gateway, the attacker can request a password, store it for later use, and feign an error.

This all said, basic authentication still is useful for providing convenient personalization or access control to documents in a friendly environment, or where privacy is desired but not absolutely necessary. In this way, basic authentication is used to prevent accidental or casual access by curious users.*

For example, inside a corporation, product management may password-protect future product plans to limit premature distribution. Basic authentication makes it sufficiently inconvenient for friendly parties to access this data.† Likewise, you might password-protect personal photos or private web sites that aren't top-secret or don't contain valuable information, but really aren't anyone else's business either.

Basic authentication can be made secure by combining it with encrypted data transmission (such as SSL) to conceal the username and password from malicious individuals. This is a common technique.

We discuss secure encryption in Chapter 14. The next chapter explains a more sophisticated HTTP authentication protocol, digest authentication, that has stronger security properties than basic authentication.

* Be careful that the username/password in basic authentication is not the same as the password on your more secure systems, or malicious users can use them to break into your secure accounts!

† While not very secure, internal employees of the company usually are unmotivated to maliciously capture passwords. That said, corporate espionage does occur, and vengeful, disgruntled employees do exist, so it is wise to place any data that would be very harmful if maliciously acquired under a stronger security scheme.

For More Information

For more information on basic authentication and LDAP, see:

http://www.ietf.org/rfc/rfc2617.txt
: RFC 2617, "HTTP Authentication: Basic and Digest Access Authentication."

http://www.ietf.org/rfc/rfc2616.txt
: RFC 2616 "Hypertext Transfer Protocol—HTTP/1.1."

CHAPTER 13

Digest Authentication

Basic authentication is convenient and flexible but completely insecure. Usernames and passwords are sent in the clear,* and there is no attempt to protect messages from tampering. The only way to use basic authentication securely is to use it in conjunction with SSL.

Digest authentication was developed as a compatible, more secure alternative to basic authentication. We devote this chapter to the theory and practice of digest authentication. Even though digest authentication is not yet in wide use, the concepts still are important for anyone implementing secure transactions.

The Improvements of Digest Authentication

Digest authentication is an alternate HTTP authentication protocol that tries to fix the most serious flaws of basic authentication. In particular, digest authentication:

- Never sends secret passwords across the network in the clear
- Prevents unscrupulous individuals from capturing and replaying authentication handshakes
- Optionally can guard against tampering with message contents
- Guards against several other common forms of attacks

Digest authentication is not the most secure protocol possible.† Many needs for secure HTTP transactions cannot be met by digest authentication. For those needs, Transport Layer Security (TLS) and Secure HTTP (HTTPS) are more appropriate protocols.

* Usernames and passwords are scrambled using a trivial base-64 encoding, which can be decoded easily. This protects against unintentional accidental viewing but offers no protection against malicious parties.

† For example, compared to public key–based mechanisms, digest authentication does not provide a strong authentication mechanism. Also, digest authentication offers no confidentiality protection beyond protecting the actual password—the rest of the request and response are available to eavesdroppers.

However, digest authentication is significantly stronger than basic authentication, which it was designed to replace. Digest authentication also is stronger than many popular schemes proposed for other Internet services, such as CRAM-MD5, which has been proposed for use with LDAP, POP, and IMAP.

To date, digest authentication has not been widely deployed. However, because of the security risks inherent to basic authentication, the HTTP architects counsel in RFC 2617 that "any service in present use that uses Basic should be switched to Digest as soon as practical."* It is not yet clear how successful this standard will become.

Using Digests to Keep Passwords Secret

The motto of digest authentication is "never send the password across the network." Instead of sending the password, the client sends a "fingerprint" or "digest" of the password, which is an irreversible scrambling of the password. The client and the server both know the secret password, so the server can verify that the digest provided a correct match for the password. Given only the digest, a bad guy has no easy way to find what password it came from, other than going through every password in the universe, trying each one!†

Let's see how this works (this is a simplified version):

- In Figure 13-1a, the client requests a protected document.
- In Figure 13-1b, the server refuses to serve the document until the client authenticates its identity by proving it knows the password. The server issues a challenge to the client, asking for the username and a digested form of the password.
- In Figure 13-1c, the client proves that it knows the password by passing along the digest of the password. The server knows the passwords for all the users,‡ so it can verify that the user knows the password by comparing the client-supplied digest with the server's own internally computed digest. Another party would not easily be able to make up the right digest if it didn't know the password.
- In Figure 13-1d, the server compares the client-provided digest with the server's internally computed digest. If they match, it shows that the client knows the password (or made a really lucky guess!). The digest function can be set to generate so many digits that lucky guesses effectively are impossible. When the server verifies the match, the document is served to the client—all without ever sending the password over the network.

* There has been significant debate about the relevance of digest authentication, given the popularity and widespread adoption of SSL-encrypted HTTP. Time will tell if digest authentication gains the critical mass required.

† There are techniques, such as dictionary attacks, where common passwords are tried first. These cryptanalysis techniques can dramatically ease the process of cracking passwords.

‡ In fact, the server really needs to know only the digests of the passwords.

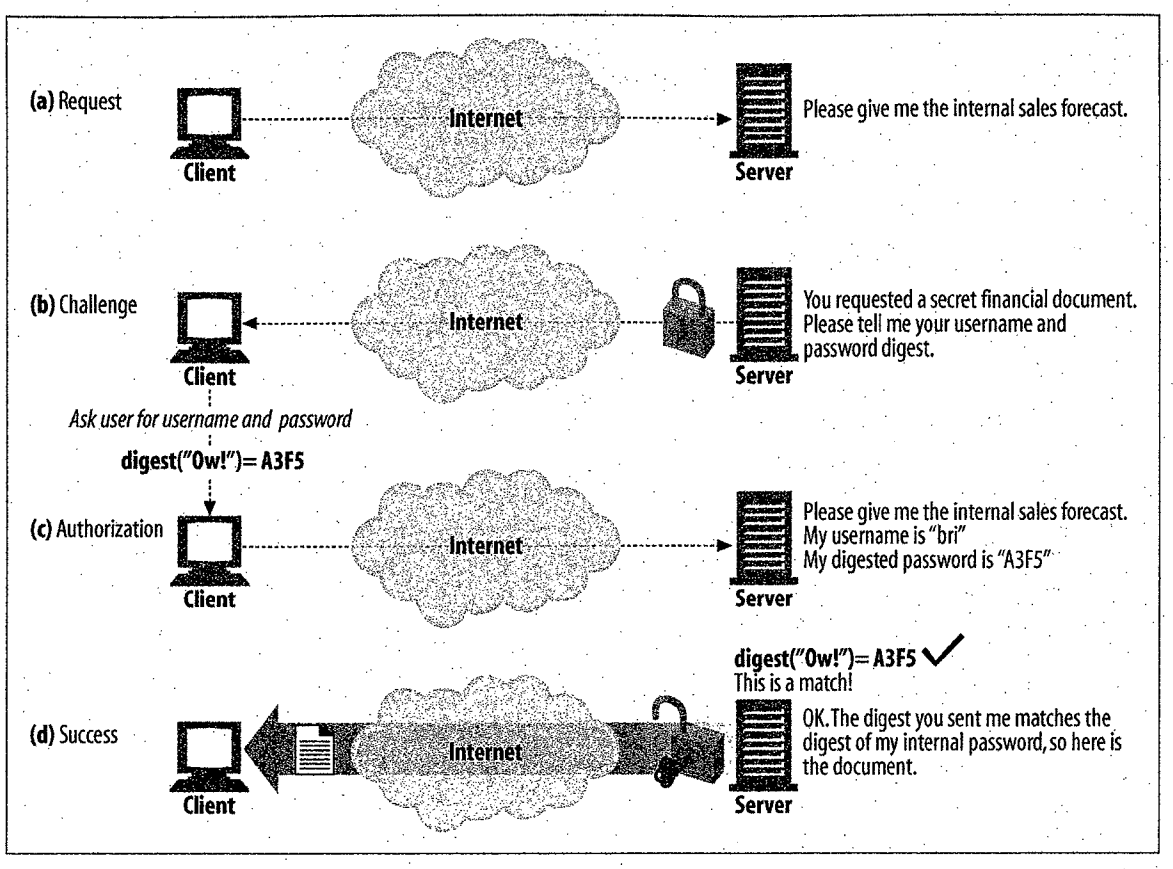


Figure 13-1. Using digests for password-obscured authentication

We'll discuss the particular headers used in digest authentication in more detail in Table 13-8.

One-Way Digests

A digest is a "condensation of a body of information."* Digests act as one-way functions, typically converting an infinite number of possible input values into a finite range of condensations.† One popular digest function, MD5,‡ converts any arbitrary sequence of bytes, of any length, into a 128-bit digest.

128 bits = 2^{128}, or about 1,000,000,000,000,000,000,000,000,000,000,000,000,000 possible distinct condensations.

* Merriam-Webster dictionary, 1998.

† In theory, because we are converting an infinite number of input values into a finite number of output values, it is possible to have two distinct inputs map to the same digest. This is called a *collision*. In practice, the number of potential outputs is so large that the chance of a collision in real life is vanishingly small and, for the purpose of password matching, unimportant.

‡ MD5 stands for "Message Digest #5," one in a series of digest algorithms. The Secure Hash Algorithm (SHA) is another popular digest function.

What is important about these digests is that if you don't know the secret password, you'll have an awfully hard time guessing the correct digest to send to the server. And likewise, if you have the digest, you'll have an awfully hard time figuring out which of the effectively infinite number of input values generated it.

The 128 bits of MD5 output often are written as 32 hexadecimal characters, each character representing 4 bits. Table 13-1 shows a few examples of MD5 digests of sample inputs. Notice how MD5 takes arbitrary inputs and yields a fixed-length digest output.

Table 13-1. MD5 digest examples

Input	MD5 digest
"Hi"	C1A5298F939E87E8F962A5EDFC206918
"bri:0w!"	BEAAA0E34EBDB072F8627C038AB211F8
"3.1415926535897"	475B977E19ECEE70835BC6DF46F4F6DE
"http://www.http-guide.com/index.htm"	C617C0C7D1D05F66F595E22A4B0EAAA5
"WE hold these Truths to be self-evident, that all Men are created equal, that they are endowed by their Creator with certain unalienable Rights, that among these are Life, Liberty and the Pursuit of Happiness—That to secure these Rights, Governments are instituted among Men, deriving their just Powers from the Consent of the Governed, that whenever any Form of Government becomes destructive of these Ends, it is the Right of the People to alter or to abolish it, and to institute new Government, laying its Foundation on such Principles, and organizing its Powers in such Form, as to them shall seem most likely to effect their Safety and Happiness."	66C4EF58DA7CB956BD04233FBB64E0A4

Digest functions sometimes are called cryptographic checksums, one-way hash functions, or fingerprint functions.

Using Nonces to Prevent Replays

One-way digests save us from having to send passwords in the clear. We can just send a digest of the password instead, and rest assured that no malicious party can easily decode the original password from the digest.

Unfortunately, obscured passwords alone do not save us from danger, because a bad guy can capture the digest and replay it over and over again to the server, even though the bad guy doesn't know the password. The digest is just as good as the password.

To prevent such replay attacks, the server can pass along to the client a special token called a *nonce*,* which changes frequently (perhaps every millisecond, or for every

* The word nonce means "the present occasion" or "the time being." In a computer-security sense, the nonce captures a particular point in time and figures that into the security calculations.

authentication). The client appends this nonce token to the password before computing the digest.

Mixing the nonce in with the password causes the digest to change each time the nonce changes. This prevents replay attacks, because the recorded password digest is valid only for a particular nonce value, and without the secret password, the attacker cannot compute the correct digest.

Digest authentication requires the use of nonces, because a trivial replay weakness would make un-nonced digest authentication effectively as weak as basic authentication. Nonces are passed from server to client in the WWW-Authenticate challenge.

The Digest Authentication Handshake

The HTTP digest authentication protocol is an enhanced version of authentication that uses headers similar to those used in basic authentication. Some new options are added to the traditional headers, and one new optional header, Authorization-Info, is added.

The simplified three-phase handshake of digest authentication is depicted in Figure 13-2.

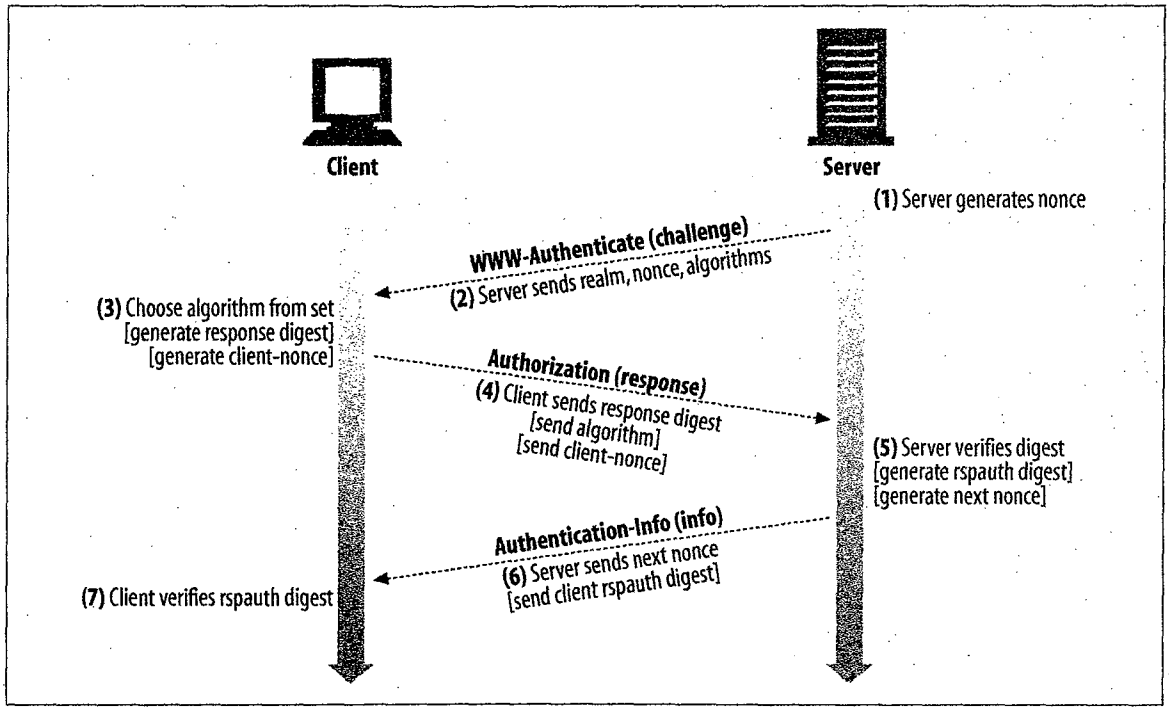


Figure 13-2. Digest authentication handshake

Here's what's happening in Figure 13-2:

- In Step 1, the server computes a nonce value. In Step 2, the server sends the nonce to the client in a WWW-Authenticate challenge message, along with a list of algorithms that the server supports.

- In Step 3, the client selects an algorithm and computes the digest of the secret password and the other data. In Step 4, it sends the digest back to the server in an Authorization message. If the client wants to authenticate the server, it can send a client nonce.
- In Step 5, the server receives the digest, chosen algorithm, and supporting data and computes the same digest that the client did. The server then compares the locally generated digest with the network-transmitted digest and validates that they match. If the client symmetrically challenged the server with a client nonce, a client digest is created. Additionally, the next nonce can be precomputed and handed to the client in advance, so the client can preemptively issue the right digest the next time.

Many of these pieces of information are optional and have defaults. To clarify things, Figure 13-3 compares the messages sent for basic authentication (Figure 13-3a–d) with a simple example of digest authentication (Figure 13-3e–h).

Now let's look a bit more closely at the internal workings of digest authentication.

Digest Calculations

The heart of digest authentication is the one-way digest of the mix of public information, secret information, and a time-limited nonce value. Let's look now at how the digests are computed. The digest calculations generally are straightforward.* Sample source code is provided in Appendix F.

Digest Algorithm Input Data

Digests are computed from three components:

- A pair of functions consisting of a one-way hash function H(*d*) and digest KD(*s*,*d*), where *s* stands for secret and *d* stands for data
- A chunk of data containing security information, including the secret password, called A1
- A chunk of data containing nonsecret attributes of the request message, called A2

The two pieces of data, A1 and A2, are processed by H and KD to yield a digest.

The Algorithms H(d) and KD(s,d)

Digest authentication supports the selection of a variety of digest algorithms. The two algorithms suggested in RFC 2617 are MD5 and MD5-sess (where "sess" stands for session), and the algorithm defaults to MD5 if no other algorithm is specified.

* However, they are made a little more complicated for beginners by the optional compatibility modes of RFC 2617 and by the lack of background material in the specifications. We'll try to help...

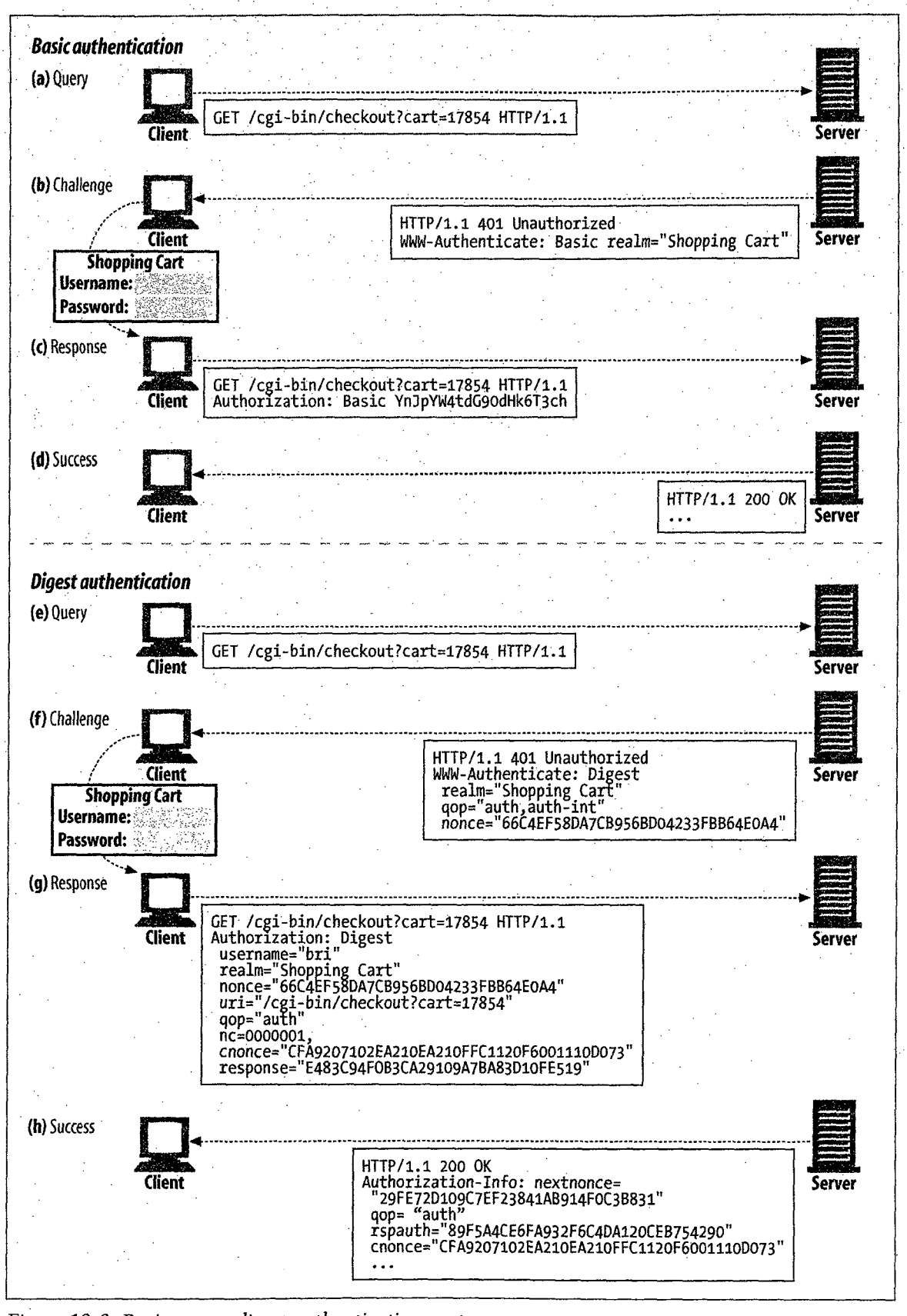


Figure 13-3. Basic versus digest authentication syntax

If either MD5 or MD5-sess is used, the H function computes the MD5 of the data, and the KD digest function computes the MD5 of the colon-joined secret and nonsecret data. In other words:

```
H(<data>) = MD5(<data>)
KD(<secret>,<data>) = H(concatenate(<secret>:<data>))
```

The Security-Related Data (A1)

The chunk of data called A1 is a product of secret and protection information, such as the username, password, protection realm, and nonces. A1 pertains only to security information, not to the underlying message itself. A1 is used along with H, KD, and A2 to compute digests.

RFC 2617 defines two ways of computing A1, depending on the algorithm chosen:

MD5
: One-way hashes are run for every request; A1 is the colon-joined triple of username, realm, and secret password.

MD5-sess
: The hash function is run only once, on the first WWW-Authenticate handshake; the CPU-intensive hash of username, realm, and secret password is done once and prepended to the current nonce and client nonce (cnonce) values.

The definitions of A1 are shown in Table 13-2.

Table 13-2. Definitions for A1 by algorithm

Algorithm	A1
MD5	A1 = <user>:<realm>:<password>
MD5-sess	A1 = MD5(<user>:<realm>:<password>):<nonce>:<cnonce>

The Message-Related Data (A2)

The chunk of data called A2 represents information about the message itself, such as the URL, request method, and message entity body. A2 is used to help protect against method, resource, or message tampering. A2 is used along with H, KD, and A1 to compute digests.

RFC 2617 defines two schemes for A2, depending on the quality of protection (qop) chosen:

- The first scheme involves only the HTTP request method and URL. This is used when qop="auth", which is the default case.
- The second scheme adds in the message entity body to provide a degree of message integrity checking. This is used when qop="auth-int".

The definitions of A2 are shown in Table 13-3.

Table 13-3. Definitions for A2 by algorithm (request digests)

qop	A2
undefined	<request-method>:<uri-directive-value>
auth	<request-method>:<uri-directive-value>
auth-int	<request-method>:<uri-directive-value>:H(<request-entity-body>)

The *request-method* is the HTTP request method. The *uri-directive-value* is the request URI from the request line. This may be "*," an "absoluteURL," or an "abs_path," but it must agree with the request URI. In particular, it must be an absolute URL if the request URI is an absoluteURL.

Overall Digest Algorithm

RFC 2617 defines two ways of computing digests, given H, KD, A1, and A2:

- The first way is intended to be compatible with the older specification RFC 2069, used when the qop option is missing. It computes the digest using the hash of the secret information and the nonced message data.
- The second way is the modern, preferred approach—it includes support for nonce counting and symmetric authentication. This approach is used whenever qop is "auth" or "auth-int". It adds nonce count, qop, and cnonce data to the digest.

The definitions for the resulting digest function are shown in Table 13-4. Notice the resulting digests use H, KD, A1, and A2.

Table 13-4. Old and new digest algorithms

qop	Digest algorithm	Notes
undefined	KD(H(A1), <nonce>:H(A2))	Deprecated
auth or auth-int	KD(H(A1), <nonce>:<nc>:<cnonce>:<qop>:H(A2))	Preferred

It's a bit easy to get lost in all the layers of derivational encapsulation. This is one of the reasons that some readers have difficulty with RFC 2617. To try to make it a bit easier, Table 13-5 expands away the H and KD definitions, and leaves digests in terms of A1 and A2.

Table 13-5. Unfolded digest algorithm cheat sheet

qop	Algorithm	Unfolded algorithm
undefined	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:MD5(A2))

Table 13-5. Unfolded digest algorithm cheat sheet (continued)

qop	Algorithm	Unfolded algorithm
auth	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:<nc>:<cnonce>:<qop>:MD5(A2))
auth-int	<undefined> MD5 MD5-sess	MD5(MD5(A1):<nonce>:<nc>:<cnonce>:<qop>:MD5(A2))

Digest Authentication Session

The client response to a WWW-Authenticate challenge for a protection space starts an authentication session with that protection space (the realm combined with the canonical root of the server being accessed defines a "protection space").

The authentication session lasts until the client receives another WWW-Authenticate challenge from any server in the protection space. A client should remember the username, password, nonce, nonce count, and opaque values associated with an authentication session to use to construct the Authorization header in future requests within that protection space.

When the nonce expires, the server can choose to accept the old Authorization header information, even though the nonce value included may not be fresh. Alternatively, the server may return a 401 response with a new nonce value, causing the client to retry the request; by specifying "stale=true" with this response, the server tells the client to retry with the new nonce without prompting for a new username and password.

Preemptive Authorization

In normal authentication, each request requires a request/challenge cycle before the transaction can be completed. This is depicted in Figure 13-4a.

This request/challenge cycle can be eliminated if the client knows in advance what the next nonce will be, so it can generate the correct Authorization header before the server asks for it. If the client can compute the Authorization header before it is requested, the client can preemptively issue the Authorization header to the server, without first going through a request/challenge. The performance impact is depicted in Figure 13-4b.

Preemptive authorization is trivial (and common) for basic authentication. Browsers commonly maintain client-side databases of usernames and passwords. Once a user authenticates with a site, the browser commonly sends the correct Authorization header for subsequent requests to that URL (see Chapter 12).

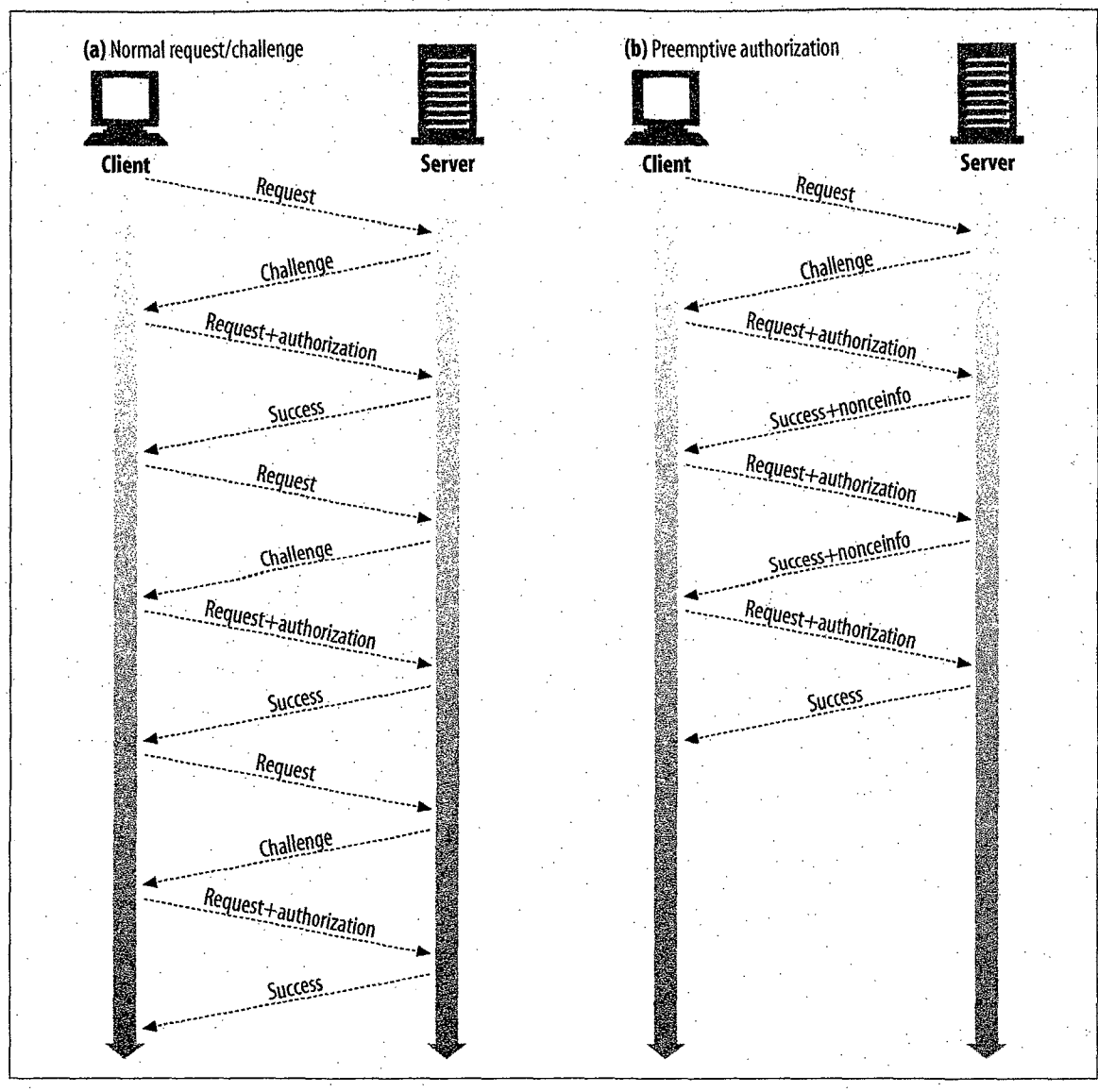


Figure 13-4. Preemptive authorization reduces message count

Preemptive authorization is a bit more complicated for digest authentication, because of the nonce technology intended to foil replay attacks. Because the server generates arbitrary nonces, there isn't always a way for the client to determine what Authorization header to send until it receives a challenge.

Digest authentication offers a few means for preemptive authorization while retaining many of the safety features. Here are three potential ways a client can obtain the correct nonce without waiting for a new WWW-Authenticate challenge:

- Server pre-sends the next nonce in the Authentication-Info success header.
- Server allows the same nonce to be reused for a small window of time.
- Both the client and server use a synchronized, predictable nonce-generation algorithm.

Next nonce pregeneration

The next nonce value can be provided in advance to the client by the Authentication-Info success header. This header is sent along with the 200 OK response from a previous successful authentication.

```
Authentication-Info: nextnonce="<nonce-value>"
```

Given the next nonce, the client can preemptively issue an Authorization header.

While this preemptive authorization avoids a request/challenge cycle (speeding up the transaction), it also effectively nullifies the ability to pipeline multiple requests to the same server, because the next nonce value must be received before the next request can be issued. Because pipelining is expected to be a fundamental technology for latency avoidance, the performance penalty may be large.

Limited nonce reuse

Instead of pregenerating a sequence of nonces, another approach is to allow limited reuse of nonces. For example, a server may allow a nonce to be reused 5 times, or for 10 seconds.

In this case, the client can freely issue requests with the Authorization header, and it can pipeline them, because the nonce is known in advance. When the nonce finally expires, the server is expected to send the client a 401 Unauthorized challenge, with the WWW-Authenticate: stale=true directive set:

```
WWW-Authenticate: Digest
    realm="<realm-value>"
    nonce="<nonce-value>"
    stale=true
```

Reusing nonces does reduce security, because it makes it easier for an attacker to succeed at replay attacks. Because the lifetime of nonce reuse is controllable, from strictly no reuse to potentially long reuse, trade-offs can be made between windows of vulnerability and performance.

Additionally, other features can be employed to make replay attacks more difficult, including incrementing counters and IP address tests. However, while making attacks more inconvenient, these techniques do not eliminate the vulnerability.

Synchronized nonce generation

It is possible to employ time-synchronized nonce-generation algorithms, where both the client and the server can generate a sequence of identical nonces, based on a shared secret key, that a third party cannot easily predict (such as secure ID cards).

These algorithms are beyond the scope of the digest authentication specification.

Nonce Selection

The contents of the nonce are opaque and implementation-dependent. However, the quality of performance, security, and convenience depends on a smart choice.

RFC 2617 suggests this hypothetical nonce formulation:

```
BASE64(time-stamp H(time-stamp ":" ETag ":" private-key))
```

where time-stamp is a server-generated time or other nonrepeating value, ETag is the value of the HTTP ETag header associated with the requested entity, and private-key is data known only to the server.

With a nonce of this form, a server will recalculate the hash portion after receiving the client authentication header and reject the request if it does not match the nonce from that header or if the time-stamp value is not recent enough. In this way, the server can limit the duration of the nonce's validity.

The inclusion of the ETag prevents a replay request for an updated version of the resource. (Note that including the IP address of the client in the nonce would appear to offer the server the ability to limit the reuse of the nonce to the same client that originally got it. However, that would break proxy farms, in which requests from a single user often go through different proxies. Also, IP address spoofing is not that hard.)

An implementation might choose not to accept a previously used nonce or digest, to protect against replay attacks. Or, an implementation might choose to use one-time nonces or digests for POST or PUT requests and time-stamps for GET requests.

Refer to "Security Considerations" for practical security considerations that affect nonce selection.

Symmetric Authentication

RFC 2617 extends digest authentication to allow the client to authenticate the server. It does this by providing a client nonce value, to which the server generates a correct response digest based on correct knowledge of the shared secret information. The server then returns this digest to the client in the Authorization-Info header.

This symmetric authentication is standard as of RFC 2617. It is optional for backward compatibility with the older RFC 2069 standard, but, because it provides important security enhancements, all modern clients and servers are strongly recommended to implement all of RFC 2617's features. In particular, symmetric authentication is required to be performed whenever a qop directive is present and required not to be performed when the qop directive is missing.

The response digest is calculated like the request digest, except that the message body information (A2) is different, because there is no method in a response, and the message entity data is different. The methods of computation of A2 for request and response digests are compared in Tables 13-6 and 13-7.

Table 13-6. Definitions for A2 by algorithm (request digests)

qop	A2
undefined	<request-method>:<uri-directive-value>
auth	<request-method>:<uri-directive-value>
auth-int	<request-method>:<uri-directive-value>:H(<request-entity-body>)

Table 13-7. Definitions for A2 by algorithm (response digests)

qop	A2
undefined	:<uri-directive-value>
auth	:<uri-directive-value>
auth-int	:<uri-directive-value>:H(<response-entity-body>)

The cnonce value and nc value must be the ones for the client request to which this message is the response. The response auth, cnonce, and nonce count directives must be present if qop="auth" or qop="auth-int" is specified.

Quality of Protection Enhancements

The qop field may be present in all three digest headers: WWW-Authenticate, Authorization, and Authentication-Info.

The qop field lets clients and servers negotiate for different types and qualities of protection. For example, some transactions may want to sanity check the integrity of message bodies, even if that slows down transmission significantly.

The server first exports a comma-separated list of qop options in the WWW-Authenticate header. The client then selects one of the options that it supports and that meets its needs and passes it back to the server in its Authorization qop field.

Use of qop is optional, but only for backward compatibility with the older RFC 2069 specification. The qop option should be supported by all modern digest implementations.

RFC 2617 defines two initial quality of protection values: "auth," indicating authentication, and "auth-int," indicating authentication with message integrity protection. Other qop options are expected in the future.

Message Integrity Protection

If integrity protection is applied (qop="auth-int"), H (the entity body) is the hash of the entity body, not the message body. It is computed before any transfer encoding is applied by the sender and after it has been removed by the recipient. Note that this includes multipart boundaries and embedded headers in each part of any multipart content type.

Digest Authentication Headers

Both the basic and digest authentication protocols contain an authorization challenge, carried by the WWW-Authenticate header, and an authorization response, carried by the Authorization header. Digest authentication adds an optional Authorization-Info header, which is sent after successful authentication, to complete a three-phase handshake and pass along the next nonce to use. The basic and digest authentication headers are shown in Table 13-8.

Table 13-8. HTTP authentication headers

Phase	Basic	Digest
Challenge	`WWW-Authenticate: Basic realm="<realm-value>"`	`WWW-Authenticate: Digest realm="<realm-value>" nonce="<nonce-value>" [domain="<list-of-URIs>"] [opaque="<opaque-token-value>"] [stale=<true-or-false>] [algorithm=<digest-algorithm>] [qop="<list-of-qop-values>"] [<extension-directive>]`
Response	`Authorization: Basic <base64(user:pass)>`	`Authorization: Digest username="<username>" realm="<realm-value>" nonce="<nonce-value>" uri=<request-uri> response="<32-hex-digit-digest>" [algorithm=<digest-algorithm>] [opaque="<opaque-token-value>"] [cnonce="<nonce-value>"] [qop=<qop-value>] [nc=<8-hex-digit-nonce-count>] [<extension-directive>]`
Info	n/a	`Authentication-Info: nextnonce="<nonce-value>" [qop="<list-of-qop-values>"] [rspauth="<hex-digest>"] [cnonce="<nonce-value>"] [nc=<8-hex-digit-nonce-count>]`

The digest authentication headers are quite a bit more complicated. They are described in detail in Appendix F.

Practical Considerations

There are several things you need to consider when working with digest authentication. This section discusses some of these issues.

Multiple Challenges

A server can issue multiple challenges for a resource. For example, if a server does not know the capabilities of a client, it may provide both basic and digest authentication challenges. When faced with multiple challenges, the client must choose to answer with the strongest authentication mechanism that it supports.

User agents must take special care in parsing the WWW-Authenticate or Proxy-Authenticate header field value if it contains more than one challenge or if more than one WWW-Authenticate header field is provided, as a challenge may itself contain a comma-separated list of authentication parameters. Note that many browsers recognize only basic authentication and require that it be the first authentication mechanism presented.

There are obvious "weakest link" security concerns when providing a spectrum of authentication options. Servers should include basic authentication only if it is minimally acceptable, and administrators should caution users about the dangers of sharing the same password across systems when different levels of security are being employed.

Error Handling

In digest authentication, if a directive or its value is improper, or if a required directive is missing, the proper response is 400 Bad Request.

If a request's digest does not match, a login failure should be logged. Repeated failures from a client may indicate an attacker attempting to guess passwords.

The authenticating server must assure that the resource designated by the "uri" directive is the same as the resource specified in the request line; if they are different, the server should return a 400 Bad Request error. (As this may be a symptom of an attack, server designers may want to consider logging such errors.) Duplicating information from the request URL in this field deals with the possibility that an intermediate proxy may alter the client's request line. This altered (but, presumably, semantically equivalent) request would not result in the same digest as that calculated by the client.

Protection Spaces

The realm value, in combination with the canonical root URL of the server being accessed, defines the protection space.

Realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value is a string, generally assigned by the origin server, which may have additional semantics specific to the authentication scheme. Note that there may be multiple challenges with the same authorization scheme but different realms.

The protection space determines the domain over which credentials can be automatically applied. If a prior request has been authorized, the same credentials may be reused for all other requests within that protection space for a period of time determined by the authentication scheme, parameters, and/or user preference. Unless otherwise defined by the authentication scheme, a single protection space cannot extend outside the scope of its server.

The specific calculation of protection space depends on the authentication mechanism:

- In basic authentication, clients assume that all paths at or below the request URI are within the same protection space as the current challenge. A client can preemptively authorize for resources in this space without waiting for another challenge from the server.
- In digest authentication, the challenge's WWW-Authenticate: domain field more precisely defines the protection space. The domain field is a quoted, space-separated list of URIs. All the URIs in the domain list, and all URIs logically beneath these prefixes, are assumed to be in the same protection space. If the domain field is missing or empty, all URIs on the challenging server are in the protection space.

Rewriting URIs

Proxies may rewrite URIs in ways that change the URI syntax but not the actual resource being described. For example:

- Hostnames may be normalized or replaced with IP addresses.
- Embedded characters may be replaced with "%" escape forms.
- Additional attributes of a type that doesn't affect the resource fetched from the particular origin server may be appended or inserted into the URI.

Because URIs can be changed by proxies, and because digest authentication sanity checks the integrity of the URI value, the digest authentication will break if any of these changes are made. See "The Message-Related Data (A2)" for more information.

Caches

When a shared cache receives a request containing an Authorization header and a response from relaying that request, it must not return that response as a reply to any other request, unless one of two Cache-Control directives was present in the response:

- If the original response included the "must-revalidate" Cache-Control directive, the cache may use the entity of that response in replying to a subsequent request. However, it must first revalidate it with the origin server, using the request headers from the new request, so the origin server can authenticate the new request.
- If the original response included the "public" Cache-Control directive, the response entity may be returned in reply to any subsequent request.

Security Considerations

RFC 2617 does an admirable job of summarizing some of the security risks inherent in HTTP authentication schemes. This section describes some of these risks.

Header Tampering

To provide a foolproof system against header tampering, you need either end-to-end encryption or a digital signature of the headers—preferably a combination of both! Digest authentication is focused on providing a tamper-proof authentication scheme, but it does not necessarily extend that protection to the data. The only headers that have some level of protection are WWW-Authenticate and Authorization.

Replay Attacks

A replay attack, in the current context, is when someone uses a set of snooped authentication credentials from a given transaction for another transaction. While this problem is an issue with GET requests, it is vital that a foolproof method for avoiding replay attacks be available for POST and PUT requests. The ability to successfully replay previously used credentials while transporting form data could cause security nightmares.

Thus, in order for a server to accept "replayed" credentials, the nonce values must be repeated. One of the ways to mitigate this problem is to have the server generate a nonce containing a digest of the client's IP address, a time-stamp, the resource ETag, and a private server key (as recommended earlier). In such a scenario, the combination of an IP address and a short timeout value may provide a huge hurdle for the attacker.

However, this solution has a major drawback. As we discussed earlier, using the client's IP address in creating a nonce breaks transmission through proxy farms, in which requests from a single user may go through different proxies. Also, IP spoofing is not too difficult.

One way to completely avoid replay attacks is to use a unique nonce value for every transaction. In this implementation, for each transaction, the server issues a unique nonce along with a timeout value. The issued nonce value is valid only for the given transaction, and only for the duration of the timeout value. This accounting may increase the load on servers; however, the increase should be miniscule.

Multiple Authentication Mechanisms

When a server supports multiple authentication schemes (such as basic and digest), it usually provides the choice in WWW-Authenticate headers. Because the client is

not required to opt for the strongest authentication mechanism, the strength of the resulting authentication is only as good as that of the weakest of the authentication schemes.

The obvious ways to avoid this problem is to have the clients always choose the strongest authentication scheme available. If this is not practical (as most of us do use commercially available clients), the only other option is to use a proxy server to retain only the strongest authentication scheme. However, such an approach is feasible only in a domain in which all of the clients are known to be able to support the chosen authentication scheme—e.g., a corporate network.

Dictionary Attacks

Dictionary attacks are typical password-guessing attacks. A malicious user can eavesdrop on a transaction and use a standard password-guessing program against nonce/response pairs. If the users are using relatively simple passwords and the servers are using simplistic nonces, it is quite possible to find a match. If there is no password aging policy, given enough time and the one-time cost of cracking the passwords, it is easy to collect enough passwords to do some real damage.

There really is no good way to solve this problem, other than using relatively complex passwords that are hard to crack and a good password aging policy.

Hostile Proxies and Man-in-the-Middle Attacks

Much Internet traffic today goes through a proxy at one point or another. With the advent of redirection techniques and intercepting proxies, a user may not even realize that his request is going through a proxy. If one of those proxies is hostile or compromised, it could leave the client vulnerable to a man-in-the-middle attack.

Such an attack could be in the form of eavesdropping, or altering available authentication schemes by removing all of the offered choices and replacing them with the weakest authentication scheme (such as basic authentication).

One of the ways to compromise a trusted proxy is though its extension interfaces. Proxies sometimes provide sophisticated programming interfaces, and with such proxies it may be feasible to write an extension (i.e., plug-in) to intercept and modify the traffic. However, the data-center security and security offered by proxies themselves make the possibility of man-in-the-middle attacks via rogue plug-ins quite remote.

There is no good way to fix this problem. Possible solutions include clients providing visual cues regarding the authentication strength, configuring clients to always use the strongest possible authentication, etc., but even when using the strongest possible authentication scheme, clients still are vulnerable to eavesdropping. The only foolproof way to guard against these attacks is by using SSL.

Chosen Plaintext Attacks

Clients using digest authentication use a nonce supplied by the server to generate the response. However, if there is a compromised or malicious proxy in the middle intercepting the traffic (or a malicious origin server), it can easily supply a nonce for response computation by the client. Using the known key for computing the response may make the cryptanalysis of the response easier. This is called a *chosen plaintext attack*. There are a few variants of chosen plaintext attacks:

Precomputed dictionary attacks

This is a combination of a dictionary attack and a chosen plaintext attack. First, the attacking server generates a set of responses, using a predetermined nonce and common password variations, and creates a dictionary. Once a sizeable dictionary is available, the attacking server/proxy can complete the interdiction of the traffic and start sending predetermined nonces to the clients. When it gets a response from a client, the attacker searches the generated dictionary for matches. If a there is a match, the attacker has the password for that particular user.

Batched brute-force attacks

The difference in a batched brute-force attack is in the computation of the password. Instead of trying to match a precomputed digest, a set of machines goes to work on enumerating all of the possible passwords for a given space. As the machines get faster, the brute-force attack becomes more and more viable.

In general, the threat posed by these attacks is easily countered. One way to prevent them is to configure clients to use the optional cnonce directive, so that the response is generated at the client's discretion, not using the nonce supplied by the server (which could be compromised by the attacker). This, combined with policies enforcing reasonably strong passwords and a good password aging mechanism, can mitigate the threat of chosen plaintext attacks completely.

Storing Passwords

The digest authentication mechanism compares the user response to what is stored internally by the server—usually, usernames and H(A1) tuples, where H(A1) is derived from the digest of username, realm, and password.

Unlike with a traditional password file on a Unix box, if a digest authentication password file is compromised, all of the documents in the realm immediately are available to the attacker; there is no need for a decrypting step.

Some of the ways to mitigate this problem are to:

- Protect the password file as though it contained clear-text passwords.
- Make sure the realm name is unique among all the realms, so that if a password file is compromised, the damage is localized to a particular realm. A fully qualified realm name with host and domain included should satisfy this requirement.

While digest authentication provides a much more robust and secure solution than basic authentication, it still does not provide any protection for security of the content—a truly secure transaction is feasible only through SSL, which we describe in the next chapter.

For More Information

For more information on authentication, see:

http://www.ietf.org/rfc/rfc2617.txt
: RFC 2617, "HTTP Authentication: Basic and Digest Access Authentication."

CHAPTER 14
Secure HTTP

The previous three chapters reviewed features of HTTP that help identify and authenticate users. These techniques work well in a friendly community, but they aren't strong enough to protect important transactions from a community of motivated and hostile adversaries.

This chapter presents a more complicated and aggressive technology to secure HTTP transactions from eavesdropping and tampering, using digital cryptography.

Making HTTP Safe

People use web transactions for serious things. Without strong security, people wouldn't feel comfortable doing online shopping and banking. Without being able to restrict access, companies couldn't place important documents on web servers. The Web requires a secure form of HTTP.

The previous chapters talked about some lightweight ways of providing authentication (basic and digest authentication) and message integrity (digest qop="auth-int"). These schemes are good for many purposes, but they may not be strong enough for large purchases, bank transactions, or access to confidential data. For these more serious transactions, we combine HTTP with digital encryption technology.

A secure version of HTTP needs to be efficient, portable, easy to administer, and adaptable to the changing world. It also has to meet societal and governmental requirements. We need a technology for HTTP security that provides:

- Server authentication (clients know they're talking to the real server, not a phony)
- Client authentication (servers know they're talking to the real user, not a phony)
- Integrity (clients and servers are safe from their data being changed)
- Encryption (clients and servers talk privately without fear of eavesdropping)
- Efficiency (an algorithm fast enough for inexpensive clients and servers to use)
- Ubiquity (protocols are supported by virtually all clients and servers)

- Administrative scalability (instant secure communication for anyone, anywhere)
- Adaptability (supports the best known security methods of the day)
- Social viability (meets the cultural and political needs of the society)

HTTPS

HTTPS is the most popular secure form of HTTP. It was pioneered by Netscape Communications Corporation and is supported by all major browsers and servers.

You can tell if a web page was accessed through HTTPS instead of HTTP, because the URL will start with the scheme *https://* instead of *http://* (some browsers also display iconic security cues, as shown in Figure 14-1).

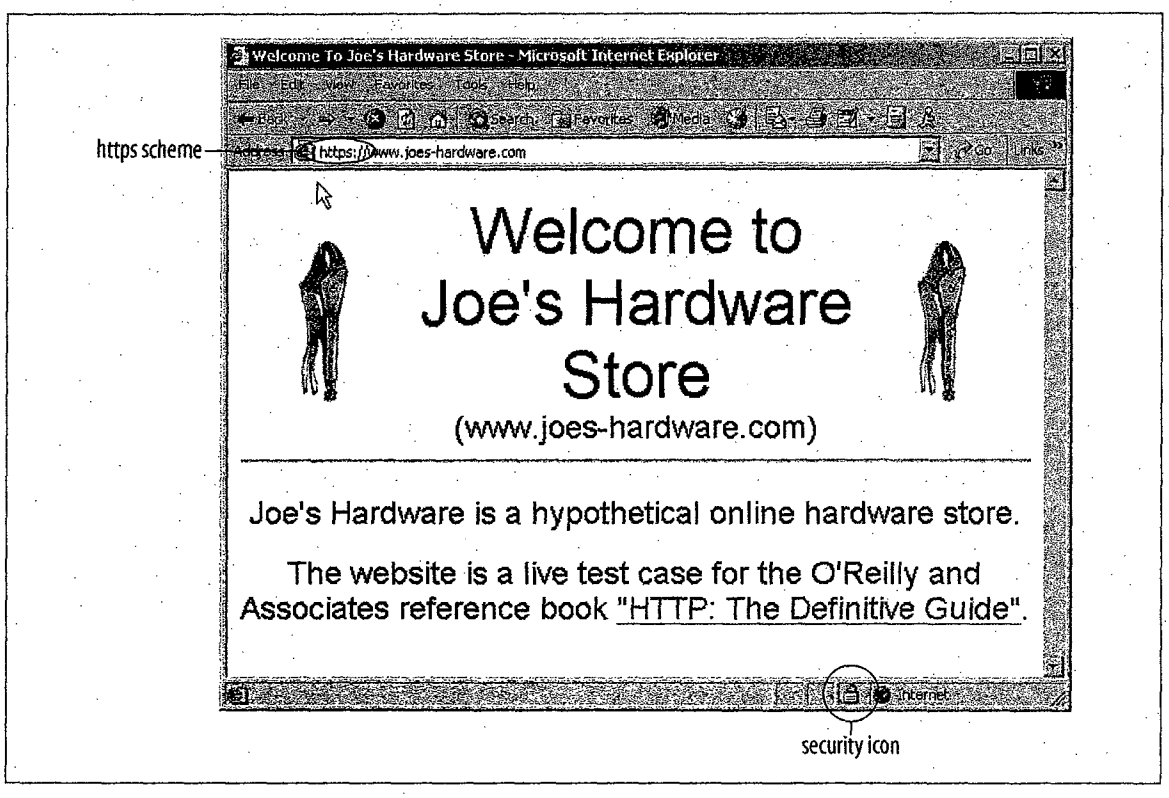


Figure 14-1. Browsing secure web sites

When using HTTPS, all the HTTP request and response data is encrypted before being sent across the network. HTTPS works by providing a transport-level cryptographic security layer—using either the Secure Sockets Layer (SSL) or its successor, Transport Layer Security (TLS)—underneath HTTP (Figure 14-2). Because SSL and TLS are so similar, in this book we use the term "SSL" loosely to represent both SSL and TLS.

Because most of the hard encoding and decoding work happens in the SSL libraries, web clients and servers don't need to change much of their protocol processing logic

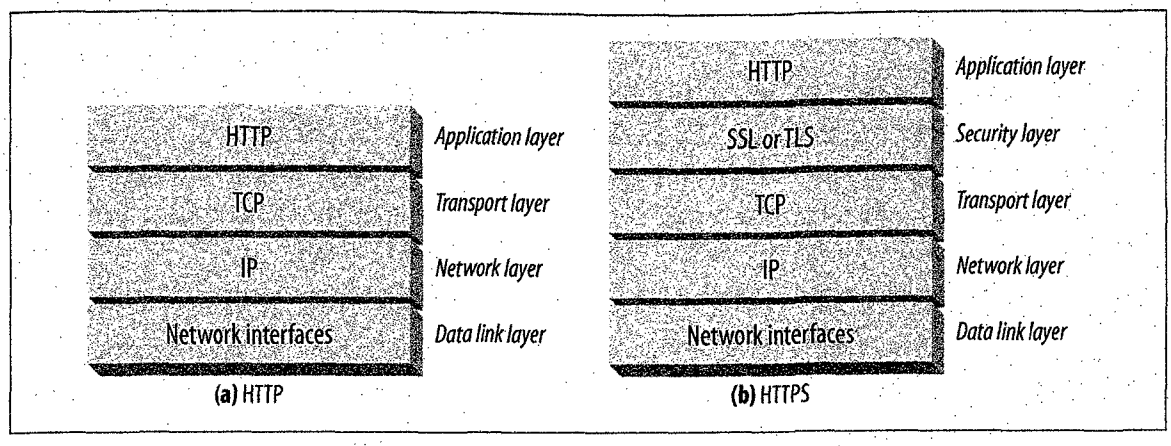


Figure 14-2. HTTPS is HTTP layered over a security layer, layered over TCP

to use secure HTTP. For the most part, they simply need to replace TCP input/output calls with SSL calls and add a few other calls to configure and manage the security information.

Digital Cryptography

Before we talk in detail about HTTPS, we need to provide a little background about the cryptographic encoding techniques used by SSL and HTTPS. In the next few sections, we'll give a speedy primer of the essentials of digital cryptography. If you already are familiar with the technology and terminology of digital cryptography, feel free to jump ahead to "HTTPS: The Details."

In this digital cryptography primer, we'll talk about:

Ciphers
: Algorithms for encoding text to make it unreadable to voyeurs

Keys
: Numeric parameters that change the behavior of ciphers

Symmetric-key cryptosystems
: Algorithms that use the same key for encoding and decoding

Asymmetric-key cryptosystems
: Algorithms that use different keys for encoding and decoding

Public-key cryptography
: A system making it easy for millions of computers to send secret messages

Digital signatures
: Checksums that verify that a message has not been forged or tampered with

Digital certificates
: Identifying information, verified and signed by a trusted organization

The Art and Science of Secret Coding

Cryptography is the art and science of encoding and decoding messages. People have used cryptographic methods to send secret messages for thousands of years. However, cryptography can do more than just encrypt messages to prevent reading by nosy folks; it also can be used to prevent tampering with messages. Cryptography even can be used to prove that you indeed authored a message or transaction, just like your handwritten signature on a check or an embossed wax seal on an envelope.

Ciphers

Cryptography is based on secret codes called *ciphers*. A cipher is a coding scheme—a particular way to encode a message and an accompanying way to decode the secret later. The original message, before it is encoded, often is called *plaintext* or *cleartext*. The coded message, after the cipher is applied, often is called *ciphertext*. Figure 14-3 shows a simple example.

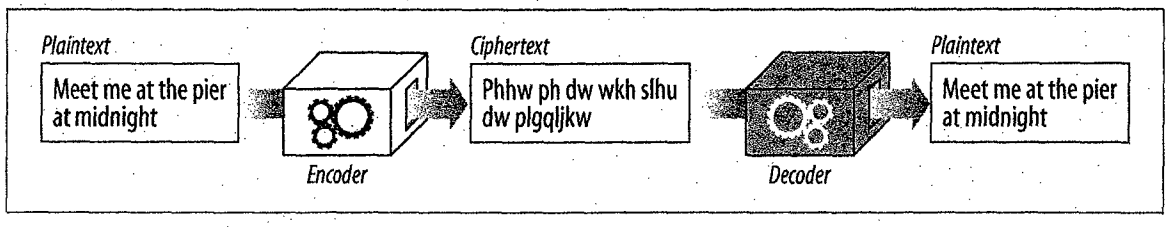


Figure 14-3. Plaintext and ciphertext

Ciphers have been used to generate secret messages for thousands of years. Legend has it that Julius Caesar used a three-character rotation cipher, where each character in the message is replaced with a character three alphabetic positions forward. In our modern alphabet, "A" would be replaced by "D," "B" would be replaced by "E," and so on.

For example, in Figure 14-4, the message "meet me at the pier at midnight" encodes into the ciphertext "phhw ph dw wkh slhu dw plgqljkw" using the rot3 (rotate by 3 characters) cipher.* The ciphertext can be decrypted back to the original plaintext message by applying the inverse coding, rotating –3 characters in the alphabet.

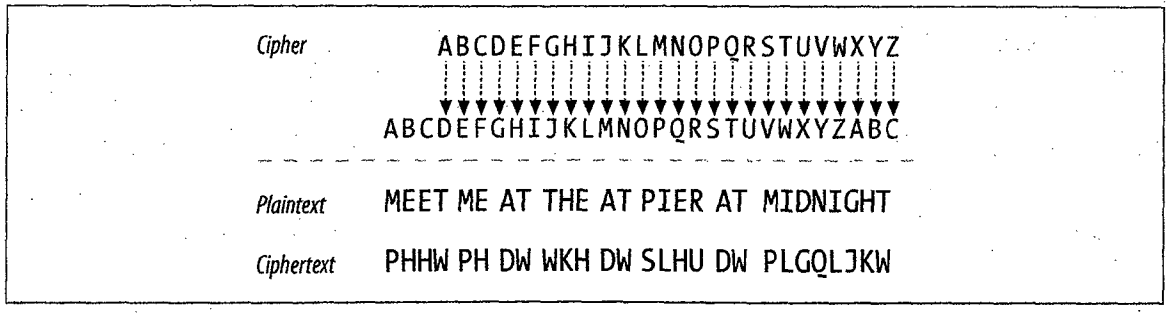


Figure 14-4. Rotate-by-3 cipher example

* For simplicity of example, we aren't rotating punctuation or whitespace, but you could.

Cipher Machines

Ciphers began as relatively simple algorithms, because human beings needed to do the encoding and decoding themselves. Because the ciphers were simple, people could work the codes using pencil and paper and code books. However, it also was possible for clever people to "crack" the codes fairly easily.

As technology advanced, people started making machines that could quickly and accurately encode and decode messages using much more complicated ciphers. Instead of just doing simple rotations, these cipher machines could substitute characters, transpose the order of characters, and slice and dice messages to make codes much harder to crack.*

Keyed Ciphers

Because code algorithms and machines could fall into enemy hands, most machines had dials that could be set to a large number of different values that changed how the cipher worked. Even if the machine was stolen, without the right dial settings (key values) the decoder wouldn't work.†

These cipher parameters were called *keys*. You needed to enter the right key into the cipher machine to get the decoding process to work correctly. Cipher keys make a single cipher machine act like a set of many virtual cipher machines, each of which behaves differently because they have different key values.

Figure 14-5 illustrates an example of keyed ciphers. The cipher algorithm is the trivial "rotate-by-N" cipher. The value of N is controlled by the key. The same input message, "meet me at the pier at midnight," passed through the same encoding machine, generates different outputs depending on the value of the key. Today, virtually all cipher algorithms use keys.

Digital Ciphers

With the advent of digital computation, two major advances occurred:

- Complicated encoding and decoding algorithms became possible, freed from the speed and function limitations of mechanical machinery.

* Perhaps the most famous mechanical code machine was the World War II German Enigma code machine. Despite the complexity of the Enigma cipher, Alan Turing and colleagues were able to crack the Enigma codes in the early 1940s, using the earliest digital computers.

† In reality, having the logic of the machine in your possession can sometimes help you to crack the code, because the machine logic may point to patterns that you can exploit. Modern cryptographic algorithms usually are designed so that even if the algorithm is publicly known, it's difficult to come up with any patterns that will help evildoers crack the code. In fact, many of the strongest ciphers in common use have their source code available in the public domain, for all to see and study!

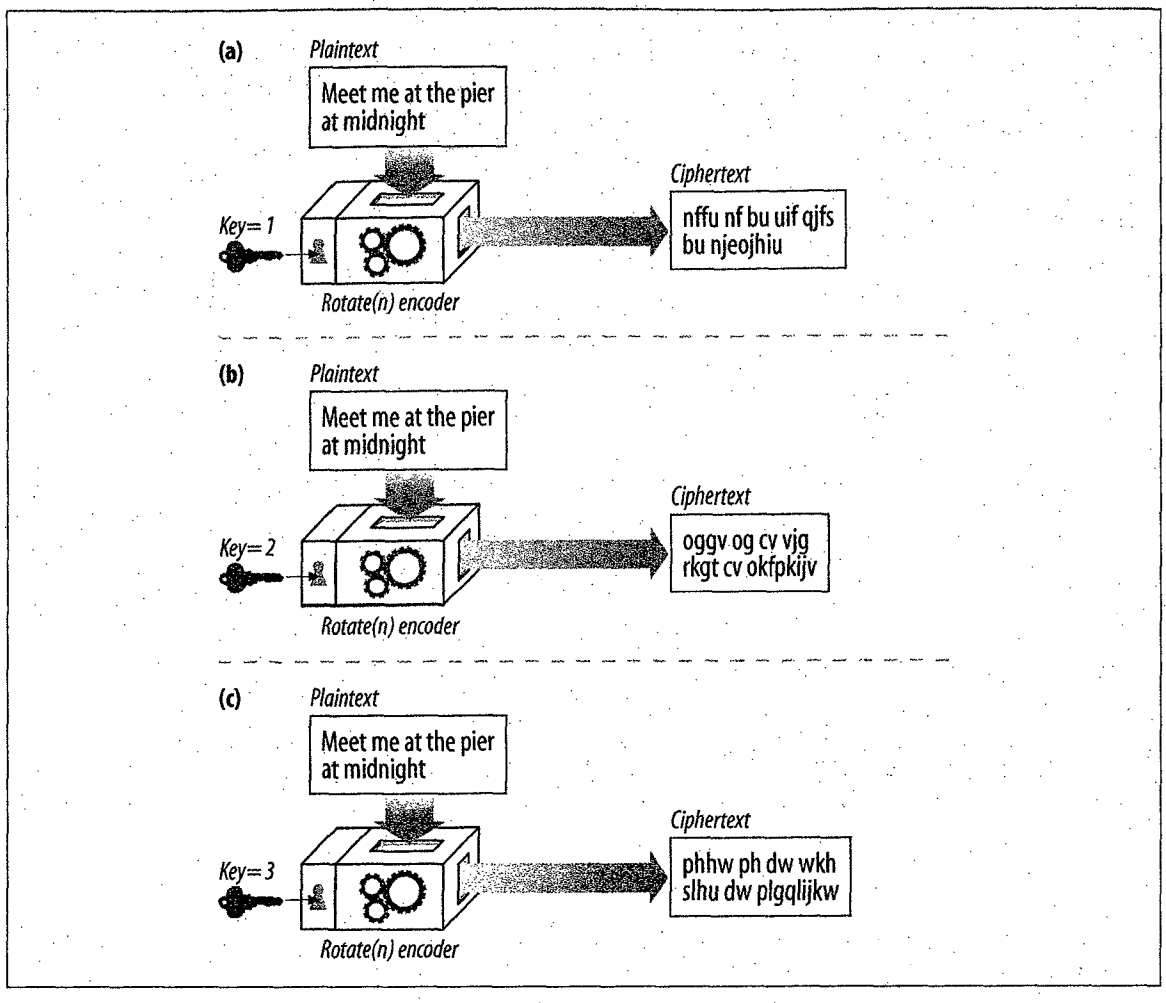


Figure 14-5. The rotate-by-N cipher, using different keys

- It became possible to support very large keys, so that a single cipher algorithm could yield trillions of virtual cipher algorithms, each differing by the value of the key. The longer the key, the more combinations of encodings are possible, and the harder it is to crack the code by randomly guessing keys.

Unlike physical metal keys or dial settings in mechanical devices, digital keys are just numbers. These digital key values are inputs to the encoding and decoding algorithms. The coding algorithms are functions that take a chunk of data and encode/decode it based on the algorithm and the value of the key.

Given a plaintext message called P, an encoding function called E, and a digital encoding key called e, you can generate a coded ciphertext message C (Figure 14-6). You can decode the ciphertext C back into the original plaintext P by using the decoder function D and the decoding key d. Of course, the decoding and encoding functions are inverses of each other; the decoding of the encoding of P gives back the original message P.

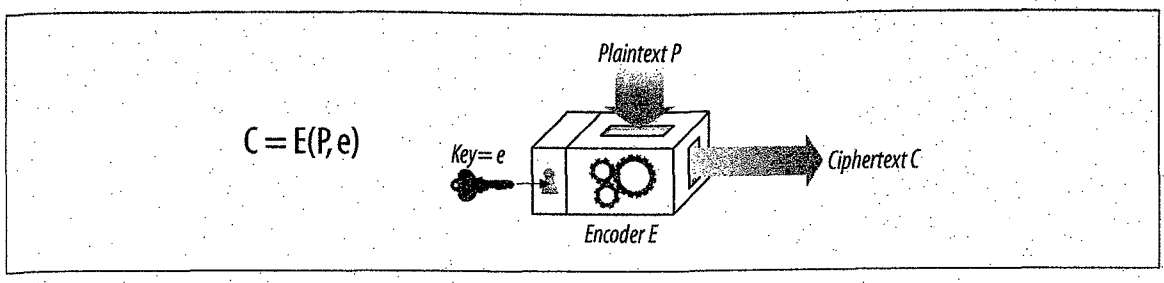


Figure 14-6. Plaintext is encoded with encoding key e, and decoded using decoding key d

Symmetric-Key Cryptography

Let's talk in more detail about how keys and ciphers work together. Many digital cipher algorithms are called *symmetric-key* ciphers, because they use the same key value for encoding as they do for decoding (e = d). Let's just call the key k.

In a symmetric key cipher, both a sender and a receiver need to have the same shared secret key, k, to communicate. The sender uses the shared secret key to encrypt the message and sends the resulting ciphertext to the receiver. The receiver takes the ciphertext and applies the decrypting function, along with the same shared secret key, to recover the original plaintext (Figure 14-7).

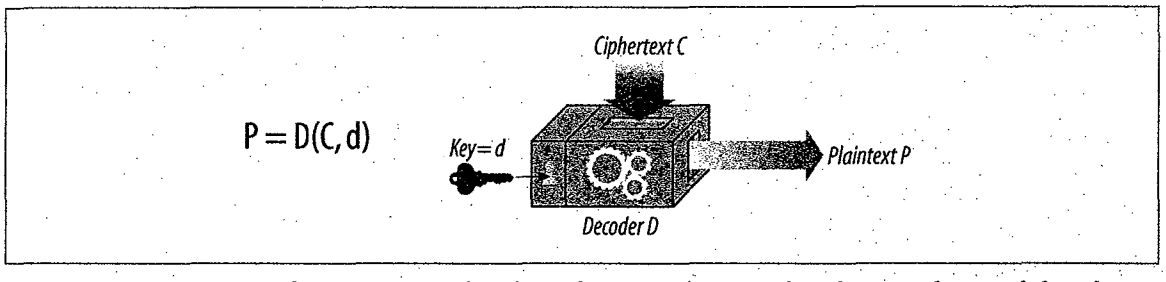


Figure 14-7. Symmetric-key cryptography algorithms use the same key for encoding and decoding

Some popular symmetric-key cipher algorithms are DES, Triple-DES, RC2, and RC4.

Key Length and Enumeration Attacks

It's very important that secret keys stay secret. In most cases, the encoding and decoding algorithms are public knowledge, so the key is the only thing that's secret!

A good cipher algorithm forces the enemy to try every single possible key value in the universe to crack the code. Trying all key values by brute force is called an *enumeration attack*. If there are only a few possible key values, a bad guy can go through all of them by brute force and eventually crack the code. But if there are a lot of possible key values, it might take the bad guy days, years, or even the lifetime of the universe to go through all the keys, looking for one that breaks the cipher.

The number of possible key values depends on the number of bits in the key and how many of the possible keys are valid. For symmetric-key ciphers, usually all of the key values are valid.* An 8-bit key would have only 256 possible keys, a 40-bit key would have 2^{40} possible keys (around one trillion keys), and a 128-bit key would generate around 340,000,000,000,000,000,000,000,000,000,000,000 possible keys.

For conventional symmetric-key ciphers, 40-bit keys are considered safe enough for small, noncritical transactions. However, they are breakable by today's high-speed workstations, which can now do billions of calculations per second.

In contrast, 128-bit keys are considered very strong for symmetric-key cryptography. In fact, long keys have such an impact on cryptographic security that the U.S. government has put export controls on cryptographic software that uses long keys, to prevent potentially antagonistic organizations from creating secret codes that the U. S. National Security Agency (NSA) would itself be unable to crack.

Bruce Schneier's excellent book, *Applied Cryptography* (John Wiley & Sons), includes a table describing the time it would take to crack a DES cipher by guessing all keys, using 1995 technology and economics.† Excerpts of this table are shown in Table 14-1.

Table 14-1. Longer keys take more effort to crack (1995 data, from "Applied Cryptography")

Attack cost	40-bit key	56-bit key	64-bit key	80-bit key	128-bit key
$100,000	2 secs	35 hours	1 year	70,000 years	10^{19} years
$1,000,000	200 msecs	3.5 hours	37 days	7,000 years	10^{18} years
$10,000,000	20 msecs	21 mins	4 days	700 years	10^{17} years
$100,000,000	2 msecs	2 mins	9 hours	70 years	10^{16} years
$1,000,000,000	200 usecs	13 secs	1 hour	7 years	10^{15} years

Given the speed of 1995 microprocessors, an attacker willing to spend $100,000 in 1995 could break a 40-bit DES code in about 2 seconds. And computers in 2002 already are 20 times faster than they were in 1995. Unless the users change keys frequently, 40-bit keys are not safe against motivated opponents.

The DES standard key size of 56 bits is more secure. In 1995 economics, a $1 million assault still would take several hours to crack the code. But a person with access to supercomputers could crack the code by brute force in a matter of seconds. In

* There are ciphers where only some of the key values are valid. For example, in RSA, the best-known asymmetric-key cryptosystem, valid keys must be related to prime numbers in a certain way. Only a small number of the possible key values have this property.

† Computation speed has increased dramatically since 1995, and cost has been reduced. And the longer it takes you to read this book, the faster they'll become! However, the table still is relatively useful, even if the times are off by a factor of 5, 10, or more.

contrast, 128-bit DES keys, similar in size to Triple-DES keys, are believed to be effectively unbreakable by anyone, at any cost, using a brute-force attack.*

Establishing Shared Keys

One disadvantage of symmetric-key ciphers is that both the sender and receiver have to have a shared secret key before they can talk to each other.

If you wanted to talk securely with Joe's Hardware store, perhaps to order some woodworking tools after watching a home-improvement program on public television, you'd have to establish a private secret key between you and *www.joes-hardware.com* before you could order anything securely. You'd need a way to generate the secret key and to remember it. Both you and Joe's Hardware, and every other Internet user, would have thousands of keys to generate and remember.

Say that Alice (A), Bob (B), and Chris (C) all wanted to talk to Joe's Hardware (J). A, B, and C each would need to establish their own secret keys with J. A would need key k^{AJ}, B would need key k^{BJ}, and C would need key k^{CJ}. Every pair of communicating parties needs its own private key. If there are N nodes, and each node has to talk securely with all the other N–1 nodes, there are roughly N^2 total secret keys: an administrative nightmare.

Public-Key Cryptography

Instead of a single encoding/decoding key for every pair of hosts, public-key cryptography uses two asymmetric keys: one for encoding messages for a host, and another for decoding the host's messages. The encoding key is publicly known to the world (thus the name public-key cryptography), but only the host knows the private decoding key (see Figure 14-8). This makes key establishment much easier, because everyone can find the public key for a particular host. But the decoding key is kept secret, so only the recipient can decode messages sent to it.

Node X can take its encoding key e^x and publish it publicly.† Now anyone wanting to send a message to node X can use the same, well-known public key. Because each host is assigned an encoding key, which everyone uses, public-key cryptography avoids the N^2 explosion of pairwise symmetric keys (see Figure 14-9).

* A large key does not mean that the cipher is foolproof, though! There may be an unnoticed flaw in the cipher algorithm or implementation that provides a weakness for an attacker to exploit. It's also possible that the attacker may have some information about how the keys are generated, so that he knows some keys are more likely than others, helping to focus a brute-force attack. Or a user might leave the secret key someplace where an attacker might be able to steal it.

† As we'll see later, most public-key lookup actually is done through digital certificates, but the details of how you find public keys don't matter much now—just know that they are publicly available somewhere.

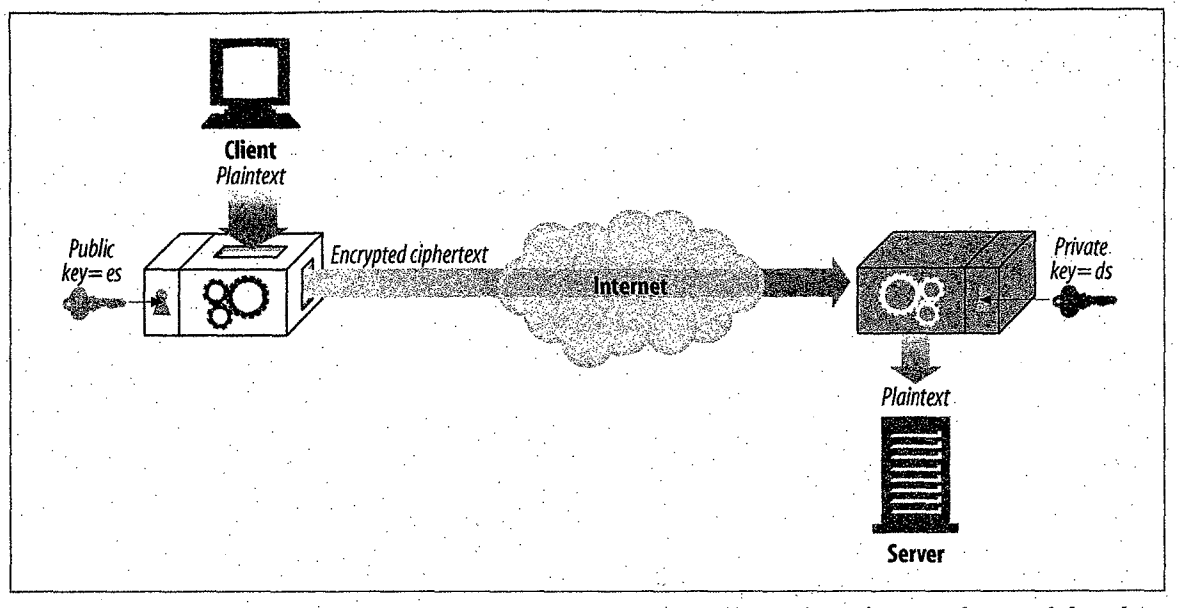


Figure 14-8. Public-key cryptography is asymmetric, using different keys for encoding and decoding

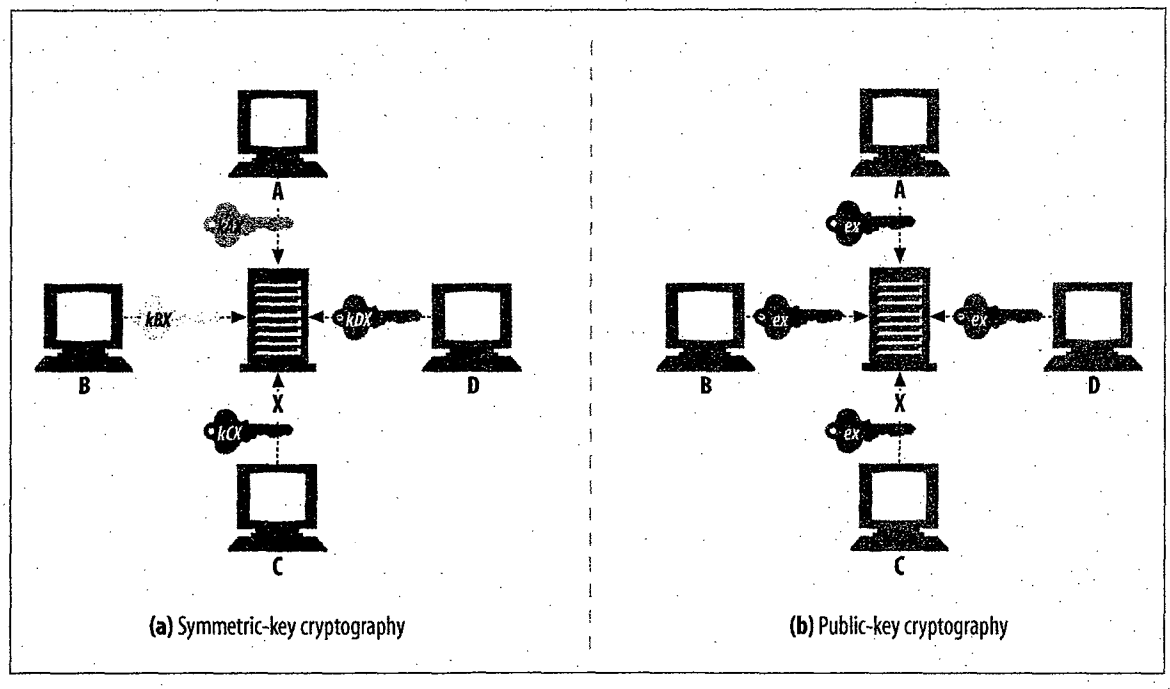


Figure 14-9. Public-key cryptography assigns a single, public encoding key to each host

Even though everyone can encode messages to X with the same key, no one other than X can decode the messages, because only X has the decoding private key d^x. Splitting the keys lets anyone encode a message but restricts the ability to decode messages to only the owner. This makes it easier for nodes to securely send messages to servers, because they can just look up the server's public key.

Public-key encryption technology makes it possible to deploy security protocols to every computer user around the world. Because of the great importance of making a

standardized public-key technology suite, a massive Public-Key Infrastructure (PKI) standards initiative has been under way for well over a decade.

RSA

The challenge of any public-key asymmetric cryptosystem is to make sure no bad guy can compute the secret, private key—even if he has all of the following clues:

- The public key (which anyone can get, because it's public)
- A piece of intercepted ciphertext (obtained by snooping the network)
- A message and its associated ciphertext (obtained by running the encoder on any text)

One popular public-key cryptosystem that meets all these needs is the RSA algorithm, invented at MIT and subsequently commercialized by RSA Data Security. Given a public key, an arbitrary piece of plaintext, the associated ciphertext from encoding the plaintext with the public key, the RSA algorithm itself, and even the source code of the RSA implementation, cracking the code to find the corresponding private key is believed to be as hard a problem as computing huge prime numbers—believed to be one of the hardest problems in all of computer science. So, if you can find a fast way of factoring large numbers into primes, not only can you break into Swiss bank accounts, but you can also win a Turing Award.

The details of RSA cryptography involve some tricky mathematics, so we won't go into them here. There are plenty of libraries available to let you perform the RSA algorithms without you needing a Ph.D. in number theory.

Hybrid Cryptosystems and Session Keys

Asymmetric, public-key cryptography is nifty, because anyone can send secure messages to a public server, just by knowing its public key. Two nodes don't first have to negotiate a private key in order to communicate securely.

But public-key cryptography algorithms tend to be computationally slow. In practice, mixtures of both symmetric and asymmetric schemes are used. For example, it is common to use public-key cryptography to conveniently set up secure communication between nodes but then to use that secure channel to generate and communicate a temporary, random symmetric key to encrypt the rest of the data through faster, symmetric cryptography.

Digital Signatures

So far, we've been talking about various kinds of keyed ciphers, using symmetric and asymmetric keys, to allow us to encrypt and decrypt secret messages.

In addition to encrypting and decrypting messages, cryptosystems can be used to *sign* messages, proving who wrote the message and proving the message hasn't been tampered with. This technique, called *digital signing*, is important for Internet security certificates, which we discuss in the next section.

Signatures Are Cryptographic Checksums

Digital signatures are special cryptographic checksums attached to a message. They have two benefits:

- Signatures prove the author wrote the message. Because only the author has the author's top-secret private key,* only the author can compute these checksums. The checksum acts as a personal "signature" from the author.
- Signatures prevent message tampering. If a malicious assailant modified the message in-flight, the checksum would no longer match. And because the checksum involves the author's secret, private key, the intruder will not be able to fabricate a correct checksum for the tampered-with message.

Digital signatures often are generated using asymmetric, public-key technology. The author's private key is used as a kind of "thumbprint," because the private key is known only by the owner.

Figure 14-10 shows an example of how node A can send a message to node B and sign it:

- Node A distills the variable-length message into a fixed-sized digest.
- Node A applies a "signature" function to the digest that uses the user's private key as a parameter. Because only the user knows the private key, a correct signature function shows the signer is the owner. In Figure 14-10, we use the decoder function D as the signature function, because it involves the user's private key.†
- Once the signature is computed, node A appends it to the end of the message and sends both the message and the signature to node B.
- On receipt, if node B wants to make sure that node A really wrote the message, and that the message hasn't been tampered with, node B can check the signature. Node B takes the private-key scrambled signature and applies the inverse function using the public key. If the unpacked digest doesn't match node B's own version of the digest, either the message was tampered with in-flight, or the sender did not have node A's private key (and therefore was not node A).

* This assumes the private key has not been stolen. Most private keys expire after a while. There also are "revocation lists" that keep track of stolen or compromised keys.

† With the RSA cryptosystem, the decoder function D is used as the signature function, because D already takes the private key as input. Note that the decoder function is just a function, so it can be used on any input. Also, in the RSA cryptosystem, the D and E functions work when applied in either order and cancel each other out. So, E(D(stuff)) = stuff, just as D(E(stuff)) = stuff.

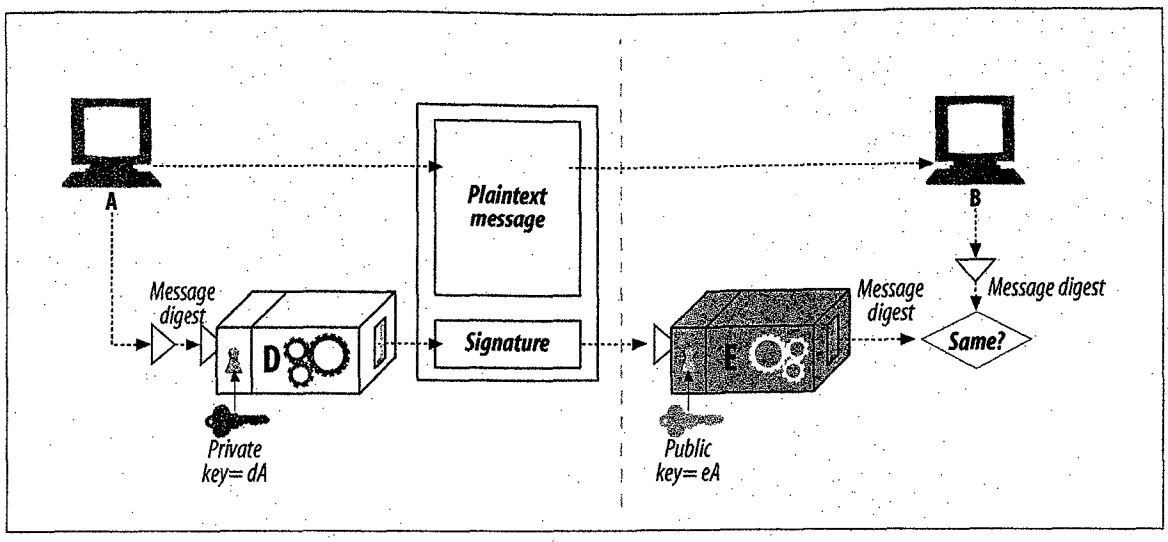


Figure 14-10. Unencrypted digital signature

Digital Certificates

In this section, we talk about digital certificates, the "ID cards" of the Internet. Digital certificates (often called "certs," like the breath mints) contain information about a user or firm that has been vouched for by a trusted organization.

We all carry many forms of identification. Some IDs, such as passports and drivers' licenses, are trusted enough to prove one's identity in many situations. For example, a U.S. driver's license is sufficient proof of identity to let you board an airplane to New York for New Year's Eve, and it's sufficient proof of your age to let you drink intoxicating beverages with your friends when you get there.

More trusted forms of identification, such as passports, are signed and stamped by a government on special paper. They are harder to forge, so they inherently carry a higher level of trust. Some corporate badges and smart cards include electronics to help strengthen the identity of the carrier. Some top-secret government organizations even need to match up your fingerprints or retinal capillary patterns with your ID before trusting it!

Other forms of ID, such as business cards, are relatively easy to forge, so people trust this information less. They may be fine for professional interactions but probably are not enough proof of employment when you apply for a home loan.

The Guts of a Certificate

Digital certificates also contain a set of information, all of which is digitally signed by an official "certificate authority." Basic digital certificates commonly contain basic things common to printed IDs, such as:

- Subject's name (person, server, organization, etc.)
- Expiration date

- Certificate issuer (who is vouching for the certificate)
- Digital signature from the certificate issuer

Additionally, digital certificates often contain the public key of the subject, as well as descriptive information about the subject and about the signature algorithm used. Anyone can create a digital certificate, but not everyone can get a well-respected signing authority to vouch for the certificate's information and sign the certificate with its private key. A typical certificate structure is shown in Figure 14-11.

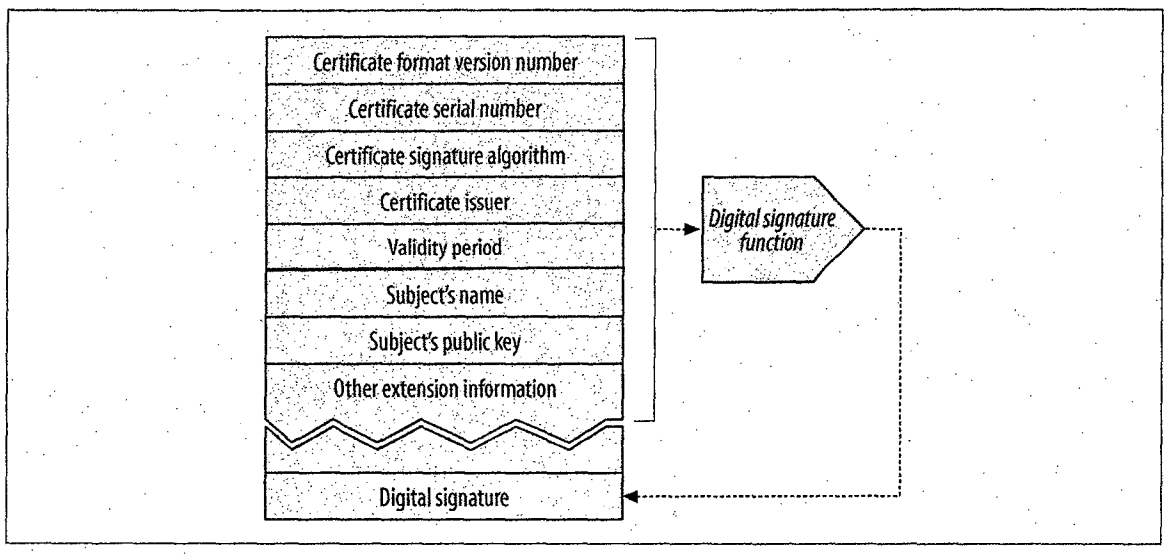


Figure 14-11. Typical digital signature format

X.509 v3 Certificates

Unfortunately, there is no single, universal standard for digital certificates. There are many, subtly different styles of digital certificates, just as not all printed ID cards contain the same information in the same place. The good news is that most certificates in use today store their information in a standard form, called X.509 v3. X.509 v3 certificates provide a standard way of structuring certificate information into parseable fields. Different kinds of certificates have different field values, but most follow the X.509 v3 structure. The fields of an X.509 certificate are described in Table 14-2.

Table 14-2. X.509 certificate fields

Field	Description
Version	The X.509 certificate version number for this certificate. Usually version 3 today.
Serial Number	A unique integer generated by the certification authority. Each certificate from a CA must have a unique serial number.
Signature Algorithm ID	The cryptographic algorithm used for the signature. For example, "MD2 digest with RSA encryption".
Certificate Issuer	The name for the organization that issued and signed this certificate, in X.500 format.
Validity Period	When this certificate is valid, defined by a start date and an end date.

Table 14-2. X.509 certificate fields (continued)

Field	Description
Subject's Name	The entity described in the certificate, such as a person or an organization. The subject name is in X.500 format.
Subject's Public Key Information	The public key for the certificate's subject, the algorithm used for the public key, and any additional parameters.
Issuer Unique ID (optional)	An optional unique identifier for the certificate issuer, to allow the potential reuse of the same issuer name.
Subject Unique ID (optional)	An optional unique identifier for the certificate subject, to allow the potential reuse of the same subject name.
Extensions	An optional set of extension fields (in version 3 and higher). Each extension field is flagged as critical or noncritical. Critical extensions are important and must be understood by the certificate user. If a certificate user doesn't recognize a critical extension field, it must reject the certificate. Common extension fields in use include: *Basic Constraints* Subject's relationship to certification authority *Certificate Policy* The policy under which the certificate is granted *Key Usage* Restricts how the public key can be used
Certification Authority Signature	The certification authority's digital signature of all of the above fields, using the specified signing algorithm.

There are several flavors of X.509-based certificates, including (among others) web server certificates, client email certificates, software code-signing certificates, and certificate authority certificates.

Using Certificates to Authenticate Servers

When you establish a secure web transaction through HTTPS, modern browsers automatically fetch the digital certificate for the server being connected to. If the server does not have a certificate, the secure connection fails. The server certificate contains many fields, including:

- Name and hostname of the web site
- Public key of the web site
- Name of the signing authority
- Signature from the signing authority

When the browser receives the certificate, it checks the signing authority.* If it is a public, well-respected signing authority, the browser will already know its public key

* Browsers and other Internet applications try hard to hide the details of most certificate management, to make browsing easier. But, when you are browsing through secure connections, all the major browsers allow you to personally examine the certificates of the sites to which you are talking, to be sure all is on the up-and-up.

(browsers ship with certificates of many signing authorities preinstalled), so it can verify the signature as we discussed in the previous section, "Digital Signatures." Figure 14-12 shows how a certificate's integrity is verified using its digital signature.

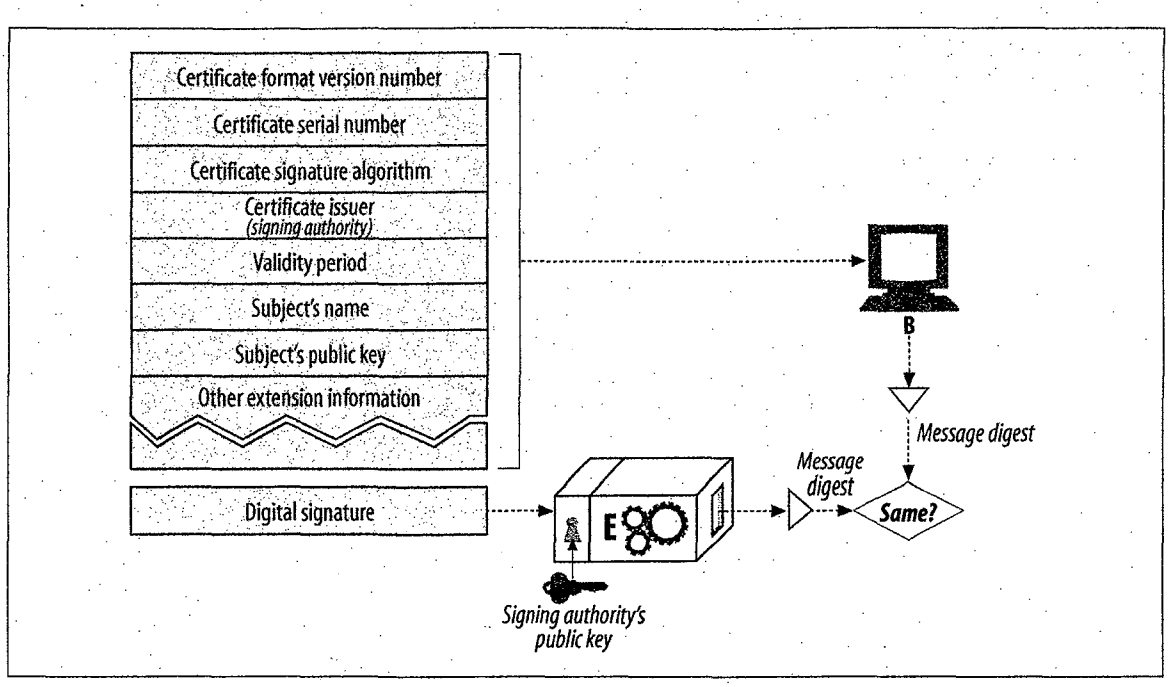


Figure 14-12. Verifying that a signature is real

If the signing authority is unknown, the browser isn't sure if it should trust the signing authority and usually displays a dialog box for the user to read and see if he trusts the signer. The signer might be the local IT department, or a software vendor.

HTTPS: The Details

HTTPS is the most popular secure version of HTTP. It is widely implemented and available in all major commercial browsers and servers. HTTPS combines the HTTP protocol with a powerful set of symmetric, asymmetric, and certificate-based cryptographic techniques, making HTTPS very secure but also very flexible and easy to administer across the anarchy of the decentralized, global Internet.

HTTPS has accelerated the growth of Internet applications and has been a major force in the rapid growth of web-based electronic commerce. HTTPS also has been critical in the wide-area, secure administration of distributed web applications.

HTTPS Overview

HTTPS is just HTTP sent over a secure transport layer. Instead of sending HTTP messages unencrypted to TCP and across the world-wide Internet (Figure 14-13a), HTTPS sends the HTTP messages first to a security layer that encrypts them before sending them to TCP (Figure 14-13b).

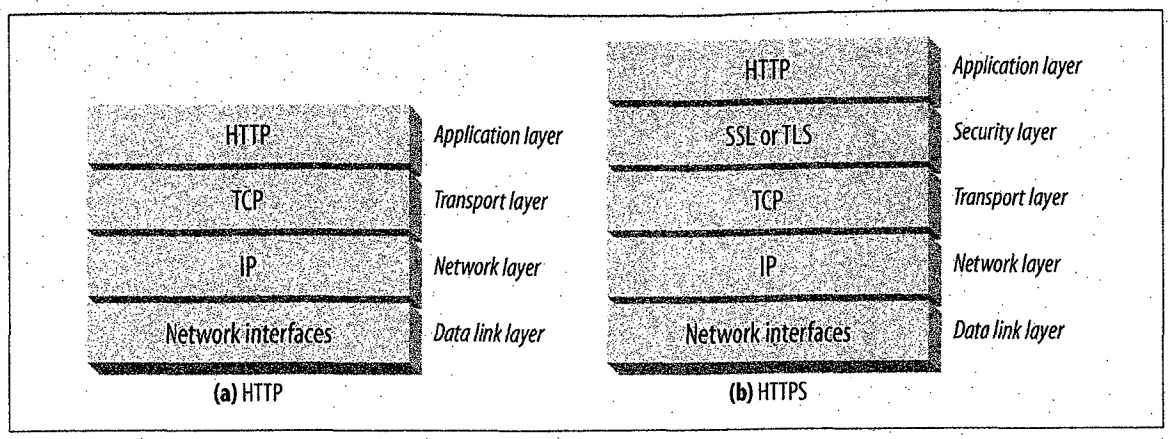


Figure 14-13. HTTP transport-level security

Today, the HTTP security layer is implemented by SSL and its modern replacement, TLS. We follow the common practice of using the term "SSL" to mean either SSL or TLS.

HTTPS Schemes

Today, secure HTTP is optional. Thus, when making a request to a web server, we need a way to tell the web server to perform the secure protocol version of HTTP. This is done in the scheme of the URL.

In normal, nonsecure HTTP, the scheme prefix of the URL is *http*, as in:

http://www.joes-hardware.com/index.html

In the secure HTTPS protocol, the scheme prefix of the URL is *https*, as in:

https://cajun-shop.securesites.com/Merchant2/merchant.mv?Store_Code=AGCGS

When a client (such as a web browser) is asked to perform a transaction on a web resource, it examines the scheme of the URL:

- If the URL has an *http* scheme, the client opens a connection to the server on port 80 (by default) and sends it plain-old HTTP commands (Figure 14-14a).
- If the URL has an *https* scheme, the client opens a connection to the server on port 443 (by default) and then "handshakes" with the server, exchanging some SSL security parameters with the server in a binary format, followed by the encrypted HTTP commands (Figure 14-14b).

Because SSL traffic is a binary protocol, completely different from HTTP, the traffic is carried on different ports (SSL usually is carried over port 443). If both SSL and HTTP traffic arrived on port 80, most web servers would interpret binary SSL traffic as erroneous HTTP and close the connection. A more integrated layering of security services into HTTP would have eliminated the need for multiple destination ports, but this does not cause severe problems in practice.

Let's look a bit more closely at how SSL sets up connections with secure servers.

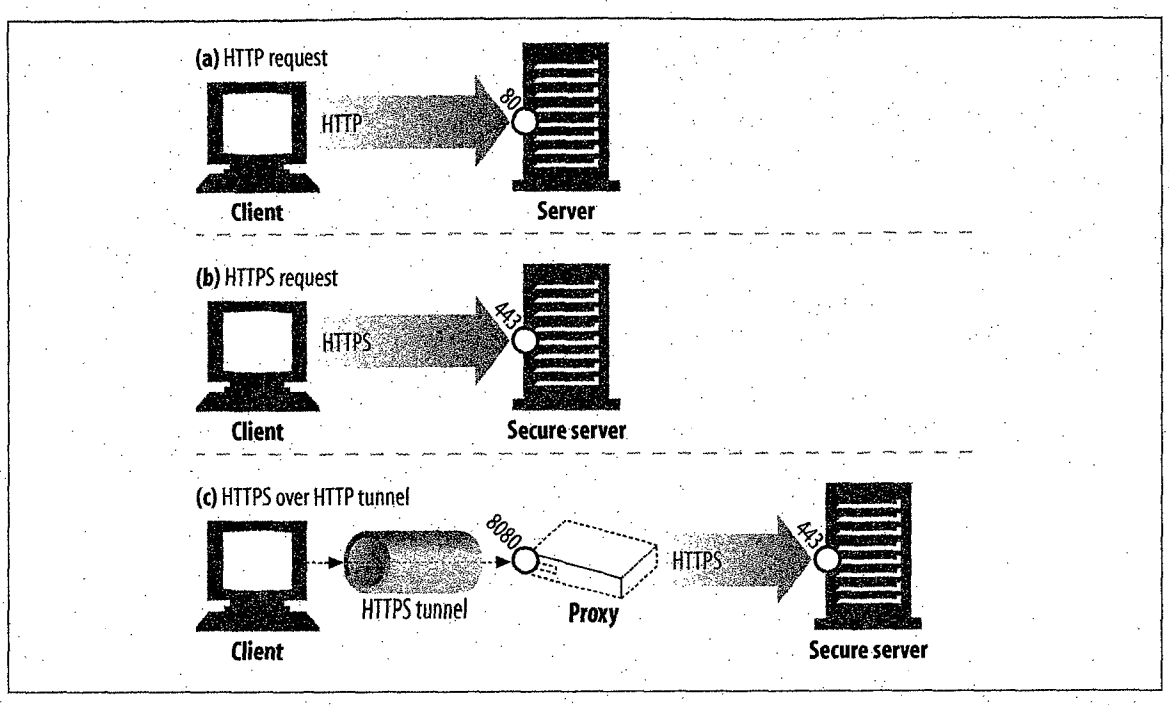


Figure 14-14. HTTP and HTTPS port numbers

Secure Transport Setup

In unencrypted HTTP, a client opens a TCP connection to port 80 on a web server, sends a request message, receives a response message, and closes the connection. This sequence is sketched in Figure 14-15a.

The procedure is slightly more complicated in HTTPS, because of the SSL security layer. In HTTPS, the client first opens a connection to port 443 (the default port for secure HTTP) on the web server. Once the TCP connection is established, the client and server initialize the SSL layer, negotiating cryptography parameters and exchanging keys. When the handshake completes, the SSL initialization is done, and the client can send request messages to the security layer. These messages are encrypted before being sent to TCP. This procedure is depicted in Figure 14-15b.

SSL Handshake

Before you can send encrypted HTTP messages, the client and server need to do an SSL handshake, where they:

- Exchange protocol version numbers
- Select a cipher that each side knows
- Authenticate the identity of each side
- Generate temporary session keys to encrypt the channel

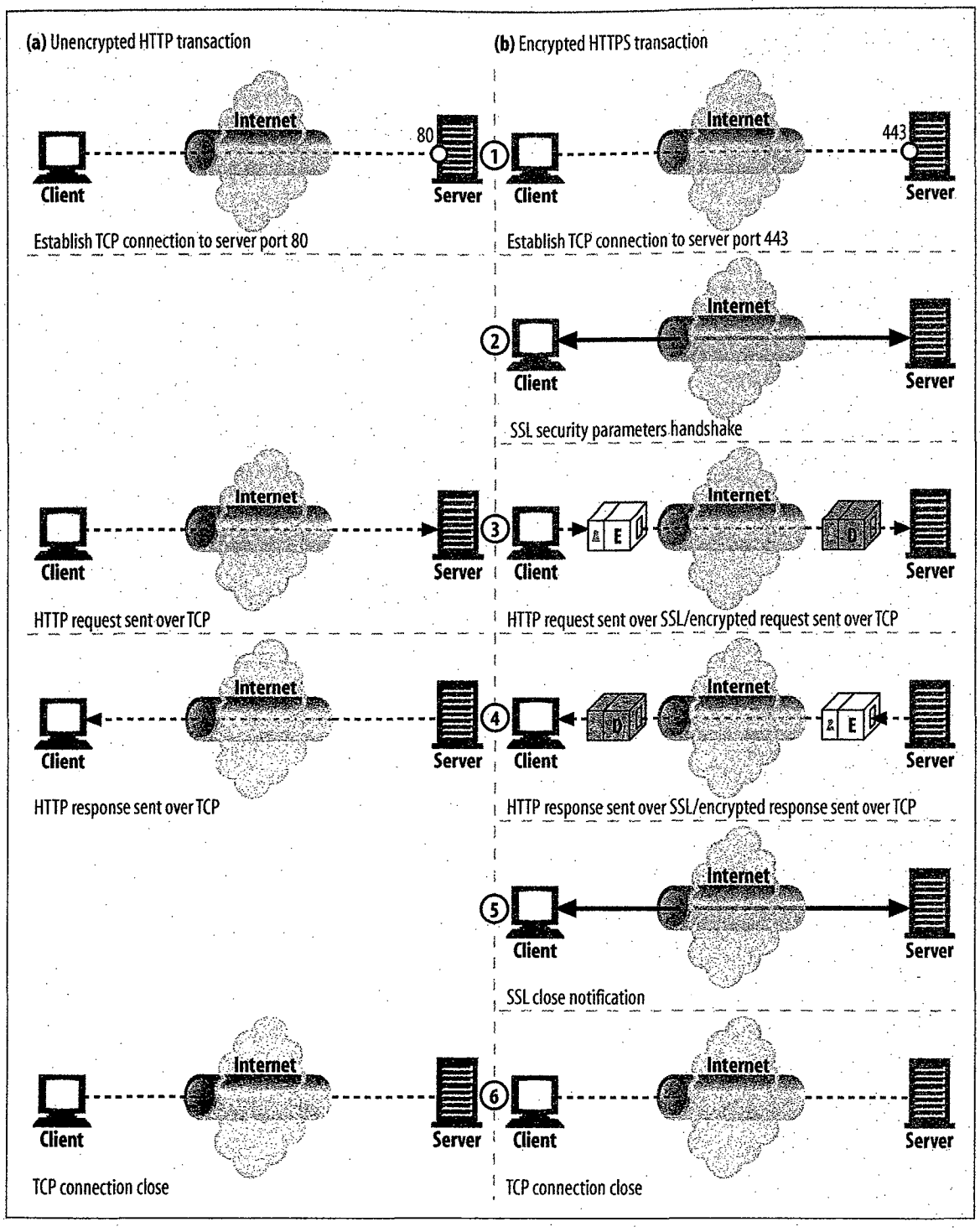


Figure 14-15. HTTP and HTTPS transactions

Before any encrypted HTTP data flies across the network, SSL already has sent a bunch of handshake data to establish the communication. The essence of the SSL handshake is shown in Figure 14-16.

This is a simplified version of the SSL handshake. Depending on how SSL is being used, the handshake can be more complicated, but this is the general idea.

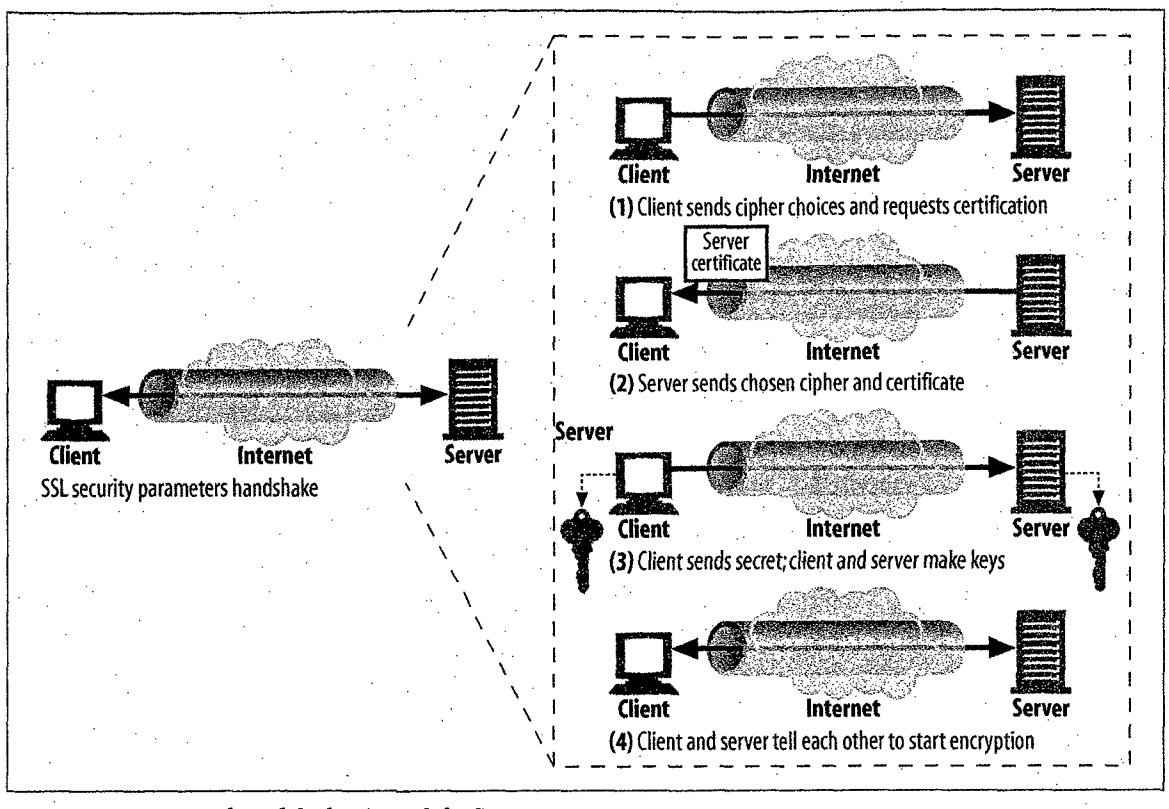


Figure 14-16. SSL handshake (simplified)

Server Certificates

SSL supports mutual authentication, carrying server certificates to clients and carrying client certificates back to servers. But today, client certificates are not commonly used for browsing. Most users don't even possess personal client certificates.* A web server can demand a client certificate, but that seldom occurs in practice.†

On the other hand, secure HTTPS transactions always require server certificates. When you perform a secure transaction on a web server, such as posting your credit card information, you want to know that you are talking to the organization you think you are talking to. Server certificates, signed by a well-known authority, help you assess how much you trust the server before sending your credit card or personal information.

The server certificate is an X.509 v3–derived certificate showing the organization's name, address, server DNS domain name, and other information (see Figure 14-17). You and your client software can examine the certificate to make sure everything seems to be on the up-and-up.

* Client certificates are used for web browsing in some corporate settings, and client certificates are used for secure email. In the future, client certificates may become more common for web browsing, but today they've caught on very slowly.

† Some organizational intranets use client certificates to control employee access to information.

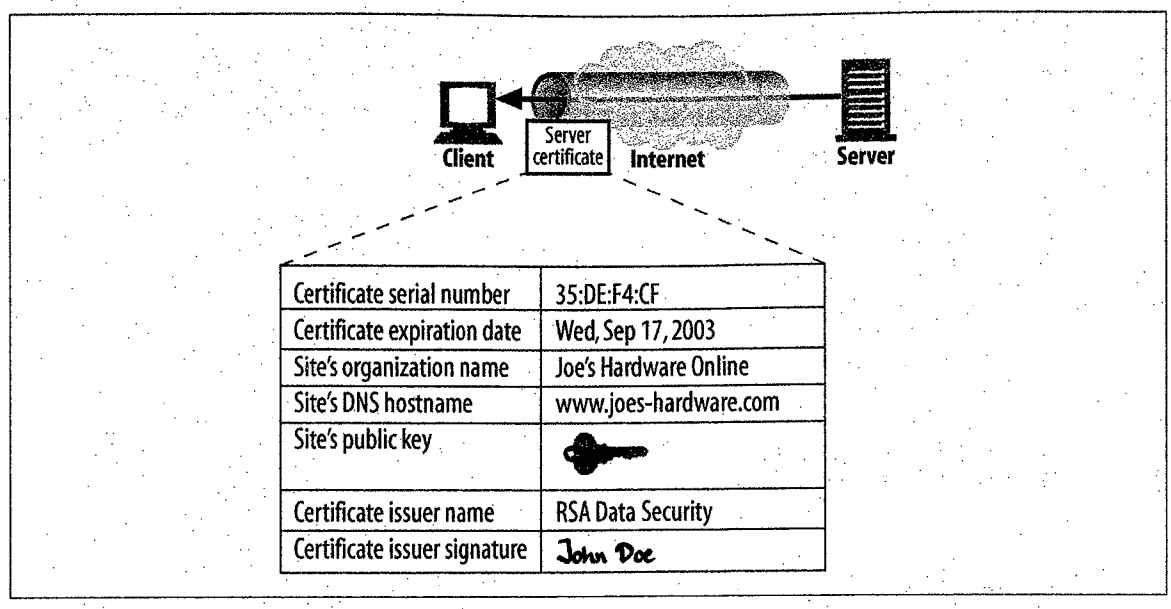


Figure 14-17. HTTPS certificates are X.509 certificates with site information

Site Certificate Validation

SSL itself doesn't require you to examine the web server certificate, but most modern browsers do some simple sanity checks on certificates and provide you with the means to do more thorough checks. One algorithm for web server certificate validation, proposed by Netscape, forms the basis of most browser's validation techniques. The steps are:

Date check

First, the browser checks the certificate's start and end dates to ensure the certificate is still valid. If the certificate has expired or has not yet become active, the certificate validation fails and the browser displays an error.

Signer trust check

Every certificate is signed by some certificate authority (CA), who vouches for the server. There are different levels of certificate, each requiring different levels of background verification. For example, if you apply for an e-commerce server certificate, you usually need to provide legal proof of incorporation as a business.

Anyone can generate certificates, but some CAs are well-known organizations with well-understood procedures for verifying the identity and good business behavior of certificate applicants. For this reason, browsers ship with a list of signing authorities that are trusted. If a browser receives a certificate signed by some unknown (and possibly malicious) authority, the browser usually displays a warning. Browsers also may choose to accept any certificates with a valid signing path to a trusted CA. In other words, if a trusted CA signs a certificate for "Sam's Signing Shop" and Sam's Signing Shop signs a site certificate, the browser may accept the certificate as deriving from a valid CA path.

Signature check
: Once the signing authority is judged as trustworthy, the browser checks the certificate's integrity by applying the signing authority's public key to the signature and comparing it to the checksum.

Site identity check
: To prevent a server from copying someone else's certificate or intercepting their traffic, most browsers try to verify that the domain name in the certificate matches the domain name of the server they talked to. Server certificates usually contain a single domain name, but some CAs create certificates that contain lists of server names or wildcarded domain names, for clusters or farms of servers. If the hostname does not match the identity in the certificate, user-oriented clients must either notify the user or terminate the connection with a bad certificate error.

Virtual Hosting and Certificates

It's sometimes tricky to deal with secure traffic on sites that are virtually hosted (multiple hostnames on a single server). Some popular web server programs support only a single certificate. If a user arrives for a virtual hostname that does not strictly match the certificate name, a warning box is displayed.

For example, consider the Louisiana-themed e-commerce site Cajun-Shop.com. The site's hosting provider provided the official name *cajun-shop.securesites.com*. When users go to *https://www.cajun-shop.com*, the official hostname listed in the server certificate (**.securesites.com*) does not match the virtual hostname the user browsed to (*www.cajun-shop.com*), and the warning in Figure 14-18 appears.

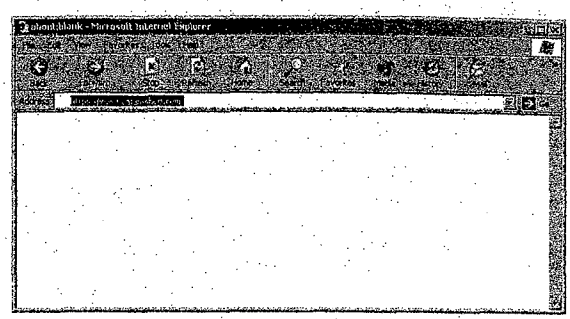
To prevent this problem, the owners of Cajun-Shop.com redirect all users to *cajun-shop.securesites.com* when they begin secure transactions. Cert management for virtually hosted sites can be a little tricky.

A Real HTTPS Client

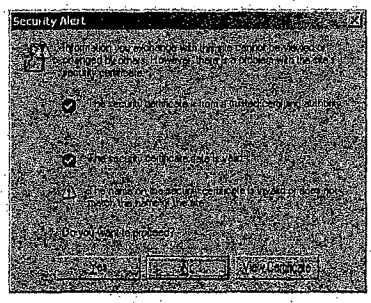
SSL is a complicated binary protocol. Unless you are a crypto expert, you shouldn't send raw SSL traffic directly. Thankfully, several commercial and open source libraries exist to make it easier to program SSL clients and servers.

OpenSSL

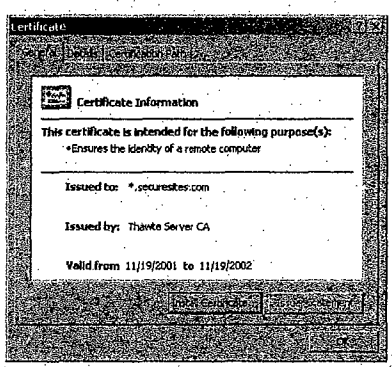
OpenSSL is the most popular open source implementation of SSL and TLS. The OpenSSL Project is a collaborative volunteer effort to develop a robust, commercial-grade, full-featured toolkit implementing the SSL and TLS protocols, as well as a full-strength, general-purpose cryptography library. You can get information about OpenSSL, and download the software, from *http://www.openssl.org*.



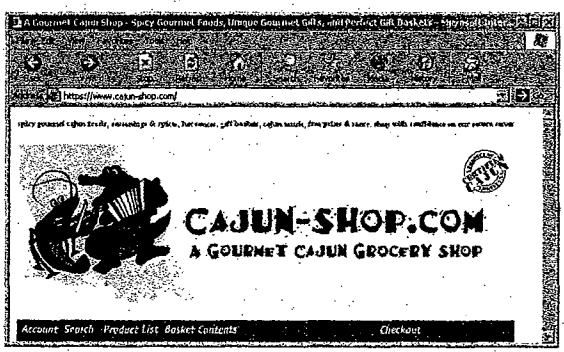
(a) The hostname in this URL (*www.cajun-shop.com*) does not match the name in the certificate, because the site is virtually hosted, and the certificate is made out to **.securesites.com*.



(b) A dialog box warns the user that the site's certificate has a valid date and is from a valid certificate authority, but the name listed in the certificate does not match the site requested in the URL.



(c) To get more details the user presses the "View Certificate" button, and sees that the certificate is a wildcard certificate made out to "*.securesites.com". With this information, the user can decide whether to accept or decline the certificate.



(d) Accepting the certificate loads the page through the secure HTTPS protocol.

To avoid this kind of user error, this particular site directs all HTTPS traffic to the hostname alias *cajun-shop.securesites.com*. This virtual hostname matches the name on the certificate provided by the ISP as part of their commerce package.

Figure 14-18. Certificate name mismatches bring up certificate error dialog boxes

You might also hear of SSLeay (pronounced S-S-L-e-a-y). OpenSSL is the successor to the SSLeay library, and it has a very similar interface. SSLeay was originally developed by Eric A. Young (the "eay" of SSLeay).

A Simple HTTPS Client

In this section, we'll use the OpenSSL package to write an extremely primitive HTTPS client. This client establishes an SSL connection with a server, prints out

some identification information from the site server, sends an HTTP GET request across the secure channel, receives an HTTP response, and prints the response.

The C program shown below is an OpenSSL implementation of the trivial HTTPS client. To keep the program simple, error-handling and certificate-processing logic has not been included.

Because error handling has been removed from this example program, you should use it only for explanatory value. The software will crash or otherwise misbehave in normal error conditions.

```
/**********************************************************************
 * https_client.c --- very simple HTTPS client with no error checking
 *     usage: https_client servername
 **********************************************************************/

#include <stdio.h>
#include <memory.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

void main(int argc, char **argv)
{
    SSL *ssl;
    SSL_CTX *ctx;
    SSL_METHOD *client_method;
    X509 *server_cert;
    int sd,err;
    char *str,*hostname,outbuf[4096],inbuf[4096],host_header[512];
    struct hostent *host_entry;
    struct sockaddr_in server_socket_address;
    struct in_addr ip;

    /*-------------------------------------*/
    /* (1) initialize SSL library */
    /*-------------------------------------*/

    SSLeay_add_ssl_algorithms( );
    client_method = SSLv2_client_method( );
    SSL_load_error_strings( );
    ctx = SSL_CTX_new(client_method);
```

```
printf("(1) SSL context initialized\n\n");

/*=============================================*/
/* (2) convert server hostname into IP address */
/*=============================================*/

hostname = argv[1];
host_entry = gethostbyname(hostname);
bcopy(host_entry->h_addr, &(ip.s_addr), host_entry->h_length);

printf("(2) '%s' has IP address '%s'\n\n", hostname, inet_ntoa(ip));

/*=================================================*/
/* (3) open a TCP connection to port 443 on server */
/*=================================================*/

sd = socket (AF_INET, SOCK_STREAM, 0);

memset(&server_socket_address, '\0', sizeof(server_socket_address));
server_socket_address.sin_family = AF_INET;
server_socket_address.sin_port = htons(443);
memcpy(&(server_socket_address.sin_addr.s_addr),
       host_entry->h_addr, host_entry->h_length);

err = connect(sd, (struct sockaddr*) &server_socket_address,
             sizeof(server_socket_address));
if (err < 0) { perror("can't connect to server port"); exit(1); }

printf("(3) TCP connection open to host '%s', port %d\n\n",
       hostname, server_socket_address.sin_port);

/*=========================================================*/
/* (4) initiate the SSL handshake over the TCP connection */
/*=========================================================*/

ssl = SSL_new(ctx);          /* create SSL stack endpoint */
SSL_set_fd(ssl, sd);         /* attach SSL stack to socket */
err = SSL_connect(ssl);      /* initiate SSL handshake */

printf("(4) SSL endpoint created & handshake completed\n\n");

/*=============================================*/
/* (5) print out the negotiated cipher chosen */
/*=============================================*/

printf("(5) SSL connected with cipher: %s\n\n", SSL_get_cipher(ssl));

/*=======================================*/
/* (6) print out the server's certificate */
/*=======================================*/

server_cert = SSL_get_peer_certificate(ssl);
```

```
    printf("(6) server's certificate was received:\n\n");

    str = X509_NAME_oneline(X509_get_subject_name(server_cert), 0, 0);
    printf("      subject: %s\n", str);

    str = X509_NAME_oneline(X509_get_issuer_name(server_cert), 0, 0);
    printf("      issuer: %s\n\n", str);

    /* certificate verification would happen here */

    X509_free(server_cert);

    /*---------------------------------------------------------*/
    /* (7) handshake complete --- send HTTP request over SSL */
    /*---------------------------------------------------------*/

    sprintf(host_header,"Host: %s:443\r\n",hostname);
    strcpy(outbuf,"GET / HTTP/1.0\r\n");
    strcat(outbuf,host_header);
    strcat(outbuf,"Connection: close\r\n");
    strcat(outbuf,"\r\n");

    err = SSL_write(ssl, outbuf, strlen(outbuf));
    shutdown (sd, 1); /* send EOF to server */

    printf("(7) sent HTTP request over encrypted channel:\n\n%s\n",outbuf);

    /*-------------------------------------------------*/
    /* (8) read back HTTP response from the SSL stack */
    /*-------------------------------------------------*/

    err = SSL_read(ssl, inbuf, sizeof(inbuf) - 1);
    inbuf[err] = '\0';
    printf ("(8) got back %d bytes of HTTP response:\n\n%s\n",err,inbuf);

    /*-----------------------------------------------*/
    /* (9) all done, so close connection & clean up */
    /*-----------------------------------------------*/

    SSL_shutdown(ssl);
    close (sd);
    SSL_free (ssl);
    SSL_CTX_free (ctx);

    printf("(9) all done, cleaned up and closed connection\n\n");
}
```

This example compiles and runs on Sun Solaris, but it is illustrative of how SSL programs work on many OS platforms. This entire program, including all the encryption and key and certificate management, fits in a three-page C program, thanks to the powerful features provided by OpenSSL.

Let's walk through the program section by section:

- The top of the program includes support files needed to support TCP networking and SSL.
- Section 1 creates the local context that keeps track of the handshake parameters and other state about the SSL connection, by calling *SSL_CTX_new*.
- Section 2 converts the input hostname (provided as a command-line argument) to an IP address, using the Unix *gethostbyname* function. Other platforms may have other ways to provide this facility.
- Section 3 opens a TCP connection to port 443 on the server by creating a local socket, setting up the remote address information, and connecting to the remote server.
- Once the TCP connection is established, we attach the SSL layer to the TCP connection using *SSL_new* and *SSL_set_fd* and perform the SSL handshake with the server by calling *SSL_connect*. When section 4 is done, we have a functioning SSL channel established, with ciphers chosen and certificates exchanged.
- Section 5 prints out the value of the chosen bulk-encryption cipher.
- Section 6 prints out some of the information contained in the X.509 certificate sent back from the server, including information about the certificate holder and the organization that issued the certificate. The OpenSSL library doesn't do anything special with the information in the server certificate. A real SSL application, such as a web browser, would do some sanity checks on the certificate to make sure it is signed properly and came from the right host. We discussed what browsers do with server certificates in "Site Certificate Validation."
- At this point, our SSL connection is ready to use for secure data transfer. In section 7, we send the simple HTTP request "GET / HTTP/1.0" over the SSL channel using *SSL_write*, then close the outbound half of the connection.
- In section 8, we read the response back from the connection using *SSL_read*, and print it on the screen. Because the SSL layer takes care of all the encryption and decryption, we can just write and read normal HTTP commands.
- Finally, we clean up in section 9.

Refer to *http://www.openssl.org* for more information about the OpenSSL libraries.

Executing Our Simple OpenSSL Client

The following shows the output of our simple HTTP client when pointed at a secure server. In this case, we pointed the client at the home page of the Morgan Stanley Online brokerage. Online trading companies make extensive use of HTTPS.

```
% https_client clients1.online.msdw.com
(1) SSL context initialized
```

```
(2) 'clients1.online.msdw.com' has IP address '63.151.15.11'

(3) TCP connection open to host 'clients1.online.msdw.com', port 443

(4) SSL endpoint created & handshake completed

(5) SSL connected with cipher: DES-CBC3-MD5

(6) server's certificate was received:

      subject: /C=US/ST=Utah/L=Salt Lake City/O=Morgan Stanley/OU=Online/CN=
        clients1.online.msdw.com
      issuer: /C=US/O=RSA Data Security, Inc./OU=Secure Server Certification
        Authority

(7) sent HTTP request over encrypted channel:

GET / HTTP/1.0
Host: clients1.online.msdw.com:443
Connection: close

(8) got back 615 bytes of HTTP response:

HTTP/1.1 302 Found
Date: Sat, 09 Mar 2002 09:43:42 GMT
Server: Stronghold/3.0 Apache/1.3.14 RedHat/3013c (Unix) mod_ssl/2.7.1 OpenSSL/0.9.6
Location: https://clients.online.msdw.com/cgi-bin/ICenter/home
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A HREF="https://clients.online.msdw.com/cgi-bin/ICenter/
home">here</A>.<P>
<HR>
<ADDRESS>Stronghold/3.0 Apache/1.3.14 RedHat/3013c Server at clients1.online.msdw.com
Port 443</ADDRESS>
</BODY></HTML>

(9) all done, cleaned up and closed connection
```

As soon as the first four sections are completed, the client has an open SSL connection. It can then inquire about the state of the connection and chosen parameters and can examine server certificates.

In this example, the client and server negotiated the DES-CBC3-MD5 bulk-encryption cipher. You also can see that the server site certificate belongs to the organization "Morgan Stanley" in "Salt Lake City, Utah, USA". The certificate was granted by RSA Data Security, and the hostname is "clients1.online.msdw.com," which matches our request.

Once the SSL channel is established and the client feels comfortable about the site certificate, it sends its HTTP request over the secure channel. In our example, the client sends a simple "GET / HTTP/1.0" HTTP request and receives back a 302 Redirect response, requesting that the user fetch a different URL.

Tunneling Secure Traffic Through Proxies

Clients often use web proxy servers to access web servers on their behalf (proxies are discussed in Chapter 6). For example, many corporations place a proxy at the security perimeter of the corporate network and the public Internet (Figure 14-19). The proxy is the only device permitted by the firewall routers to exchange HTTP traffic, and it may employ virus checking or other content controls.

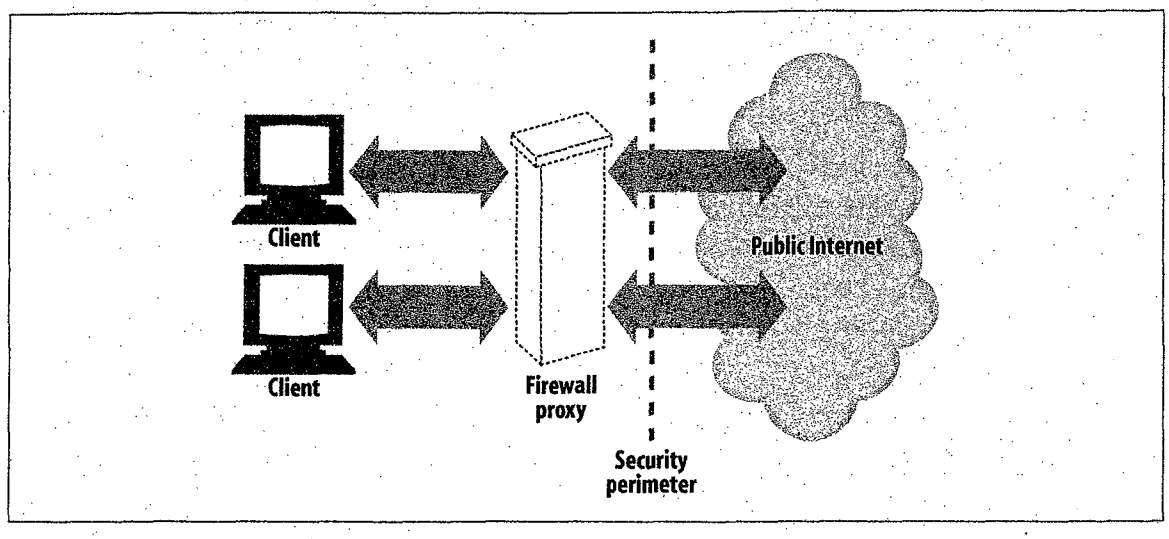


Figure 14-19. Corporate firewall proxy

But once the client starts encrypting the data to the server, using the server's public key, the proxy no longer has the ability to read the HTTP header! And if the proxy cannot read the HTTP header, it won't know where to forward the request (Figure 14-20).

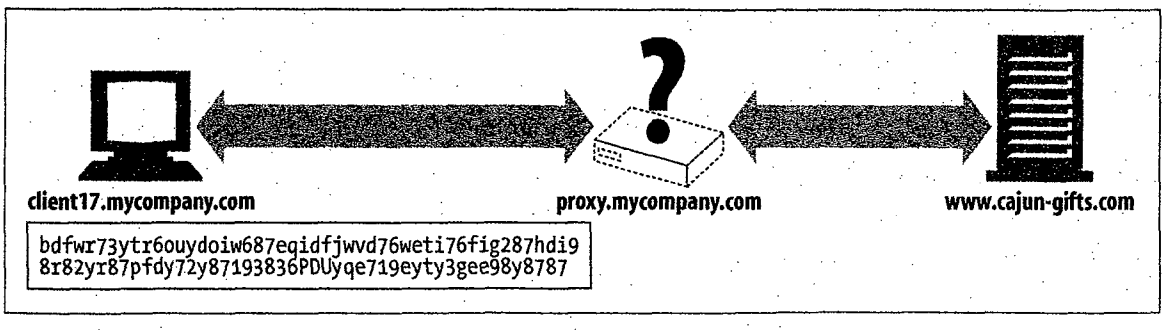


Figure 14-20. Proxy can't proxy an encrypted request

To make HTTPS work with proxies, a few modifications are needed to tell the proxy where to connect. One popular technique is the HTTPS SSL tunneling protocol.

Using the HTTPS tunneling protocol, the client first tells the proxy the secure host and port to which it wants to connect. It does this in plaintext, before encryption starts, so the proxy can read this information.

HTTP is used to send the plaintext endpoint information, using a new extension method called CONNECT. The CONNECT method tells the proxy to open a connection to the desired host and port number and, when that's done, to tunnel data directly between the client and server. The CONNECT method is a one-line text command that provides the hostname and port of the secure origin server, separated by a colon. The host:port is followed by a space and an HTTP version string followed by a CRLF. After that there is a series of zero or more HTTP request header lines, followed by an empty line. After the empty line, if the handshake to establish the connection was successful, SSL data transfer can begin. Here is an example:

```
CONNECT home.netscape.com:443 HTTP/1.0
User-agent: Mozilla/1.1N

<raw SSL-encrypted data would follow here...>
```

After the empty line in the request, the client will wait for a response from the proxy. The proxy will evaluate the request and make sure that it is valid and that the user is authorized to request such a connection. If everything is in order, the proxy will make a connection to the destination server and, if successful, send a 200 Connection Established response to the client.

```
HTTP/1.0 200 Connection established
Proxy-agent: Netscape-Proxy/1.1
```

For more information about secure tunnels and security proxies, refer back to "Tunnels" in Chapter 8.

For More Information

Security and cryptography are hugely important and hugely complicated topics. If you'd like to learn more about HTTP security, digital cryptography, digital certificates, and the Public-Key Infrastructure, here are a few starting points.

HTTP Security

Web Security, Privacy & Commerce
: Simson Garfinkel, O'Reilly & Associates, Inc. This is one of the best, most readable introductions to web security and the use of SSL/TLS and digital certificates.

http://www.ietf.org/rfc/rfc2818.txt
: RFC 2818, "HTTP Over TLS," specifies how to implement secure HTTP over Transport Layer Security (TLS), the modern successor to SSL.

http://www.ietf.org/rfc/rfc2817.txt

RFC 2817, "Upgrading to TLS Within HTTP/1.1," explains how to use the Upgrade mechanism in HTTP/1.1 to initiate TLS over an existing TCP connection. This allows unsecured and secured HTTP traffic to share the same well-known port (in this case, http: at 80 rather than https: at 443). It also enables virtual hosting, so a single HTTP+TLS server can disambiguate traffic intended for several hostnames at a single IP address.

SSL and TLS

http://www.ietf.org/rfc/rfc2246.txt

RFC 2246, "The TLS Protocol Version 1.0," specifies Version 1.0 of the TLS protocol (the successor to SSL). TLS provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

"Introduction to SSL" introduces the Secure Sockets Layer (SSL) protocol. Originally developed by Netscape, SSL has been universally accepted on the World Wide Web for authenticated and encrypted communication between clients and servers.

http://www.netscape.com/eng/ssl3/draft302.txt

"The SSL Protocol Version 3.0" is Netscape's 1996 specification for SSL.

http://developer.netscape.com/tech/security/ssl/howitworks.html

"How SSL Works" is Netscape's introduction to key cryptography.

http://www.openssl.org

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols, as well as a full-strength, general-purpose cryptography library. The project is managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation. OpenSSL is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson. The OpenSSL toolkit is licensed under an Apache-style licence, which basically means that you are free to get and use it for commercial and noncommercial purposes, subject to some simple license conditions.

Public-Key Infrastructure

http://www.ietf.org/html.charters/pkix-charter.html

The IETF PKIX Working Group was established in 1995 with the intent of developing Internet standards needed to support an X.509-based Public-Key Infrastructure. This is a nice summary of that group's activities.

http://www.ietf.org/rfc/rfc2459.txt
: RFC 2459, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," contains details about X.509 v3 digital certificates.

Digital Cryptography

Applied Cryptography
: Bruce Schneier, John Wiley & Sons. This is a classic book on cryptography for implementors.

The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography
: Simon Singh, Anchor Books. This entertaining book is a cryptography primer. While it's not intended for technology experts, it is a lively historical tour of secret coding.

PART IV

Entities, Encodings, and Internationalization

Part IV is all about the entity bodies of HTTP messages and the content that the entity bodies ship around as cargo:

- Chapter 15, *Entities and Encodings*, describes the formats and syntax of HTTP content.
- Chapter 16, *Internationalization*, surveys the web standards that allow people to exchange content in different languages and different character sets, around the globe.
- Chapter 17, *Content Negotiation and Transcoding*, explains mechanisms for negotiating acceptable content.

CHAPTER 15

Entities and Encodings

HTTP ships billions of media objects of all kinds every day. Images, text, movies, software programs... you name it, HTTP ships it. HTTP also makes sure that its messages can be properly transported, identified, extracted, and processed. In particular, HTTP ensures that its cargo:

- Can be identified correctly (using Content-Type media formats and Content-Language headers) so browsers and other clients can process the content properly
- Can be unpacked properly (using Content-Length and Content-Encoding headers)
- Is fresh (using entity validators and cache-expiration controls)
- Meets the user's needs (based on content-negotiation Accept headers)
- Moves quickly and efficiently through the network (using range requests, delta encoding, and other data compression)
- Arrives complete and untampered with (using transfer encoding headers and Content-MD5 checksums)

To make all this happen, HTTP uses well-labeled entities to carry content.

This chapter discusses entities, their associated entity headers, and how they work to transport web cargo. We'll show how HTTP provides the essentials of content size, type, and encodings. We'll also explain some of the more complicated and powerful features of HTTP entities, including range requests, delta encoding, digests, and chunked encodings.

This chapter covers:

- The format and behavior of HTTP message entities as HTTP data containers
- How HTTP describes the size of entity bodies, and what HTTP requires in the way of sizing
- The entity headers used to describe the format, alphabet, and language of content, so clients can process it properly

- Reversible content encodings, used by senders to transform the content data format before sending to make it take up less space or be more secure
- Transfer encoding, which modifies how HTTP ships data to enhance the communication of some kinds of content, and chunked encoding, a transfer encoding that chops data into multiple pieces to deliver content of unknown length safely
- The assortment of tags, labels, times, and checksums that help clients get the latest version of requested content
- The validators that act like version numbers on content, so web applications can ensure they have fresh content, and the HTTP header fields designed to control object freshness
- Ranges, which are useful for continuing aborted downloads where they left off
- HTTP delta encoding extensions, which allow clients to request just those parts of a web page that actually have changed since a previously viewed revision
- Checksums of entity bodies, which are used to detect changes in entity content as it passes through proxies

Messages Are Crates, Entities Are Cargo

If you think of HTTP messages as the crates of the Internet shipping system, then HTTP entities are the actual cargo of the messages. Figure 15-1 shows a simple entity, carried inside an HTTP response message.

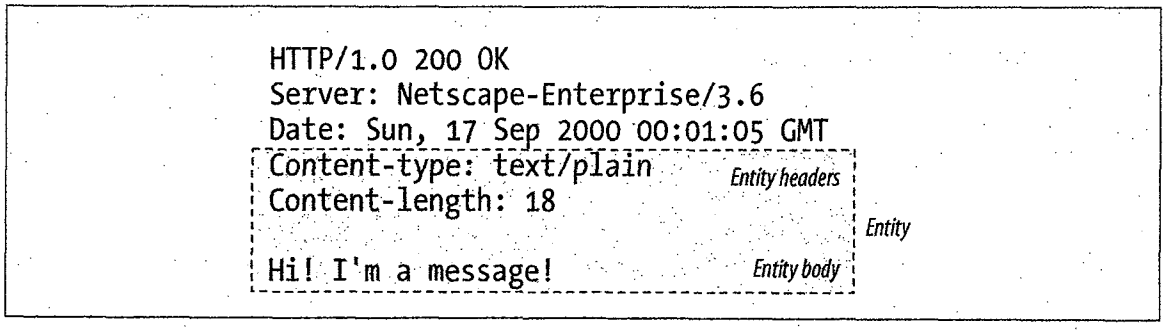


Figure 15-1. Message entity is made up of entity headers and entity body

The entity headers indicate a plaintext document (Content-Type: text/plain) that is a mere 18 characters long (Content-Length: 18). As always, a blank line (CRLF) separates the header fields from the start of the body.

HTTP entity headers (covered in Chapter 3) describe the contents of an HTTP message. HTTP/1.1 defines 10 primary entity header fields:

Content-Type
: The kind of object carried by the entity.

Content-Length
: The length or size of the message being sent.

Content-Language
: The human language that best matches the object being sent.

Content-Encoding
: Any transformation (compression, etc.) performed on the object data.

Content-Location
: An alternate location for the object at the time of the request.

Content-Range
: If this is a partial entity, this header defines which pieces of the whole are included.

Content-MD5
: A checksum of the contents of the entity body.

Last-Modified
: The date on which this particular content was created or modified at the server.

Expires
: The date and time at which this entity data will become stale.

Allow
: What request methods are legal on this resource; e.g., GET and HEAD.

ETag
: A unique validator for this particular instance* of the document. The ETag header is not defined formally as an entity header, but it is an important header for many operations involving entities.

Cache-Control
: Directives on how this document can be cached. The Cache-Control header, like the ETag header, is not defined formally as an entity header.

Entity Bodies

The entity body just contains the raw cargo.† Any other descriptive information is contained in the headers. Because the entity body cargo is just raw data, the entity headers are needed to describe the meaning of that data. For example, the Content-Type entity header tells us how to interpret the data (image, text, etc.), and the Content-Encoding entity header tells us if the data was compressed or otherwise recoded. We talk about all of this and more in upcoming sections.

The raw content begins immediately after the blank CRLF line that marks the end of the header fields. Whatever the content is—text or binary, document or image, compressed or uncompressed, English or French or Japanese—it is placed right after the CRLF.

* Instances are described later in this chapter, in the section "Time-Varying Instances."

† If there is a Content-Encoding header, the content already has been encoded by the content-encoding algorithm, and the first byte of the entity is the first byte of the encoded (e.g., compressed) cargo.

Figure 15-2 shows two examples of real HTTP messages, one carrying a text entity, the other carrying an image entity. The hexadecimal values show the exact contents of the message:

- In Figure 15-2a, the entity body begins at byte number 65, right after the end-of-headers CRLF. The entity body contains the ASCII characters for "Hi! I'm a message!"
- In Figure 15-2b, the entity body begins at byte number 67. The entity body contains the binary contents of the GIF image. GIF files begin with 6-byte version signature, a 16-bit width, and a 16-bit height. You can see all three of these directly in the entity body.

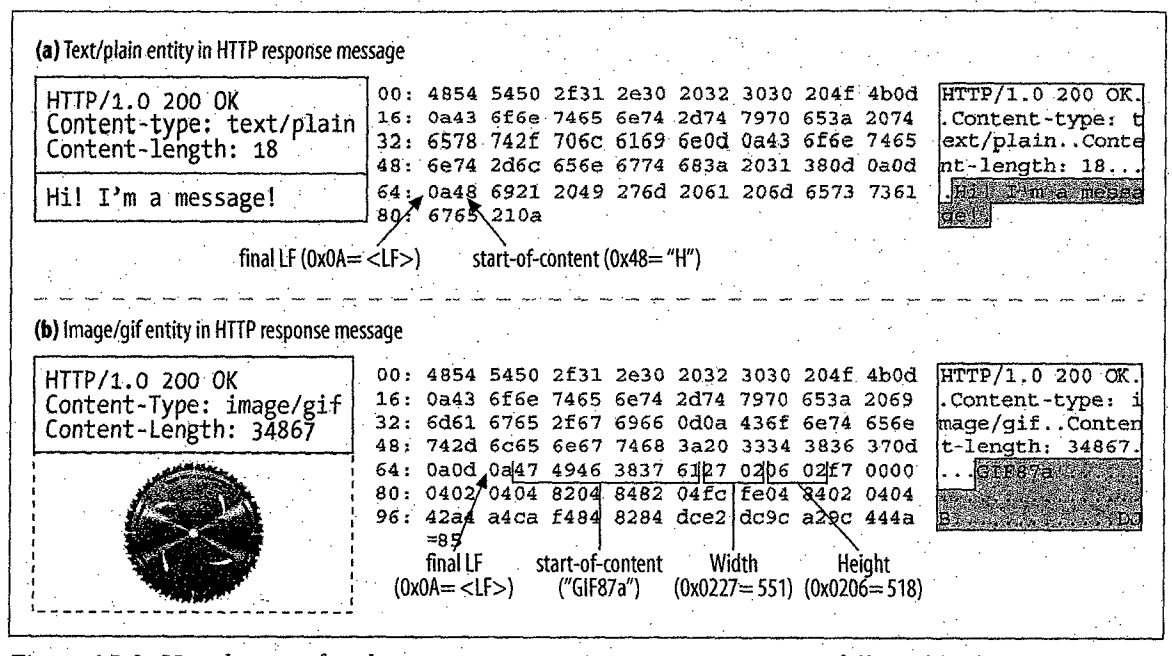


Figure 15-2. Hex dumps of real message content (raw message content follows blank CRLF)

Content-Length: The Entity's Size

The Content-Length header indicates the size of the entity body in the message, in bytes. The size includes any content encodings (the Content-Length of a gzip-compressed text file will be the compressed size, not the original size).

The Content-Length header is mandatory for messages with entity bodies, unless the message is transported using chunked encoding. Content-Length is needed to detect premature message truncation when servers crash and to properly segment messages that share a persistent connection.

Detecting Truncation

Older versions of HTTP used connection close to delimit the end of a message. But, without Content-Length, clients cannot distinguish between successful connection

close at the end of a message and connection close due to a server crash in the middle of a message. Clients need Content-Length to detect message truncation.

Message truncation is especially severe for caching proxy servers. If a cache receives a truncated message and doesn't recognize the truncation, it may store the defective content and serve it many times. Caching proxy servers generally do not cache HTTP bodies that don't have an explicit Content-Length header, to reduce the risk of caching truncated messages.

Incorrect Content-Length

An incorrect Content-Length can cause even more damage than a missing Content-Length. Because some early clients and servers had well-known bugs with respect to Content-Length calculations, some clients, servers, and proxies contain algorithms to try to detect and correct interactions with broken servers. HTTP/1.1 user agents officially are supposed to notify the user when an invalid length is received and detected.

Content-Length and Persistent Connections

Content-Length is essential for persistent connections. If the response comes across a persistent connection, another HTTP response can immediately follow the current response. The Content-Length header lets the client know where one message ends and the next begins. Because the connection is persistent, the client cannot use connection close to identify the message's end. Without a Content-Length header, HTTP applications won't know where one entity body ends and the next message begins.

As we will see in "Transfer Encoding and Chunked Encoding," there is one situation where you can use persistent connections without having a Content-Length header: when you use *chunked encoding*. Chunked encoding sends the data in a series of chunks, each with a specified size. Even if the server does not know the size of the entire entity at the time the headers are generated (often because the entity is being generated dynamically), the server can use chunked encoding to transmit pieces of well-defined size.

Content Encoding

HTTP lets you encode the contents of an entity body, perhaps to make it more secure or to compress it to take up less space (we explain compression in detail later in this chapter). If the body has been content-encoded, the Content-Length header specifies the length, in bytes, of the *encoded* body, *not* the length of the original, unencoded body.

Some HTTP applications have been known to get this wrong and to send the size of the data before the encoding, which causes serious errors, especially with persistent connections. Unfortunately, none of the headers described in the HTTP/1.1

specification can be used to send the length of the original, unencoded body, which makes it difficult for clients to verify the integrity of their unencoding processes.*

Rules for Determining Entity Body Length

The following rules describe how to correctly determine the length and end of an entity body in several different circumstances. The rules should be applied in order; the first match applies.

1. If a particular HTTP message type is not allowed to have a body, ignore the Content-Length header for body calculations. The Content-Length headers are informational in this case and do not describe the actual body length. (Naïve HTTP applications can get in trouble if they assume Content-Length always means there is a body).

   The most important example is the HEAD response. The HEAD method requests that a server send the headers that would have been returned by an equivalent GET request, but no body. Because a GET response would send back a Content-Length header, so will the HEAD response—but unlike the GET response, the HEAD response will not have a body. 1XX, 204, and 304 responses also can have informational Content-Length headers but no entity body. Messages that forbid entity bodies must terminate at the first empty line after the headers, regardless of which entity header fields are present.
2. If a message contains a Transfer-Encoding header (other than the default HTTP "identity" encoding), the entity will be terminated by a special pattern called a "zero-byte chunk," unless the message is terminated first by closing the connection. We'll discuss transfer encodings and chunked encodings later in this chapter.
3. If a message has a Content-Length header (and the message type allows entity bodies), the Content-Length value contains the body length, unless there is a non-identity Transfer-Encoding header. If a message is received with both a Content-Length header field and a non-identity Transfer-Encoding header field, you must ignore the Content-Length, because the transfer encoding will change the way entity bodies are represented and transferred (and probably the number of bytes transmitted).
4. If the message uses the "multipart/byteranges" media type and the entity length is not otherwise specified (in the Content-Length header), each part of the multipart message will specify its own size. This multipart type is the only entity body type that self-delimits its own size, so this media type must not be sent unless the sender knows the recipient can parse it.†

* Even the Content-MD5 header, which can be used to send the 128-bit MD5 of the document, contains the MD5 of the encoded document. The Content-MD5 header is described later in this chapter.

† Because a Range header might be forwarded by a more primitive proxy that does not understand multipart/byteranges, the sender must delimit the message using methods 1, 3, or 5 in this section if it isn't sure the receiver understands the self- delimiting format.

5. If none of the above rules match, the entity ends when the connection closes. In practice, only servers can use connection close to indicate the end of a message. Clients can't close the connection to signal the end of client messages, because that would leave no way for the server to send back a response.*

Entity Digests

Although HTTP typically is implemented over a reliable transport protocol such as TCP/IP, parts of messages may get modified in transit for a variety of reasons, such as noncompliant transcoding proxies or buggy intermediary proxies. To detect unintended (or undesired) modification of entity body data, the sender can generate a checksum of the data when the initial entity is generated, and the receiver can sanity check the checksum to catch any unintended entity modification.†

The Content-MD5 header is used by servers to send the result of running the MD5 algorithm on the entity body. Only the server where the response originates may compute and send the Content-MD5 header. Intermediate proxies and caches may not modify or add the header—that would violate the whole purpose of verifying end-to-end integrity. The Content-MD5 header contains the MD5 of the content after all content encodings have been applied to the entity body and before any transfer encodings have been applied to it. Clients seeking to verify the integrity of the message must first decode the transfer encodings, then compute the MD5 of the resulting unencoded entity body. As an example, if a document is compressed using the gzip algorithm, then sent with chunked encoding, the MD5 algorithm is run on the full gripped body.

In addition to checking message integrity, the MD5 can be used as a key into a hash table to quickly locate documents and reduce duplicate storage of content. Despite these possible uses, the Content-MD5 header is not sent often.

Extensions to HTTP have proposed other digest algorithms in IETF drafts. These extensions have proposed a new header, Want-Digest, that allows clients to specify the type of digest they expect with the response. Quality values can be used to suggest multiple digest algorithms and indicate preference.

* The client could do a half close of just its output connection, but many server applications aren't designed to handle this situation and will interpret a half close as the client disconnecting from the server. Connection management was never well specified in HTTP. See Chapter 4 for more details.

† This method, of course, is not immune to a malicious attack that replaces both the message body and digest header. It is intended only to detect unintentional modification. Other facilities, such as digest authentication, are needed to provide safeguards against malicious tampering.

Media Type and Charset

The Content-Type header field describes the MIME type of the entity body.* The MIME type is a standardized name that describes the underlying type of media carried as cargo (HTML file, Microsoft Word document, MPEG video, etc.). Client applications use the MIME type to properly decipher and process the content.

The Content-Type values are standardized MIME types, registered with the Internet Assigned Numbers Authority (IANA). MIME types consist of a primary media type (e.g., text, image, audio), followed by a slash, followed by a subtype that further specifies the media type. Table 15-1 lists a few common MIME types for the Content-Type header. More MIME types are listed in Appendix D.

Table 15-1. Common media types

Media type	Description
text/html	Entity body is an HTML document
text/plain	Entity body is a document in plain text
image/gif	Entity body is an image of type GIF
image/jpeg	Entity body is an image of type JPEG
audio/x-wav	Entity body contains WAV sound data
model/vrml	Entity body is a three-dimensional VRML model
application/vnd.ms-powerpoint	Entity body is a Microsoft PowerPoint presentation
multipart/byteranges	Entity body has multiple parts, each containing a different range (in bytes) of the full document
message/http	Entity body contains a complete HTTP message (see TRACE)

It is important to note that the Content-Type header specifies the media type of the original entity body. If the entity has gone through content encoding, for example, the Content-Type header will still specify the entity body type *before* the encoding.

Character Encodings for Text Media

The Content-Type header also supports optional parameters to further specify the content type. The "charset" parameter is the primary example, specifying the mechanism to convert bits from the entity into characters in a text file:

```
Content-Type: text/html; charset=iso-8859-4
```

We talk about character sets in detail in Chapter 16.

* In the case of the HEAD request, Content-Type shows the type that would have been sent if it was a GET request.

Multipart Media Types

MIME "multipart" email messages contain multiple messages stuck together and sent as a single, complex message. Each component is self-contained, with its own set of headers describing its content; the different components are concatenated together and delimited by a string.

HTTP also supports multipart bodies; however, they typically are sent in only one of two situations: in fill-in form submissions and in range responses carrying pieces of a document.

Multipart Form Submissions

When an HTTP fill-in form is submitted, variable-length text fields and uploaded objects are sent as separate parts of a multipart body, allowing forms to be filled out with values of different types and lengths. For example, you may choose to fill out a form that asks for your name and a description with your nickname and a small photo, while your friend may put down her full name and a long essay describing her passion for fixing Volkswagen buses.

HTTP sends such requests with a Content-Type: multipart/form-data header or a Content-Type: multipart/mixed header and a multipart body, like this:

```
Content-Type: multipart/form-data; boundary=[abcdefghijklmnopqrstuvwxyz]
```

where the boundary specifies the delimiter string between the different parts of the body.

The following example illustrates multipart/form-data encoding. Suppose we have this form:

```
<FORM action="http://server.com/cgi/handle"
      enctype="multipart/form-data"
      method="post">
<P>
What is your name? <INPUT type="text" name="submit-name"><BR>
What files are you sending? <INPUT type="file" name="files"><BR>
<INPUT type="submit" value="Send"> <INPUT type="reset">
</FORM>
```

If the user enters "Sally" in the text-input field and selects the text file "essayfile.txt," the user agent might send back the following data:

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Sally
--AaB03x
```

```
Content-Disposition: form-data; name="files"; filename="essayfile.txt"
Content-Type: text/plain
...contents of essayfile.txt...
--AaB03x--
```

If the user selected a second (image) file, "imagefile.gif," the user agent might construct the parts as follows:

```
Content-Type: multipart/form-data; boundary=AaB03x
--AaB03x
Content-Disposition: form-data; name="submit-name"
Sally
--AaB03x
Content-Disposition: form-data; name="files"
Content-Type: multipart/mixed; boundary=BbC04y
--BbC04y
Content-Disposition: file; filename="essayfile.txt"
Content-Type: text/plain
...contents of essayfile.txt...
--BbC04y
Content-Disposition: file; filename="imagefile.gif"
Content-Type: image/gif
Content-Transfer-Encoding: binary
...contents of imagefile.gif...
--BbC04y--
--AaB03x--
```

Multipart Range Responses

HTTP responses to range requests also can be multipart. Such responses come with a Content-Type: multipart/byteranges header and a multipart body with the different ranges. Here is an example of a multipart response to a request for different ranges of a document:

```
HTTP/1.0 206 Partial content
Server: Microsoft-IIS/5.0
Date: Sun, 10 Dec 2000 19:11:20 GMT
Content-Location: http://www.joes-hardware.com/gettysburg.txt
Content-Type: multipart/x-byteranges; boundary=--[abcdefghijklmnopqrstuvwxyz]--
Last-Modified: Sat, 09 Dec 2000 00:38:47 GMT

--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 0-174/1441

Fourscore and seven years ago our fathers brough forth on this continent
a new nation, conceived in liberty and dedicated to the proposition that
all men are created equal.
--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 552-761/1441
```

```
But in a larger sense, we can not dedicate, we can not consecrate,
we can not hallow this ground. The brave men, living and dead who
struggled here have consecrated it far above our poor power to add
or detract.
--[abcdefghijklmnopqrstuvwxyz]--
Content-Type: text/plain
Content-Range: bytes 1344-1441/1441

and that government of the people, by the people, for the people shall
not perish from the earth.

--[abcdefghijklmnopqrstuvwxyz]--
```

Range requests are discussed in more detail later in this chapter.

Content Encoding

HTTP applications sometimes want to encode content before sending it. For example, a server might compress a large HTML document before sending it to a client that is connected over a slow connection, to help lessen the time it takes to transmit the entity. A server might scramble or encrypt the contents in a way that prevents unauthorized third parties from viewing the contents of the document.

These types of encodings are applied to the content at the sender. Once the content is content-encoded, the encoded data is sent to the receiver in the entity body as usual.

The Content-Encoding Process

The content-encoding process is:

1. A web server generates an original response message, with original Content-Type and Content-Length headers.
2. A content-encoding server (perhaps the origin server or a downstream proxy) creates an encoded message. The encoded message has the same Content-Type but (if, for example, the body is compressed) a different Content-Length. The content-encoding server adds a Content-Encoding header to the encoded message, so that a receiving application can decode it.
3. A receiving program gets the encoded message, decodes it, and obtains the original.

Figure 15-3 sketches a content-encoding example.

Here, an HTML page is encoded by a gzip content-encoding function, to produce a smaller, compressed body. The compressed body is sent across the network, flagged

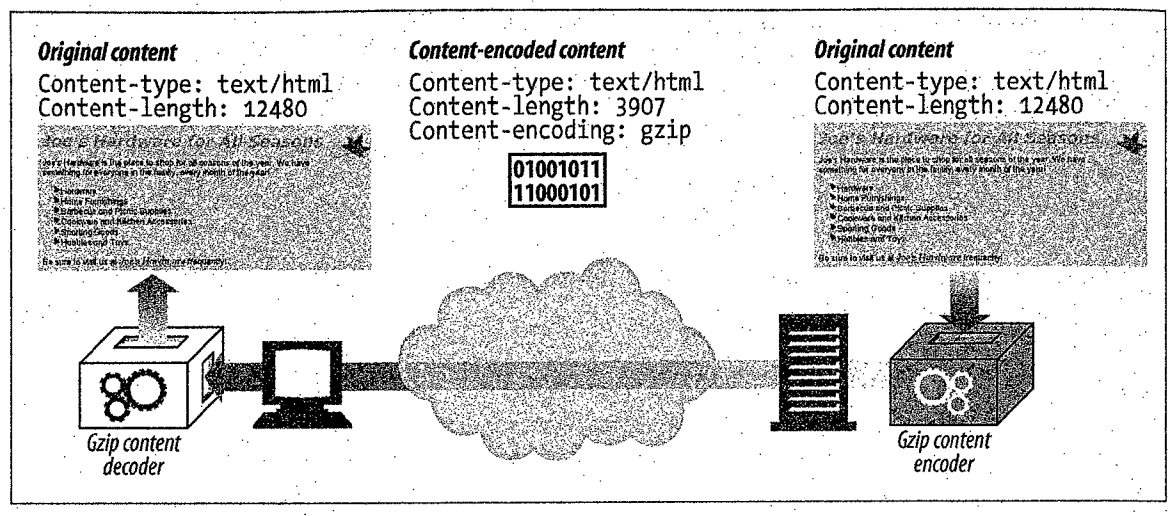


Figure 15-3. Content-encoding example

with the gzip encoding. The receiving client decompresses the entity using the gzip decoder.

This response snippet shows another example of an encoded response (a compressed image):

```
HTTP/1.1 200 OK
Date: Fri, 05 Nov 1999 22:35:15 GMT
Server: Apache/1.2.4
Content-Length: 6096
Content-Type: image/gif
Content-Encoding: gzip
[...]
```

Note that the Content-Type header can and should still be present in the message. It describes the original format of the entity—information that may be necessary for displaying the entity once it has been decoded. Remember that the Content-Length header now represents the length of the *encoded* body.

Content-Encoding Types

HTTP defines a few standard content-encoding types and allows for additional encodings to be added as extension encodings. Encodings are standardized through the IANA, which assigns a unique token to each content-encoding algorithm. The Content-Encoding header uses these standardized token values to describe the algorithm used in the encoding.

Some of the common content-encoding tokens are listed in Table 15-2.

Table 15-2. Content-encoding tokens

Content-encoding value	Description
gzip	Indicates that the GNU zip encoding was applied to the entity.[a]
compress	Indicates that the Unix file compression program has been run on the entity.
deflate	Indicates that the entity has been compressed into the zlib format.[b]
identity	Indicates that no encoding has been performed on the entity. When a Content-Encoding header is not present, this can be assumed.

[a] RFC 1952 describes the gzip encoding.
[b] RFCs 1950 and 1951 describe the zlib format and deflate compression.

The gzip, compress, and deflate encodings are lossless compression algorithms used to reduce the size of transmitted messages without loss of information. Of these, gzip typically is the most effective compression algorithm and is the most widely used.

Accept-Encoding Headers

Of course, we don't want servers encoding content in ways that the client can't decipher. To prevent servers from using encodings that the client doesn't support, the client passes along a list of supported content encodings in the Accept-Encoding request header. If the HTTP request does not contain an Accept-Encoding header, a server can assume that the client will accept any encoding (equivalent to passing Accept-Encoding: *).

Figure 15-4 shows an example of Accept-Encoding in an HTTP transaction.

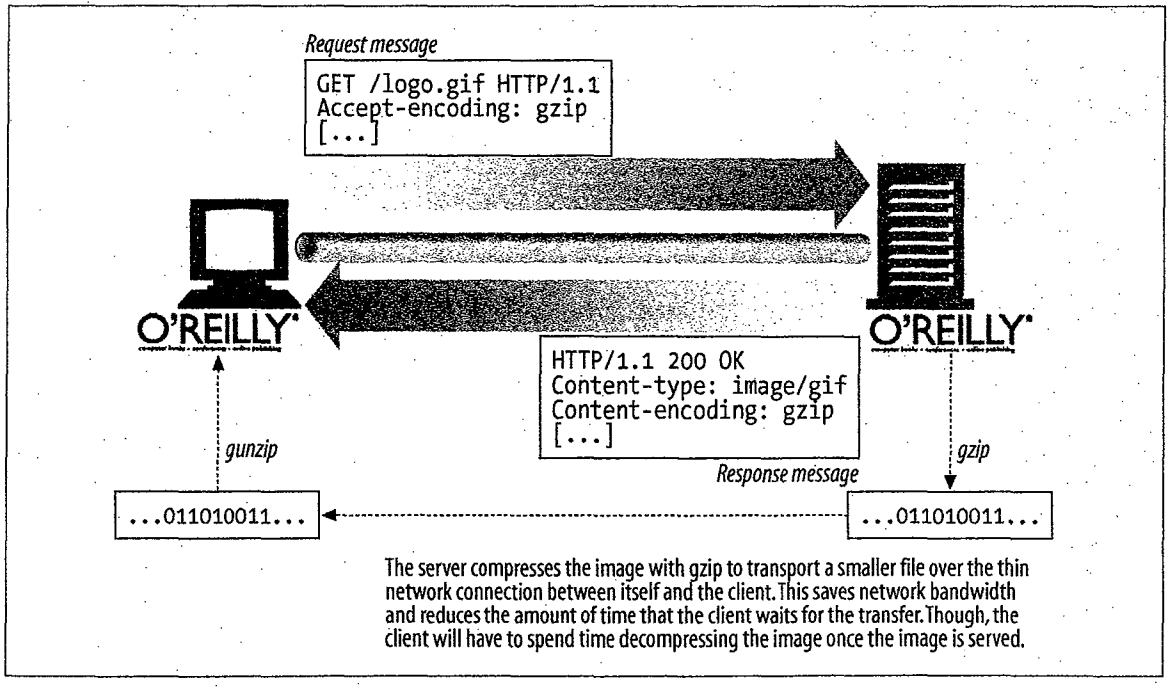


Figure 15-4. Content encoding

The Accept-Encoding field contains a comma-separated list of supported encodings. Here are a few examples:

```
Accept-Encoding: compress, gzip
Accept-Encoding:
Accept-Encoding: *
Accept-Encoding: compress;q=0.5, gzip;q=1.0
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

Clients can indicate preferred encodings by attaching Q (quality) values as parameters to each encoding. Q values can range from 0.0, indicating that the client does not want the associated encoding, to 1.0, indicating the preferred encoding. The token "*" means "anything else." The process of selecting which content encoding to apply is part of a more general process of deciding which content to send back to a client in a response. This process and the Content-Encoding and Accept-Encoding headers are discussed in more detail in Chapter 17.

The identity encoding token can be present only in the Accept-Encoding header and is used by clients to specify relative preference over other content-encoding algorithms.

Transfer Encoding and Chunked Encoding

The previous section discussed *content* encodings—reversible transformations applied to the body of the message. Content encodings are tightly associated with the details of the particular content format. For example, you might compress a text file with gzip, but not a JPEG file, because JPEGs don't compress well with gzip.

This section discusses *transfer* encodings. Transfer encodings also are reversible transformations performed on the entity body, but they are applied for architectural reasons and are independent of the format of the content. You apply a transfer encoding to a message to change the way message data is transferred across the network (Figure 15-5).

Safe Transport

Historically, transfer encodings exist in other protocols to provide "safe transport" of messages across a network. The concept of safe transport has a different focus for HTTP, where the transport infrastructure is standardized and more forgiving. In HTTP, there are only a few reasons why transporting message bodies can cause trouble. Two of these are:

Unknown size

Some gateway applications and content encoders are unable to determine the final size of a message body without generating the content first. Often, these servers would like to start sending the data before the size is known. Because

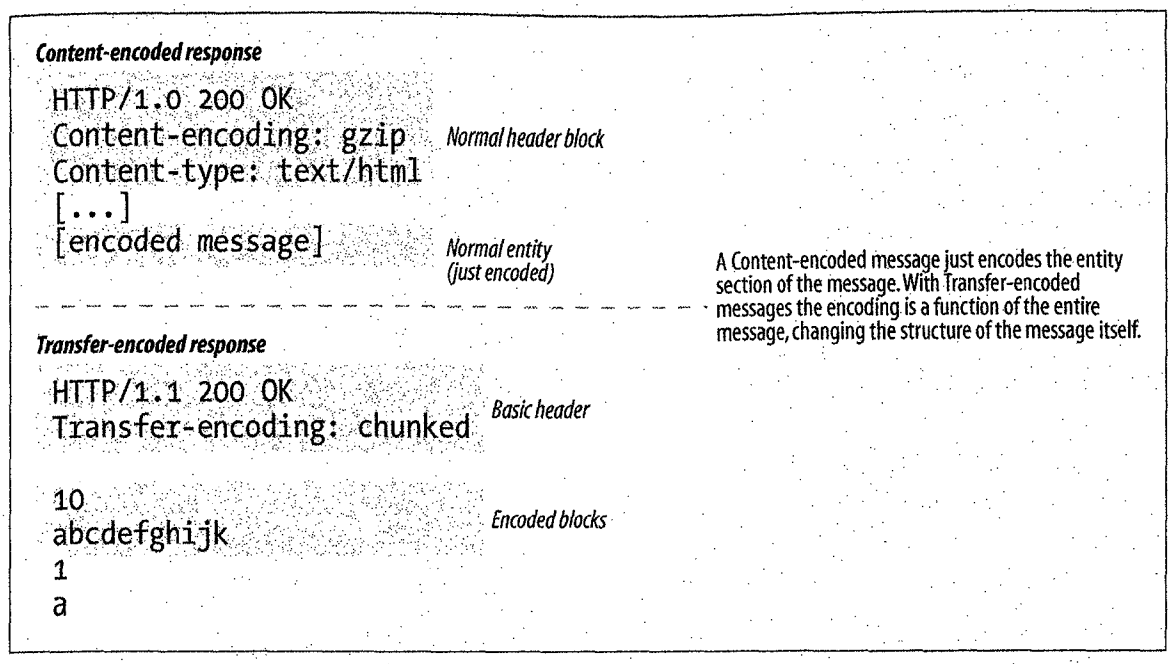


Figure 15-5. Content encodings versus transfer encodings

HTTP requires the Content-Length header to precede the data, some servers apply a transfer encoding to send the data with a special terminating footer that indicates the end of data.*

Security

You might use a transfer encoding to scramble the message content before sending it across a shared transport network. However, because of the popularity of transport layer security schemes like SSL, transfer-encoding security isn't very common.

Transfer-Encoding Headers

There are just two defined headers to describe and control transfer encoding:

Transfer-Encoding

Tells the receiver what encoding has been performed on the message in order for it to be safely transported

TE

Used in the request header to tell the server what extension transfer encodings are okay to use†

* You could close the connection as a "poor man's" end-of-message signal, but this breaks persistent connections.

† The meaning of the TE header would be more intuitive if it were called the Accept-Transfer-Encoding header.

In the following example, the request uses the TE header to tell the server that it accepts the chunked encoding (which it must if it's an HTTP 1.1 application) and is willing to accept trailers on the end of chunk-encoded messages:

```
GET /new_products.html HTTP/1.1
Host: www.joes-hardware.com
User-Agent: Mozilla/4.61 [en] (WinNT; I)
TE: trailers, chunked
...
```

The response includes a Transfer-Encoding header to tell the receiver that the message has been transfer-encoded with the chunked encoding:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Server: Apache/3.0
...
```

After this initial header, the structure of the message will change.

All transfer-encoding values are case-insensitive. HTTP/1.1 uses transfer-encoding values in the TE header field and in the Transfer-Encoding header field. The latest HTTP specification defines only one transfer encoding, chunked encoding.

The TE header, like the Accept-Encoding header, can have Q values to describe preferred forms of transfer encoding. The HTTP/1.1 specification, however, forbids the association of a Q value of 0.0 to chunked encoding.

Future extensions to HTTP may drive the need for additional transfer encodings. If and when this happens, the chunked transfer encoding should always be applied on top of the extension transfer encodings. This guarantees that the data will get "tunneled" through HTTP/1.1 applications that understand chunked encoding but not other transfer encodings.

Chunked Encoding

Chunked encoding breaks messages into chunks of known size. Each chunk is sent one after another, eliminating the need for the size of the full message to be known before it is sent.

Note that chunked encoding is a form of transfer encoding and therefore is an attribute of the message, not the body. Multipart encoding, described earlier in this chapter, is an attribute of the body and is completely separate from chunked encoding.

Chunking and persistent connections

When the connection between the client and server is not persistent, clients do not need to know the size of the body they are reading—they expect to read the body until the server closes the connection.

With persistent connections, the size of the body must be known and sent in the Content-Length header before the body can be written. When content is dynamically created at a server, it may not be possible to know the length of the body before sending it.

Chunked encoding provides a solution for this dilemma, by allowing servers to send the body in chunks, specifying only the size of each chunk. As the body is dynamically generated, a server can buffer up a portion of it, send its size and the chunk, and then repeat the process until the full body has been sent. The server can signal the end of the body with a chunk of size 0 and still keep the connection open and ready for the next response.

Chunked encoding is fairly simple. Figure 15-6 shows the basic anatomy of a chunked message. It begins with an initial HTTP response header block, followed by a stream of chunks. Each chunk contains a length value and the data for that chunk. The length value is in hexadecimal form and is separated from the chunk data with a CRLF. The size of the chunk data is measured in bytes and includes neither the CRLF sequence between the length value and the data nor the CRLF sequence at the end of the chunk. The last chunk is special—it has a length of zero, which signifies "end of body."

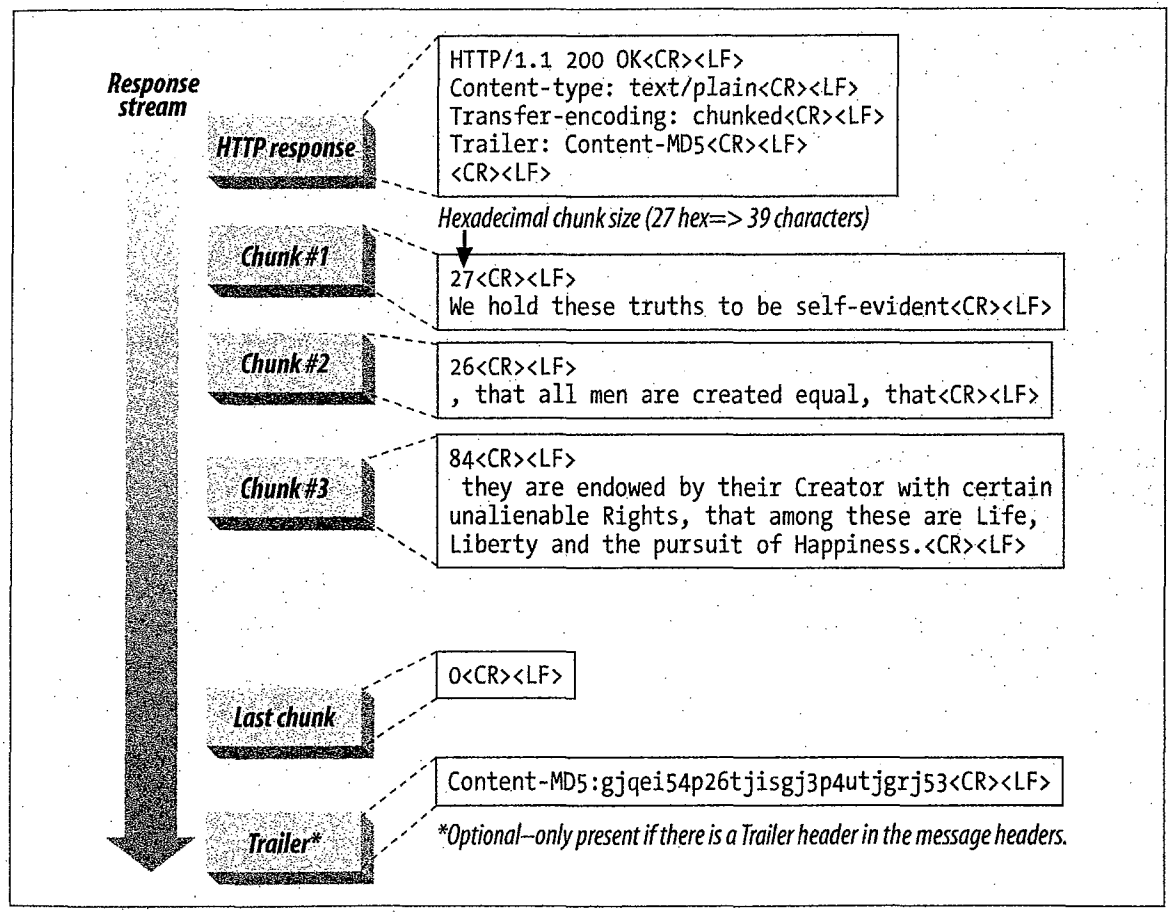


Figure 15-6. Anatomy of a chunked message

A client also may send chunked data to a server. Because the client does not know beforehand whether the server accepts chunked encoding (servers do not send TE headers in responses to clients), it must be prepared for the server to reject the chunked request with a 411 Length Required response.

Trailers in chunked messages

A trailer can be added to a chunked message if the client's TE header indicates that it accepts trailers, or if the trailer is added by the server that created the original response and the contents of the trailer are optional metadata that it is not necessary for the client to understand and use (it is okay for the client to ignore and discard the contents of the trailer).*

The trailer can contain additional header fields whose values might not have been known at the start of the message (e.g., because the contents of the body had to be generated first). An example of a header that can be sent in the trailer is the Content-MD5 header—it would be difficult to calculate the MD5 of a document before the document has been generated. Figure 15-6 illustrates the use of trailers. The message headers contain a Trailer header listing the headers that will follow the chunked message. The last chunk is followed by the headers listed in the Trailer header.

Any of the HTTP headers can be sent as trailers, except for the Transfer-Encoding, Trailer, and Content-Length headers.

Combining Content and Transfer Encodings

Content encoding and transfer encoding can be used simultaneously. For example, Figure 15-7 illustrates how a sender can compress an HTML file using a content encoding and send the data chunked using a transfer encoding. The process to "reconstruct" the body is reversed on the receiver.

Transfer-Encoding Rules

When a transfer encoding is applied to a message body, a few rules must be followed:

- The set of transfer encodings must include "chunked." The only exception is if the message is terminated by closing the connection.
- When the chunked transfer encoding is used, it is required to be the last transfer encoding applied to the message body.
- The chunked transfer encoding must not be applied to a message body more than once.

* The Trailer header was added after the initial chunked encoding was added to drafts of the HTTP/1.1 specification, so some applications may not understand it (or understand trailers) even if they claim to be HTTP/1.1-compliant.

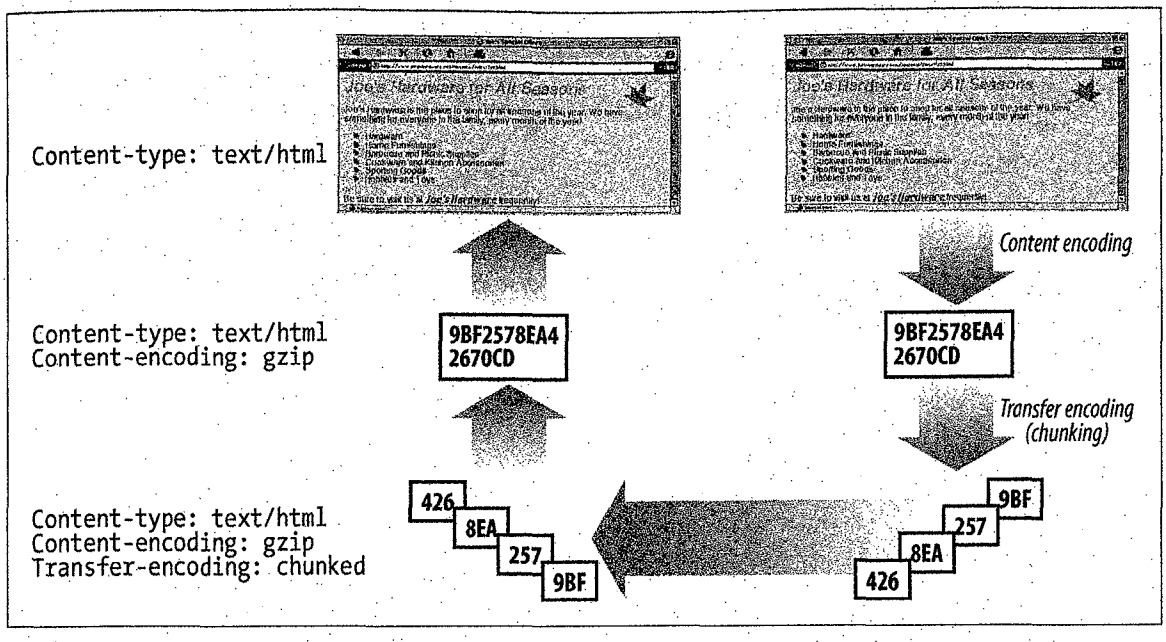


Figure 15-7. Combining content encoding with transfer encoding

These rules allow the recipient to determine the transfer length of the message.

Transfer encodings are a relatively new feature of HTTP, introduced in Version 1.1. Servers that implement transfer encodings need to take special care not to send transfer-encoded messages to non-HTTP/1.1 applications. Likewise, if a server receives a transfer-encoded message that it can not understand, it should respond with the 501 Unimplemented status code. However, all HTTP/1.1 applications must at least support chunked encoding.

Time-Varying Instances

Web objects are not static. The same URL can, over time, point to different versions of an object. Take the CNN home page as an example—going to "http://www.cnn.com" several times in a day is likely to result in a slightly different page being returned each time.

Think of the CNN home page as being an object and its different versions as being different *instances* of the object (see Figure 15-8). The client in the figure requests the same resource (URL) multiple times, but it gets different instances of the resource as it changes over time. At time (a) and (b) it has the same instance; at time (c) it has a different instance.

The HTTP protocol specifies operations for a class of requests and responses, called *instance manipulations*, that operate on instances of an object. The two main instance-manipulation methods are range requests and delta encoding. Both of these methods require clients to be able to identify the exact copy of the resource that they have (if any) and request new instances conditionally. These mechanisms are discussed later in this chapter.

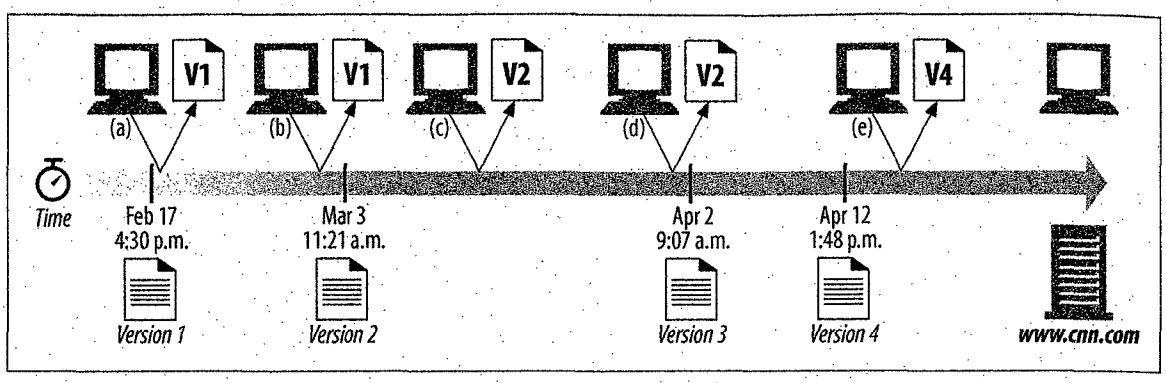


Figure 15-8. Instances are "snapshots" of a resource in time

Validators and Freshness

Look back at Figure 15-8. The client does not initially have a copy of the resource, so it sends a request to the server asking for it. The server responds with Version 1 of the resource. The client can now cache this copy, but for how long?

Once the document has "expired" at the client (i.e., once the client can no longer consider its copy a valid copy), it must request a fresh copy from the server. If the document has not changed at the server, however, the client does not need to receive it again—it can just continue to use its cached copy.

This special request, called a *conditional request*, requires that the client tell the server which version it currently has, using a *validator*, and ask for a copy to be sent only if its current copy is no longer valid. Let's look at the three key concepts—freshness, validators, and conditionals—in more detail.

Freshness

Servers are expected to give clients information about how long clients can cache their content and consider it fresh. Servers can provide this information using one of two headers: Expires and Cache-Control.

The Expires header specifies the exact date and time when the document "expires"—when it can no longer be considered fresh. The syntax for the Expires header is:

```
Expires: Sun Mar 18 23:59:59 GMT 2001
```

For a client and server to use the Expires header correctly, their clocks must be synchronized. This is not always easy, because neither may run a clock synchronization protocol such as the Network Time Protocol (NTP). A mechanism that defines expiration using relative time is more useful. The Cache-Control header can be used to specify the maximum age for a document in seconds—the total amount of time since the document left the server. Age is not dependent on clock synchronization and therefore is likely to yield more accurate results.

The Cache-Control header actually is very powerful. It can be used by both servers and clients to describe freshness using more directives than just specifying an age or expiration time. Table 15-3 lists some of the directives that can accompany the Cache-Control header.

Table 15-3. Cache-Control header directives

Directive	Message type	Description
no-cache	Request	Do not return a cached copy of the document without first revalidating it with the server.
no-store	Request	Do not return a cached copy of the document. Do not store the response from the server.
max-age	Request	The document in the cache must not be older than the specified age.
max-stale	Request	The document may be stale based on the server-specified expiration information, but it must not have been expired for longer than the value in this directive.
min-fresh	Request	The document's age must not be more than its age plus the specified amount. In other words, the response must be fresh for at least the specified amount of time.
no-transform	Request	The document must not be transformed before being sent.
only-if-cached	Request	Send the document only if it is in the cache, without contacting the origin server.
public	Response	Response may be cached by any cache.
private	Response	Response may be cached such that it can be accessed only by a single client.
no-cache	Response	If the directive is accompanied by a list of header fields, the content may be cached and served to clients, but the listed header fields must first be removed. If no header fields are specified, the cached copy must not be served without revalidation with the server.
no-store	Response	Response must not be cached.
no-transform	Response	Response must not be modified in any way before being served.
must-revalidate	Response	Response must be revalidated with the server before being served.
proxy-revalidate	Response	Shared caches must revalidate the response with the origin server before serving. This directive can be ignored by private caches.
max-age	Response	Specifies the maximum length of time the document can be cached and still considered fresh.
s-max-age	Response	Specifies the maximum age of the document as it applies to shared caches (overriding the max-age directive, if one is present). This directive can be ignored by private caches.

Caching and freshness were discussed in more detail in Chapter 7.

Conditionals and Validators

When a cache's copy is requested, and it is no longer fresh, the cache needs to make sure it has a fresh copy. The cache can fetch the current copy from the origin server, but in many cases, the document on the server is still the same as the stale copy in the cache. We saw this in Figure 15-8b; the cached copy may have expired, but the

server content still is the same as the cache content. If a cache always fetches a server's document, even if it's the same as the expired cache copy, the cache wastes network bandwidth, places unnecessary load on the cache and server, and slows everything down.

To fix this, HTTP provides a way for clients to request a copy *only if the resource has changed*, using special requests called *conditional requests*. Conditional requests are normal HTTP request messages, but they are performed only if a particular condition is true. For example, a cache might send the following conditional GET message to a server, asking it to send the file */announce.html* only if the file has been modified since June 29, 2002 (the date the cached document was last changed by the author):

```
GET /announce.html HTTP/1.0
If-Modified-Since: Sat, 29 Jun 2002, 14:30:00 GMT
```

Conditional requests are implemented by conditional headers that start with "If-". In the example above, the conditional header is If-Modified-Since. A conditional header allows a method to execute only if the condition is true. If the condition is not true, the server sends an HTTP error code back.

Each conditional works on a particular *validator*. A validator is a particular attribute of the document instance that is tested. Conceptually, you can think of the validator like the serial number, version number, or last change date of a document. A wise client in Figure 15-8b would send a conditional validation request to the server saying, "send me the resource only if it is no longer Version 1; I have Version 1." We discussed conditional cache revalidation in Chapter 7, but we'll study the details of entity validators more carefully in this chapter.

The If-Modified-Since conditional header tests the last-modified date of a document instance, so we say that the last-modified date is the validator. The If-None-Match conditional header tests the ETag value of a document, which is a special keyword or version-identifying tag associated with the entity. Last-Modified and ETag are the two primary validators used by HTTP. Table 15-4 lists four of the HTTP headers used for conditional requests. Next to each conditional header is the type of validator used with the header.

Table 15-4. Conditional request types

Request type	Validator	Description
If-Modified-Since	Last-Modified	Send a copy of the resource if the version that was last modified at the time in your previous Last-Modified response header is no longer the latest one.
If-Unmodified-Since	Last-Modified	Send a copy of the resource only if it is the same as the version that was last modified at the time in your previous Last-Modified response header.
If-Match	ETag	Send a copy of the resource if its entity tag is the same as that of the one in your previous ETag response header.
If-None-Match	ETag	Send a copy of the resource if its entity tag is different from that of the one in your previous ETag response header.

HTTP groups validators into two classes: *weak validators* and *strong validators*. Weak validators may not always uniquely identify an instance of a resource; strong validators must. An example of a weak validator is the size of the object in bytes. The resource content might change even though the size remains the same, so a hypothetical byte-count validator only weakly indicates a change. A cryptographic checksum of the contents of the resource (such as MD5), however, is a strong validator; it changes when the document changes.

The last-modified time is considered a weak validator because, although it specifies the time at which the resource was last modified, it specifies that time to an accuracy of at most one second. Because a resource can change multiple times in a second, and because servers can serve thousands of requests per second, the last-modified date might not always reflect changes. The ETag header is considered a strong validator, because the server can place a distinct value in the ETag header every time a value changes. Version numbers and digest checksums are good candidates for the ETag header, but they can contain any arbitrary text. ETag headers are flexible; they take arbitrary text values ("tags"), and can be used to devise a variety of client and server validation strategies.

Clients and servers may sometimes want to adopt a looser version of entity-tag validation. For example, a server may want to make cosmetic changes to a large, popular cached document without triggering a mass transfer when caches revalidate. In this case, the server might advertise a "weak" entity tag by prefixing the tag with "W/". A weak entity tag should change only when the associated entity changes in a semantically significant way. A strong entity tag must change whenever the associated entity value changes in any way.

The following example shows how a client might revalidate with a server using a weak entity tag. The server would return a body only if the content changed in a meaningful way from Version 4.0 of the document:

```
GET /announce.html HTTP/1.1
If-None-Match: W/"v4.0"
```

In summary, when clients access the same resource more than once, they first need to determine whether their current copy still is fresh. If it is not, they must get the latest version from the server. To avoid receiving an identical copy in the event that the resource has not changed, clients can send conditional requests to the server, specifying validators that uniquely identify their current copies. Servers will then send a copy of the resource only if it is different from the client's copy. For more details on cache revalidation, please refer back to "Cache Processing Steps" in Chapter 7.

Range Requests

We now understand how a client can ask a server to send it a resource only if the client's copy of the resource is no longer valid. HTTP goes further: it allows clients to actually request just part or a range of a document.

Imagine if you were three-fourths of the way through downloading the latest hot software across a slow modem link, and a network glitch interrupted your connection. You would have been waiting for a while for the download to complete, and now you would have to start all over again, hoping the same thing does not happen again.

With range requests, an HTTP client can resume downloading an entity by asking for the range or part of the entity it failed to get (provided that the object did not change at the origin server between the time the client first requested it and its subsequent range request). For example:

```
GET /bigfile.html HTTP/1.1
Host: www.joes-hardware.com
Range: bytes=4000-
User-Agent: Mozilla/4.61 [en] (WinNT; I)
...
```

In this example, the client is requesting the remainder of the document after the first 4,000 bytes (the end bytes do not have to be specified, because the size of the document may not be known to the requestor). Range requests of this form can be used for a failed request where the client received the first 4,000 bytes before the failure. The Range header also can be used to request multiple ranges (the ranges can be specified in any order and may overlap)—for example, imagine a client connecting to multiple servers simultaneously, requesting different ranges of the same document from different servers in order to speed up overall download time for the document. In the case where clients request multiple ranges in a single request, responses come back as a single entity, with a multipart body and a Content-Type: multipart/byteranges header.

Not all servers accept range requests, but many do. Servers can advertise to clients that they accept ranges by including the header Accept-Ranges in their responses. The value of this header is the unit of measure, usually bytes.* For example:

```
HTTP/1.1 200 OK
Date: Fri, 05 Nov 1999 22:35:15 GMT
Server: Apache/1.2.4
Accept-Ranges: bytes
...
```

Figure 15-9 shows an example of a set of HTTP transactions involving ranges.

Range headers are used extensively by popular peer-to-peer file-sharing client software to download different parts of multimedia files simultaneously, from different peers.

Note that range requests are a class of instance manipulations, because they are exchanges between a client and a server for a particular instance of an object. That is, a client's range request makes sense only if the client and server have the same version of a document.

* The HTTP/1.1 specification defines only the bytes token, but server and client implementors could come up with their own units to measure or chop up an entity.

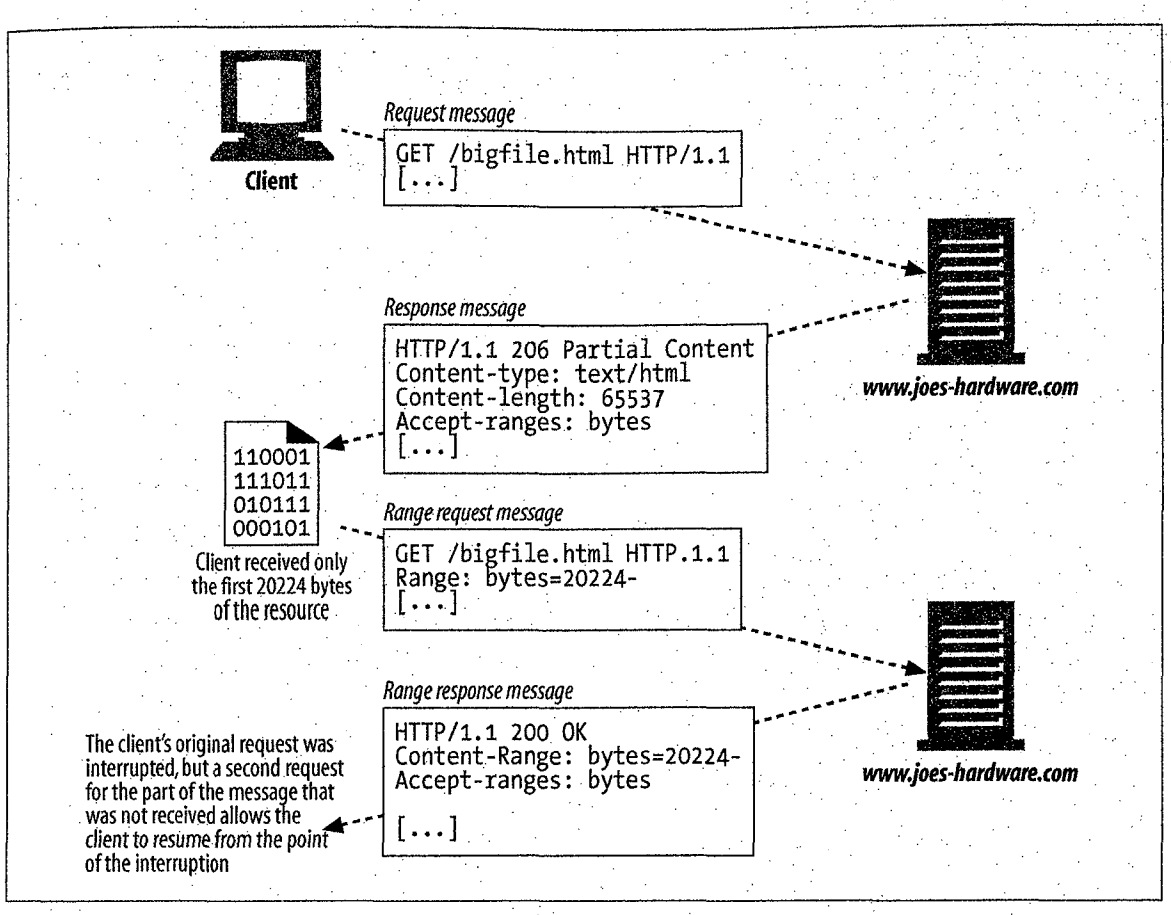


Figure 15-9. Entity range request example

Delta Encoding

We have described different versions of a web page as different instances of a page. If a client has an expired copy of a page, it requests the latest instance of the page. If the server has a newer instance of the page, it will send it to the client, and it will send the full new instance of the page even if only a small portion of the page actually has changed.

Rather than sending it the entire new page, the client would get the page faster if the server sent just the changes to the client's copy of the page (provided that the number of changes is small). Delta encoding is an extension to the HTTP protocol that optimizes transfers by communicating changes instead of entire objects. Delta encoding is a type of instance manipulation, because it relies on clients and servers exchanging information about particular instances of an object. RFC 3229 describes delta encoding.

Figure 15-10 illustrates more clearly the mechanism of requesting, generating, receiving, and applying a delta-encoded document. The client has to tell the server which version of the page it has, that it is willing to accept a *delta* from the latest version of page, and which algorithms it knows for applying those deltas to its current version.

The server has to check if it has the client's version of the page and how to compute deltas from the latest version and the client's version (there are several algorithms for computing the difference between two objects). It then has to compute the delta, send it to the client, let the client know that it's sending a delta, and specify the new identifier for the latest version of the page (because this is the version that the client will end up with after it applies the delta to its old version).

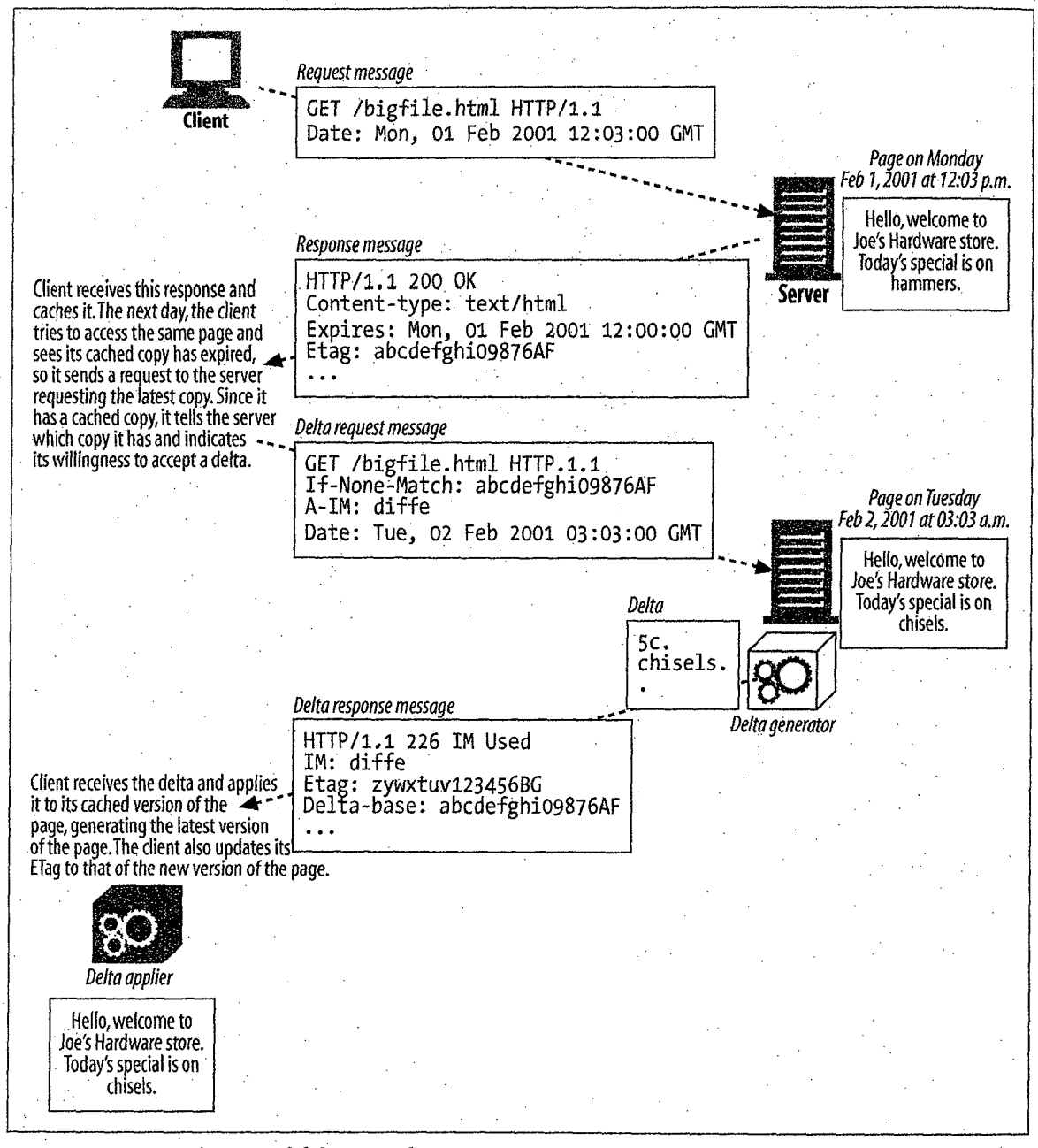


Figure 15-10. Mechanics of delta-encoding

The client uses the unique identifier for its version of the page (sent by the server in its previous response to the client in the ETag header) in an If-None-Match header. This is the client's way of telling the server, "if the latest version of the page you have

does not have this same ETag, send me the latest version of the page." Just the If-None-Match header, then, would cause the server to send the client the full latest version of the page (if it was different from the client's version).

The client can tell the server, however, that it is willing to accept a delta of the page by also sending an A-IM header. A-IM is short for Accept-Instance-Manipulation ("Oh, by the way, I do accept some forms of instance manipulation, so if you apply one of those you will not have to send me the full document."). In the A-IM header, the client specifies the algorithms it knows how to apply in order to generate the latest version of a page given an old version and a delta. The server sends back the following: a special response code (226 IM Used) telling the client that it is sending it an instance manipulation of the requested object, not the full object itself; an IM (short for Instance-Manipulation) header, which specifies the algorithm used to compute the delta; the new ETag header; and a Delta-Base header, which specifies the ETag of the document used as the base for computing the delta (ideally, the same as the ETag in the client's If-None-Match request!). The headers used in delta encoding are summarized in Table 15-5.

Table 15-5. Delta-encoding headers

Header	Description
ETag	Unique identifier for each instance of a document. Sent by the server in the response; used by clients in subsequent requests in If-Match and If-None-Match headers.
If-None-Match	Request header sent by the client, asking the server for a document if and only if the client's version of the document is different from the server's.
A-IM	Client request header indicating types of instance manipulations accepted.
IM	Server response header specifying the type of instance manipulation applied to the response. This header is sent when the response code is 226 IM Used.
Delta-Base	Server response header that specifies the ETag of the base document used for generating the delta (should be the same as the ETag in the client request's If-None-Match header).

Instance Manipulations, Delta Generators, and Delta Appliers

Clients can specify the types of instance manipulation they accept using the A-IM header. Servers specify the type of instance manipulation used in the IM header. Just what are the types of instance manipulation that are accepted, and what do they do? Table 15-6 lists some of the IANA registered types of instance manipulations.

Table 15-6. IANA registered types of instance manipulations

Type	Description
vcdiff	Delta using the vcdiff algorithm[a]
diffe	Delta using the Unix *diff -e* command
gdiff	Delta using the gdiff algorithm[b]

Table 15-6. IANA registered types of instance manipulations (continued)

Type	Description
gzip	Compression using the gzip algorithm
deflate	Compression using the deflate algorithm
range	Used in a server response to indicate that the response is partial content as the result of a range selection
identity	Used in a client request's A-IM header to indicate that the client is willing to accept an identity instance manipulation

[a] Internet draft *draft-korn-vcdiff-01* describes the vcdiff algorithm. This specification was approved by the IESG in early 2002 and should be released in RFC form shortly.

[b] *http://www.w3org/TR/NOTE-gdiff-19970901.html* describes the GDIFF algorithm.

A "delta generator" at the server, as in Figure 15-10, takes the base document and the latest instance of the document and computes the delta between the two using the algorithm specified by the client in the A-IM header. At the client side, a "delta applier" takes the delta and applies it to the base document to generate the latest instance of the document. For example, if the algorithm used to generate the delta is the Unix *diff -e* command, the client can apply the delta using the functionality of the Unix *ed* text editor, because *diff -e <file1> <file2>* generates the set of *ed* commands that will convert *<file1>* into *<file2>*. *ed* is a very simple editor with a few supported commands. In the example in Figure 15-10, *5c* says delete line 5 in the base document, and *chisels.<cr>*. says add "chisels.". That's it. More complicated instructions can be generated for bigger changes. The Unix *diff -e* algorithm does a line-by-line comparison of files. This obviously is okay for text files but breaks down for binary files. The vcdiff algorithm is more powerful, working even for non-text files and generally producing smaller deltas than *diff -e*.

The delta encoding specification defines the format of the A-IM and IM headers in detail. Suffice it to say that multiple instance manipulations can be specified in these headers (along with corresponding quality values). Documents can go through multiple instance manipulations before being returned to clients, in order to maximize compression. For example, deltas generated by the vcdiff algorithm may in turn be compressed using the gzip algorithm. The server response would then contain the header IM: vcdiff, gzip. The client would first gunzip the content, then apply the results of the delta to its base page in order to generate the final document.

Delta encoding can reduce transfer times, but it can be tricky to implement. Imagine a page that changes frequently and is accessed by many different people. A server supporting delta encoding must keep all the different copies of that page as it changes over time, in order to figure out what's changed between any requesting client's copy and the latest copy. (If the document changes frequently, as different clients request the document, they will get different instances of the document. When they make subsequent requests to the server, they will be requesting changes between their instance of the document and the latest instance of the document. To be able to send them just the changes, the server must keep copies of all the previous

instances that the clients have.) In exchange for reduced latency in serving documents, servers need to increase disk space to keep old instances of documents around. The extra disk space necessary to do so may quickly negate the benefits from the smaller transfer amounts.

For More Information

For more information on entities and encodings, see:

http://www.ietf.org/rfc/rfc2616.txt
: The HTTP/1.1 specification, RFC 2616, is the primary reference for entity body management and encodings.

http://www.ietf.org/rfc/rfc3229.txt
: RFC 3229, "Delta Encoding in HTTP," describes how delta encoding can be supported as an extension to HTTP/1.1.

Introduction to Data Compression
: Khalid Sayood, Morgan Kaufmann Publishers. This book explains some of the compression algorithms supported by HTTP content encodings.

http://www.ietf.org/rfc/rfc1521.txt
: RFC 1521, "Multipurpose Internet Mail Extensions, Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," describes the format of MIME bodies. This reference material is useful because HTTP borrows heavily from MIME. In particular, this document is designed to provide facilities to include multiple objects in a single message, to represent body text in character sets other than US-ASCII, to represent formatted multi-font text messages, and to represent nontextual material such as images and audio fragments.

http://www.ietf.org/rfc/rfc2045.txt
: RFC 2045, "Multipurpose Internet Mail Extensions, Part One: Format of Internet Message Bodies," specifies the various headers used to describe the structure of MIME messages, many of which are similar or identical to HTTP.

http://www.ietf.org/rfc/rfc1864.txt
: RFC 1864, "The Content-MD5 Header Field," provides some historical detail about the behavior and intended use of the Content-MD5 header field in MIME content as a message integrity check.

http://www.ietf.org/rfc/rfc3230.txt
: RFC 3230, "Instance Digests in HTTP," describes improvements to HTTP entity-digest handling that fix weaknesses present in the Content-MD5 formulation.

CHAPTER 16

Internationalization

Every day, billions of people write documents in hundreds of languages. To live up to the vision of a truly world-wide Web, HTTP needs to support the transport and processing of international documents, in many languages and alphabets.

This chapter covers two primary internationalization issues for the Web: *character set encodings* and *language tags*. HTTP applications use character set encodings to request and display text in different alphabets, and they use language tags to describe and restrict content to languages the user understands. We finish with a brief chat about multilingual URIs and dates.

This chapter:

- Explains how HTTP interacts with schemes and standards for multilingual alphabets
- Gives a rapid overview of the terminology, technology, and standards to help HTTP programmers do things right (readers familiar with character encodings can skip this section)
- Explains the standard naming system for languages, and how standardized language tags describe and select content
- Outlines rules and cautions for international URIs
- Briefly discusses rules for dates and other internationalization issues

HTTP Support for International Content

HTTP messages can carry content in any language, just as it can carry images, movies, or any other kind of media. To HTTP, the entity body is just a box of bits.

To support international content, servers need to tell clients about the alphabet and languages of each document, so the client can properly unpack the document bits into characters and properly process and present the content to the user.

Servers tell clients about a document's alphabet and language with the HTTP Content-Type charset parameter and Content-Language headers. These headers describe what's in the entity body's "box of bits," how to convert the contents into the proper characters that can be displayed onscreen, and what spoken language the words represent.

At the same time, the client needs to tell the server which languages the user understands and which alphabetic coding algorithms the browser has installed. The client sends Accept-Charset and Accept-Language headers to tell the server which character set encoding algorithms and languages the client understands, and which of them are preferred.

The following HTTP Accept headers might be sent by a French speaker who prefers his native language (but speaks some English in a pinch) and who uses a browser that supports the iso-8859-1 West European charset encoding and the UTF-8 Unicode charset encoding:

```
Accept-Language: fr, en;q=0.8
Accept-Charset: iso-8859-1, utf-8
```

The parameter "q=0.8" is a *quality factor*, giving lower priority to English (0.8) than to French (1.0 by default).

Character Sets and HTTP

So, let's jump right into the most important (and confusing) aspects of web internationalization—international alphabetic scripts and their character set encodings.

Web character set standards can be pretty confusing. Lots of people get frustrated when they first try to write international web software, because of complex and inconsistent terminology, standards documents that you have to pay to read, and unfamiliarity with foreign languages. This section and the next section should make it easier for you to use character sets with HTTP.

Charset Is a Character-to-Bits Encoding

The HTTP charset values tell you how to convert from entity content bits into characters in a particular alphabet. Each charset tag names an algorithm to translate bits to characters (and vice versa). The charset tags are standardized in the MIME character set registry, maintained by the IANA (see *http://www.iana.org/assignments/character-sets*). Appendix H summarizes many of them.

The following Content-Type header tells the receiver that the content is an HTML file, and the charset parameter tells the receiver to use the iso-8859-6 Arabic character set decoding scheme to decode the content bits into characters:

```
Content-Type: text/html; charset=iso-8859-6
```

The iso-8859-6 encoding scheme maps 8-bit values into both the Latin and Arabic alphabets, including numerals, punctuation and other symbols.* For example, in Figure 16-1, the highlighted bit pattern has code value 225, which (under iso-8859-6) maps into the Arabic letter "FEH" (a sound like the English letter "F").

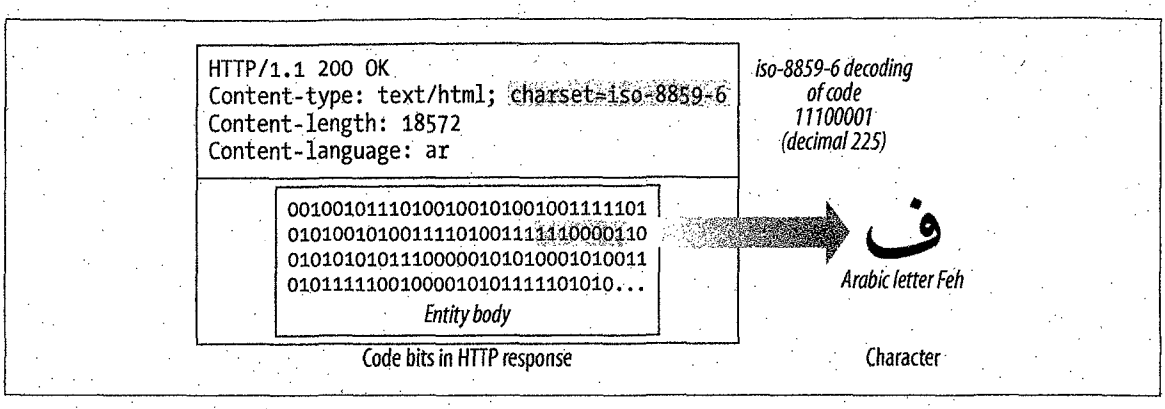


Figure 16-1. The charset parameter tells the client how to go from bits to characters

Some character encodings (e.g., UTF-8 and iso-2022-jp) are more complicated, *variable-length* codes, where the number of bits per character varies. This type of coding lets you use extra bits to support alphabets with large numbers of characters (such as Chinese and Japanese), while using fewer bits to support standard Latin characters.

How Character Sets and Encodings Work

Let's see what character sets and encodings really do.

We want to convert from bits in a document into characters that we can display onscreen. But because there are many different alphabets, and many different ways of encoding characters into bits (each with advantages and disadvantages), we need a standard way to describe and apply the bits-to-character decoding algorithm.

Bits-to-character conversions happen in two steps, as shown in Figure 16-2:

- In Figure 16-2a, bits from a document are converted into a character code that identifies a particular numbered character in a particular coded character set. In the example, the decoded character code is numbered 225.
- In Figure 16-2b, the character code is used to select a particular element of the coded character set. In iso-8859-6, the value 225 corresponds to "ARABIC LETTER FEH." The algorithms used in Steps a and b are determined from the MIME charset tag.

A key goal of internationalized character systems is the isolation of the semantics (letters) from the presentation (graphical presentation forms). HTTP concerns itself

* Unlike Chinese and Japanese, Arabic has only 28 characters. Eight bits provides 256 unique values, which gives plenty of room for Latin characters, Arabic characters, and other useful symbols.

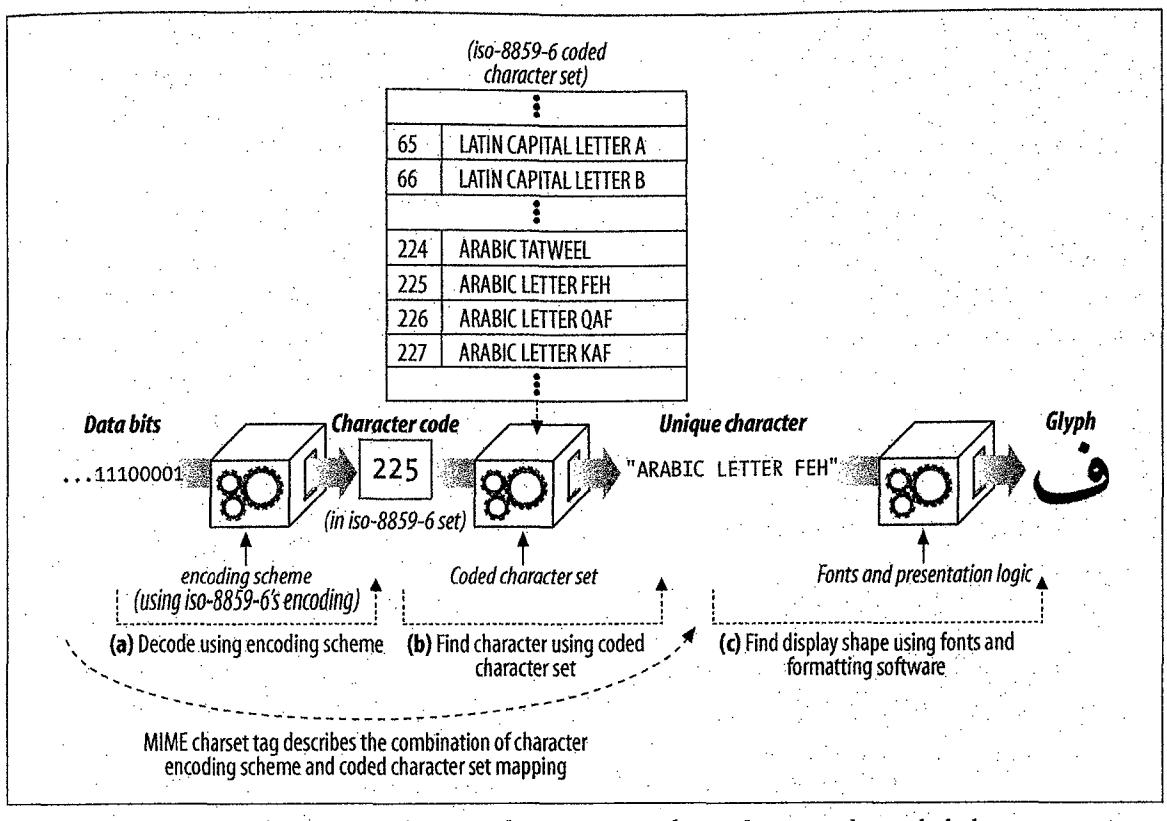


Figure 16-2. HTTP "charset" combines a character encoding scheme and a coded character set

only with transporting the character data and the associated language and charset labels. The presentation of the character shapes is handled by the user's graphics display software (browser, operating system, fonts), as shown in Figure 16-2c.

The Wrong Charset Gives the Wrong Characters

If the client uses the wrong charset parameter, the client will display strange, bogus characters. Let's say a browser got the value 225 (binary 11100001) from the body:

- If the browser thinks the body is encoded with iso-8859-1 Western European character codes, it will show a lowercase Latin "a" with acute accent:

á

- If the browser is using iso-8859-6 Arabic codes, it will show "FEH":

ف

- If the browser is using iso-8859-7 Greek, it will show a small "Alpha":

α

- If the browser is using iso-8859-8 Hebrew codes, it will show "BET":

ב

Standardized MIME Charset Values

The combination of a particular character encoding and a particular coded character set is called a *MIME charset*. HTTP uses standardized MIME charset tags in the Content-Type and Accept-Charset headers. MIME charset values are registered with the IANA.* Table 16-1 lists a few MIME charset encoding schemes used by documents and browsers. A more complete list is provided in Appendix H.

Table 16-1. MIME charset encoding tags

MIME charset value	Description
us-ascii	The famous character encoding standardized in 1968 as ANSI_X3.4-1968. It is also named ASCII, but the "US" prefix is preferred because of several international variants in ISO 646 that modify selected characters. US-ASCII maps 7-bit values into 128 characters. The high bit is unused.
iso-8859-1	iso-8859-1 is an 8-bit extension to ASCII to support Western European languages. It uses the high bit to include many West European characters, while leaving the ASCII codes (0–127) intact. Also called iso-latin-1, or nicknamed "Latin1."
iso-8859-2	Extends ASCII to include characters for Central and Eastern European languages, including Czech, Polish, and Romanian. Also called iso-latin-2.
iso-8859-5	Extends ASCII to include Cyrillic characters, for languages including Russian, Serbian, and Bulgarian.
iso-8859-6	Extends ASCII to include Arabic characters. Because the shapes of Arabic characters change depending on their position in a word, Arabic requires a display engine that analyzes the context and generates the correct shape for each character.
iso-8859-7	Extends ASCII to include modern Greek characters. Formerly known as ELOT-928 or ECMA-118:1986.
iso-8859-8	Extends ASCII to include Hebrew and Yiddish characters.
iso-8859-15	Updates iso-8859-1, replacing some less-needed punctuation and fraction symbols with forgotten French and Finnish letters and replacing the international currency sign with the symbol for the new Euro currency. This character set is nicknamed "Latin0" and may one day replace iso-8859-1 as the preferred default character set in Europe.
iso-2022-jp	iso-2022-jp is a widely used encoding for Japanese email and web content. It is a variable-length encoding scheme that supports ASCII characters with single bytes but uses three-character modal escape sequences to shift into three different Japanese character sets.
euc-jp	euc-jp is an ISO 2022–compliant variable-length encoding that uses explicit bit patterns to identify each character, without requiring modes and escape sequences. It uses 1-byte, 2-byte, and 3-byte sequences of characters to identify characters in multiple Japanese character sets.
Shift_JIS	This encoding was originally developed by Microsoft and sometimes is called SJIS or MS Kanji. It is a bit complicated, for reasons of historic compatibility, and it cannot map all characters, but it still is common.

* See *http://www.iana.org/numbers.htm* for the list of registered charset values.

Table 16-1. MIME charset encoding tags (continued)

MIME charset value	Description
koi8-r	KOI8-R is a popular 8-bit Internet character set encoding for Russian, defined in IETF RFC 1489. The initials are transliterations of the acronym for "Code for Information Exchange, 8 bit, Russian."
utf-8	UTF-8 is a common variable-length character encoding scheme for representing UCS (Unicode), which is the Universal Character Set of the world's characters. UTF-8 uses a variable-length encoding for character code values, representing each character by from one to six bytes. One of the primary features of UTF-8 is backward compatibility with ordinary 7-bit ASCII text.
windows-1252	Microsoft calls its coded character sets "code pages." Windows code page 1252 (a.k.a. "CP1252" or "WinLatin1") is an extension of iso-8859-1.

Content-Type Charset Header and META Tags

Web servers send the client the MIME charset tag in the Content-Type header, using the charset parameter:

```
Content-Type: text/html; charset=iso-2022-jp
```

If no charset is explicitly listed, the receiver may try to infer the character set from the document contents. For HTML content, character sets might be found in <META HTTP-EQUIV="Content-Type"> tags that describe the charset.

Example 16-1 shows how HTML META tags set the charset to the Japanese encoding iso-2022-jp. If the document is not HTML, or there is no META Content-Type tag, software may attempt to infer the character encoding by scanning the actual text for common patterns indicative of languages and encodings.

Example 16-1. Character encoding can be specified in HTML META tags

```
<HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-2022-jp">
    <META LANG="jp">
    <TITLE>A Japanese Document</TITLE>
</HEAD>
<BODY>
  ...
```

If a client cannot infer a character encoding, it assumes iso-8859-1.

The Accept-Charset Header

There are thousands of defined character encoding and decoding methods, developed over the past several decades. Most clients do not support all the various character coding and mapping systems.

HTTP clients can tell servers precisely which character systems they support, using the Accept-Charset request header. The Accept-Charset header value provides a list of character encoding schemes that the client supports. For example, the following HTTP request header indicates that a client accepts the Western European iso-8859-1

character system as well as the UTF-8 variable-length Unicode compatibility system. A server is free to return content in either of these character encoding schemes.

```
Accept-Charset: iso-8859-1, utf-8
```

Note that there is no Content-Charset response header to match the Accept-Charset request header. The response character set is carried back from the server by the charset parameter of the Content-Type response header, to be compatible with MIME. It's too bad this isn't symmetric, but all the information still is there.

Multilingual Character Encoding Primer

The previous section described how the HTTP Accept-Charset header and the Content-Type charset parameter carry character-encoding information from the client and server. HTTP programmers who do a lot of work with international applications and content need to have a deeper understanding of multilingual character systems to understand technical specifications and properly implement software.

It isn't easy to learn multilingual character systems—the terminology is complex and inconsistent, you often have to pay to read the standards documents, and you may be unfamiliar with the other languages with which you're working. This section is an overview of character systems and standards. If you are already comfortable with character encodings, or are not interested in this detail, feel free to jump ahead to "Language Tags and HTTP."

Character Set Terminology

Here are eight terms about electronic character systems that you should know:

Character
: An alphabetic letter, numeral, punctuation mark, ideogram (as in Chinese), symbol, or other textual "atom" of writing. The Universal Character Set (UCS) initiative, known informally as Unicode,* has developed a standardized set of textual names for many characters in many languages, which often are used to conveniently and uniquely name characters.†

Glyph
: A stroke pattern or unique graphical shape that describes a character. A character may have multiple glyphs if it can be written different ways (see Figure 16-3).

Coded character
: A unique number assigned to a character so that we can work with it.

Coding space
: A range of integers that we plan to use as character code values.

* Unicode is a commercial consortium based on UCS that drives commercial products.

† The names look like "LATIN CAPITAL LETTER S" and "ARABIC LETTER QAF."

Code width
: The number of bits in each (fixed-size) character code.

Character repertoire
: A particular working set of characters (a subset of all the characters in the world).

Coded character set
: A set of coded characters that takes a character repertoire (a selection of characters from around the world) and assigns each character a code from a coding space. In other words, it maps numeric character codes to real characters.

Character encoding scheme
: An algorithm to encode numeric character codes into a sequence of content bits (and to decode them back). Character encoding schemes can be used to reduce the amount of data required to identify characters (compression), work around transmission restrictions, and unify overlapping coded character sets.

Charset Is Poorly Named

Technically, the MIME charset tag (used in the Content-Type charset parameter and the Accept-Charset header) doesn't specify a character set at all. The MIME charset value names a total algorithm for mapping data bits to codes to unique characters. It combines the two separate concepts of *character encoding scheme* and *coded character set* (see Figure 16-2).

This terminology is sloppy and confusing, because there already are published standards for character encoding schemes and for coded character sets.* Here's what the HTTP/1.1 authors say about their use of terminology (in RFC 2616):

> The term "character set" is used in this document to refer to a method ... to convert a sequence of octets into a sequence of characters... Note: This use of the term "character set" is more commonly referred to as a "character encoding." However, since HTTP and MIME share the same registry, it's important that the terminology also be shared.

The IETF also adopts nonstandard terminology in RFC 2277:

> This document uses the term "charset" to mean a set of rules for mapping from a sequence of octets to a sequence of characters, such as the combination of a coded character set and a character encoding scheme; this is also what is used as an identifier in MIME "charset=" parameters, and registered in the IANA charset registry. (Note that this is NOT a term used by other standards bodies, such as ISO).

So, be careful when reading standards documents, so you know exactly what's being defined. Now that we've got the terminology sorted out, let's look a bit more closely at characters, glyphs, character sets, and character encodings.

* Worse, the MIME charset tag often co-opts the name of a particular coded character set or encoding scheme. For example, iso-8859-1 is a coded character set (it assigns numeric codes to a set of 256 European characters), but MIME uses the charset value "iso-8859-1" to mean an 8-bit identity encoding of the coded character set. This imprecise terminology isn't fatal, but when reading standards documents, be clear on the assumptions.

Characters

Characters are the most basic building blocks of writing. A character represents an alphabetic letter, numeral, punctuation mark, ideogram (as in Chinese), mathematical symbol, or other basic unit of writing.

Characters are independent of font and style. Figure 16-3 shows several variants of the same character, called "LATIN SMALL LETTER A." A native reader of Western European languages would immediately recognize all five of these shapes as the same character, even though the stroke patterns and styles are quite different.

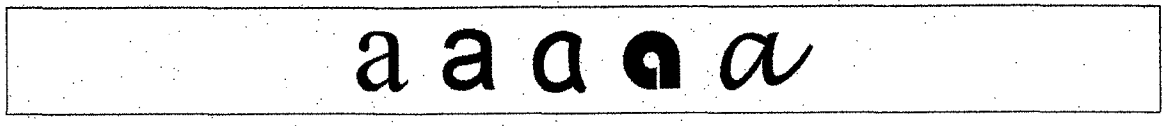


Figure 16-3. One character can have many different written forms

Many writing systems also have different stroke shapes for a single character, depending on the position of the character in the word. For example, the four strokes in Figure 16-4 all represent the character "ARABIC LETTER AIN."* Figure 16-4a shows how "AIN" is written as a standalone character. Figure 16-4d shows "AIN" at the beginning of a word, Figure 16-4c shows "AIN" in the middle of a word, and Figure 16-4b shows "AIN" at the end of a word.†

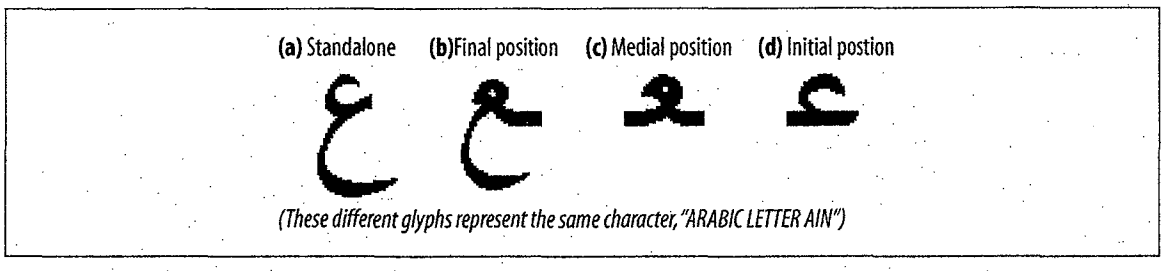


Figure 16-4. Four positional forms of the single character "ARABIC LETTER AIN"

Glyphs, Ligatures, and Presentation Forms

Don't confuse characters with glyphs. Characters are the unique, abstract "atoms" of language. Glyphs are the particular ways you draw each character. Each character has many different glyphs, depending on the artistic style and script.‡

Also, don't confuse characters with presentation forms. To make writing look nicer, many handwritten scripts and typefaces let you join adjacent characters into pretty *ligatures*, in which the two characters smoothly connect. English-speaking

* The sound "AIN" is pronounced something like "ayine," but toward the back of the throat.

† Note that Arabic words are written from right to left.

‡ Many people use the term "glyph" to mean the final rendered bitmap image, but technically a glyph is the inherent shape of a character, independent of font and minor artistic style. This distinction isn't very easy to apply, or useful for our purposes.

typesetters often join "F" and "I" into an "FI ligature" (see Figure 16-5a–b), and Arabic writers often join the "LAM" and "ALIF" characters into an attractive ligature (Figure 16-5c–d).

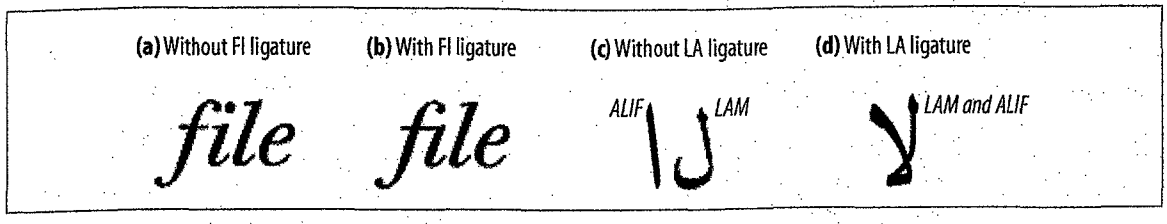


Figure 16-5. Ligatures are stylistic presentation forms of adjacent characters, not new characters

Here's the general rule: if the meaning of the text changes when you replace one glyph with another, the glyphs are different characters. Otherwise, they are the same characters, with a different stylistic presentation.*

Coded Character Sets

Coded character sets, defined in RFCs 2277 and 2130, map integers to characters. Coded character sets often are implemented as arrays,† indexed by code number (see Figure 16-6). The array elements are characters.‡

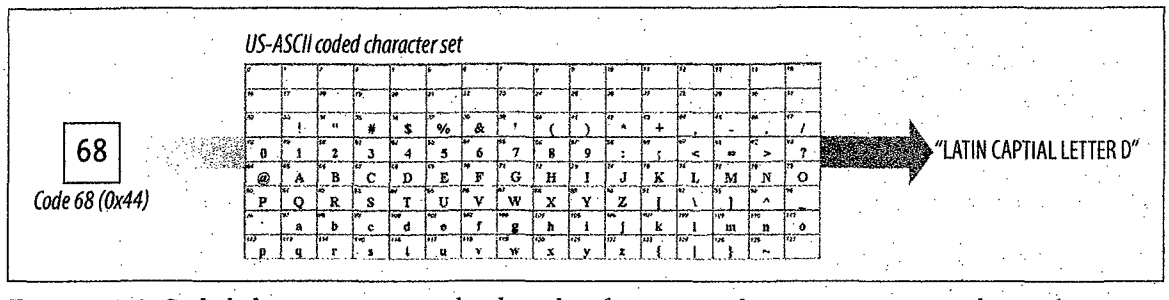


Figure 16-6. Coded character sets can be thought of as arrays that map numeric codes to characters

Let's look at a few important coded character set standards, including the historic US-ASCII character set, the iso-8859 extensions to ASCII, the Japanese JIS X 0201 character set, and the Universal Character Set (Unicode).

US-ASCII: The mother of all character sets

ASCII is the most famous coded character set, standardized back in 1968 as ANSI standard X3.4 "American Standard Code for Information Interchange." ASCII uses

* The division between semantics and presentation isn't always clear. For ease of implementation, some presentation variants of the same characters have been assigned distinct characters, but the goal is to avoid this.

† The arrays can be multidimensional, so different bits of the code number index different axes of the array.

‡ Figure 16-6 uses a grid to represent a coded character set. Each element of the grid contains a character image. These images are symbolic. The presence of an image "D" is shorthand for the character "LATIN CAPITAL LETTER D," not for any particular graphical glyph.

only the code values 0–127, so only 7 bits are required to cover the code space. The preferred name for ASCII is "US-ASCII," to distinguish it from international variants of the 7-bit character set.

HTTP messages (headers, URIs, etc.) use US-ASCII.

iso-8859

The iso-8859 character set standards are 8-bit supersets of US-ASCII that use the high bit to add characters for international writing. The additional space provided by the extra bit (128 extra codes) isn't large enough to hold even all of the European characters (not to mention Asian characters), so iso-8859 provides customized character sets for different regions:

iso-8859-1	Western European languages (e.g., English, French)
iso-8859-2	Central and Eastern European languages (e.g., Czech, Polish)
iso-8859-3	Southern European languages
iso-8859-4	Northern European languages (e.g., Latvian, Lithuanian, Greenlandic)
iso-8859-5	Cyrillic (e.g., Bulgarian, Russian, Serbian)
iso-8859-6	Arabic
iso-8859-7	Greek
iso-8859-8	Hebrew
iso-8859-9	Turkish
iso-8859-10	Nordic languages (e.g., Icelandic, Inuit)
iso-8859-15	Modification to iso-8859-1 that includes the new Euro currency character

iso-8859-1, also known as Latin1, is the default character set for HTML. It can be used to represent text in most Western European languages. There has been some discussion of replacing iso-8859-1 with iso-8859-15 as the default HTTP coded character set, because it includes the new Euro currency symbol. However, because of the widespread adoption of iso-8859-1, it's unlikely that a widespread change to iso-8859-15 will be adopted for quite some time.

JIS X 0201

JIS X 0201 is an extremely minimal character set that extends ASCII with Japanese half width katakana characters. The half-width katakana characters were originally used in the Japanese telegraph system. JIS X 0201 is often called "JIS Roman." JIS is an acronym for "Japanese Industrial Standard."

JIS X 0208 and JIS X 0212

Japanese includes thousands of characters from several writing systems. While it is possible to limp by (painfully) using the 63 basic phonetic katakana characters in JIS X 0201, a much more complete character set is required for practical use.

The JIS X 0208 character set was the first multi-byte Japanese character set; it defined 6,879 coded characters, most of which are Chinese-based kanji. The JIS X 0212 character set adds an additional 6,067 characters.

UCS

The Universal Character Set (UCS) is a worldwide standards effort to combine all of the world's characters into a single coded character set. UCS is defined by ISO 10646. Unicode is a commercial consortium that tracks the UCS standards. UCS has coding space for millions of characters, although the basic set consists of only about 50,000 characters.

Character Encoding Schemes

Character encoding schemes pack character code numbers into content bits and unpack them back into character codes at the other end (Figure 16-7). There are three broad classes of character encoding schemes:

Fixed width
: Fixed-width encodings represent each coded character with a fixed number of bits. They are fast to process but can waste space.

Variable width (nonmodal)
: Variable-width encodings use different numbers of bits for different character code numbers. They can reduce the number of bits required for common characters, and they retain compatibility with legacy 8-bit character sets while allowing the use of multiple bytes for international characters.

Variable width (modal)
: Modal encodings use special "escape" patterns to shift between different modes. For example, a modal encoding can be used to switch between multiple, overlapping character sets in the middle of text. Modal encodings are complicated to process, but they can efficiently support complicated writing systems.

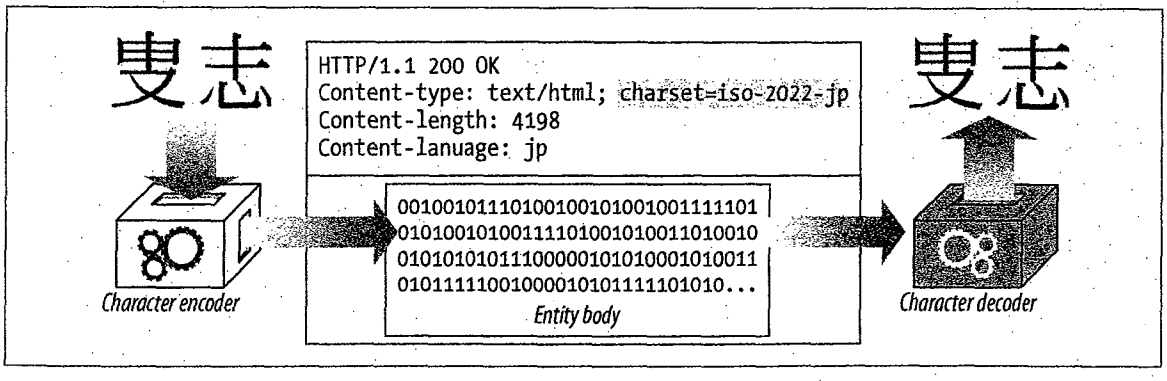


Figure 16-7. Character encoding scheme encodes character codes into bits and back again

Let's look at a few common encoding schemes.

8-bit

The 8-bit fixed-width identity encoding simply encodes each character code with its corresponding 8-bit value. It supports only character sets with a code range of 256 characters. The iso-8859 family of character sets uses the 8-bit identity encoding.

UTF-8

UTF-8 is a popular character encoding scheme designed for UCS (UTF stands for "UCS Transformation Format"). UTF-8 uses a nonmodal, variable-length encoding for the character code values, where the leading bits of the first byte tell the length of the encoded character in bytes, and any subsequent byte contains six bits of code value (see Table 16-2).

If the first encoded byte has a high bit of 0, the length is just 1 byte, and the remaining 7 bits contain the character code. This has the nice result of ASCII compatibility (but not iso-8859 compatibility, because iso-8859 uses the high bit).

Table 16-2. UTF-8 variable-width, nonmodal encoding

Character code bits	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
0–7	0ccccccc	-	-	-	-	-
8–11	110ccccc	10cccccc	-	-	-	-
12–16	1110cccc	10cccccc	10cccccc	-	-	-
17–21	11110ccc	10cccccc	10cccccc	10cccccc	-	-
22–26	111110cc	10cccccc	10cccccc	10cccccc	10cccccc	-
27–31	1111110c	10cccccc	10cccccc	10cccccc	10cccccc	10cccccc

For example, character code 90 (ASCII "Z") would be encoded as 1 byte (01011010), while code 5073 (13-bit binary value 1001111010001) would be encoded into 3 bytes:

```
11100001 10001111 10010001
```

iso-2022-jp

iso-2022-jp is a widely used encoding for Japanese Internet documents. iso-2022-jp is a variable-length, modal encoding, with all values less than 128 to prevent problems with non–8-bit-clean software.

The encoding context always is set to one of four predefined character sets.* Special "escape sequences" shift from one set to another. iso-2022-jp initially uses the US-ASCII character set, but it can switch to the JIS X 0201 (JIS-Roman) character set or the much larger JIS X 0208-1978 and JIS X 0208-1983 character sets using 3-byte escape sequences.

* The iso-2022-jp encoding is tightly bound to these four character sets, whereas some other encodings are independent of the particular character set.

The escape sequences are shown in Table 16-3. In practice, Japanese text begins with "ESC $ @" or "ESC $ B" and ends with "ESC (B" or "ESC (J".

Table 16-3. iso-2022-jp character set switching escape sequences

Escape sequence	Resulting coded character set	Bytes per code
ESC (B	US-ASCII	1
ESC (J	JIS X 0201-1976 (JIS Roman)	1
ESC $ @	JIS X 0208-1978	2
ESC $ B	JIS X 0208-1983	2

When in the US-ASCII or JIS-Roman modes, a single byte is used per character. When using the larger JIS X 0208 character set, two bytes are used per character code. The encoding restricts the bytes sent to be between 33 and 126.*

euc-jp

euc-jp is another popular Japanese encoding. EUC stands for "Extended Unix Code," first developed to support Asian characters on Unix operating systems.

Like iso-2022-jp, the euc-jp encoding is a variable-length encoding that allows the use of several standard Japanese character sets. But unlike iso-2022-jp, the euc-jp encoding is not modal. There are no escape sequences to shift between modes.

euc-jp supports four coded character sets: JIS X 0201 (JIS-Roman, ASCII with a few Japanese substitutions), JIS X 0208, half-width katakana (63 characters used in the original Japanese telegraph system), and JIS X 0212.

One byte is used to encode JIS Roman (ASCII compatible), two bytes are used for JIS X 0208 and half-width katakana, and three bytes are used for JIS X 0212. The coding is a bit wasteful but is simple to process.

The encoding patterns are outlined in Table 16-4.

Table 16-4. euc-jp encoding values

Which byte	Encoding values
JIS X 0201 (94 coded characters)	
1st byte	33–126
JIS X 0208 (6879 coded characters)	
1st byte	161–254
2nd byte	161–254

* Though the bytes can have only 94 values (between 33 and 126), this is sufficient to cover all the characters in the JIS X 0208 character sets, because the character sets are organized into a 94×94 grid of code values, enough to cover all JIS X 0208 character codes.

Table 16-4. euc-jp encoding values (continued)

Which byte	Encoding values
Half-width katakana (63 coded characters)	
1st byte	142
2nd byte	161–223
JIS X 0212 (6067 coded characters)	
1st byte	143
2nd byte	161–254
3rd byte	161–254

This wraps up our survey of character sets and encodings. The next section explains language tags and how HTTP uses language tags to target content to audiences. Please refer to Appendix H for a detailed listing of standardized character sets.

Language Tags and HTTP

Language tags are short, standardized strings that name spoken languages.

We need standardized names, or some people will tag French documents as "French," others will use "Français," others still might use "France," and lazy people might just use "Fra" or "F." Standardized language tags avoid this confusion.

There are language tags for English (en), German (de), Korean (ko), and many other languages. Language tags can describe regional variants and dialects of languages, such as Brazilian Portuguese (pt-BR), U.S. English (en-US), and Hunan Chinese (zh-xiang). There is even a standard language tag for Klingon (i-klingon)!

The Content-Language Header

The Content-Language entity header field describes the target audience languages for the entity. If the content is intended primarily for a French audience, the Content-Language header field would contain:

```
Content-Language: fr
```

The Content-Language header isn't limited to text documents. Audio clips, movies, and applications might all be intended for a particular language audience. Any media type that is targeted to particular language audiences can have a Content-Language header. In Figure 16-8, the audio file is tagged for a Navajo audience.

If the content is intended for multiple audiences, you can list multiple languages. As suggested in the HTTP specification, a rendition of the "Treaty of Waitangi," presented simultaneously in the original Maori and English versions, would call for:

```
Content-Language: mi, en
```

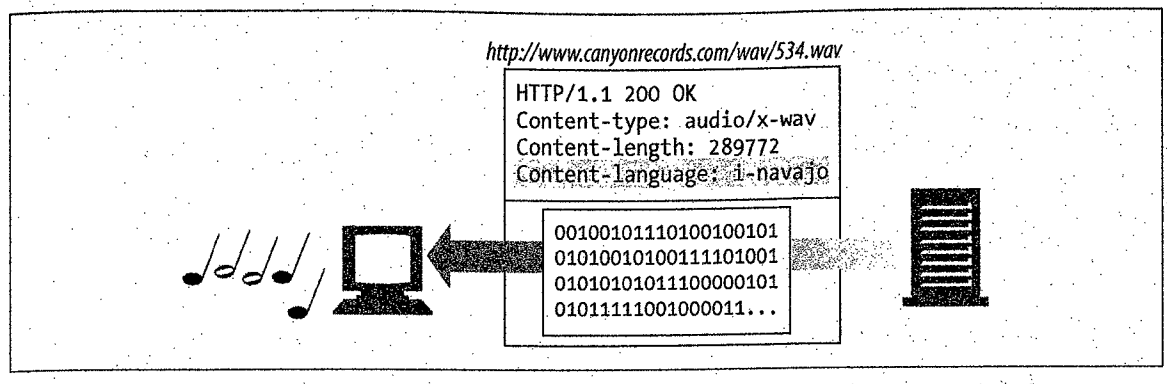


Figure 16-8. Content-Language header marks a "Rain Song" audio clip for Navajo speakers

However, just because multiple languages are present within an entity does not mean that it is intended for multiple linguistic audiences. A beginner's language primer, such as "A First Lesson in Latin," which clearly is intended to be used by an English-literate audience, would properly include only "en".

The Accept-Language Header

Most of us know at least one language. HTTP lets us pass our language restrictions and preferences along to web servers. If the web server has multiple versions of a resource, in different languages, it can give us content in our preferred language.*

Here, a client requests Spanish content:

```
Accept-Language: es
```

You can place multiple language tags in the Accept-Language header to enumerate all supported languages and the order of preference (left to right). Here, the client prefers English but will accept Swiss German (de-CH) or other variants of German (de):

```
Accept-Language: en, de-CH, de
```

Clients use Accept-Language and Accept-Charset to request content they can understand. We'll see how this works in more detail in Chapter 17.

Types of Language Tags

Language tags have a standardized syntax, documented in RFC 3066, "Tags for the Identification of Languages." Language tags can be used to represent:

- General language classes (as in "es" for Spanish)
- Country-specific languages (as in "en-GB" for English in Great Britain)
- Dialects of languages (as in "no-bok" for Norwegian "Book Language")

* Servers also can use the Accept-Language header to generate dynamic content in the language of the user or to select images or target language-appropriate merchandising promotions.

- Regional languages (as in "sgn-US-MA" for Martha's Vineyard sign language)
- Standardized nonvariant languages (e.g., "i-navajo")
- Nonstandard languages (e.g., "x-snowboarder-slang"*)

Subtags

Language tags have one or more parts, separated by hyphens, called *subtags*:

- The first subtag called the *primary subtag*. The values are standardized.
- The second subtag is optional and follows its own naming standard.
- Any trailing subtags are unregistered.

The primary subtag contains only letters (A–Z). Subsequent subtags can contain letters or numbers, up to eight characters in length. An example is shown in Figure 16-9.

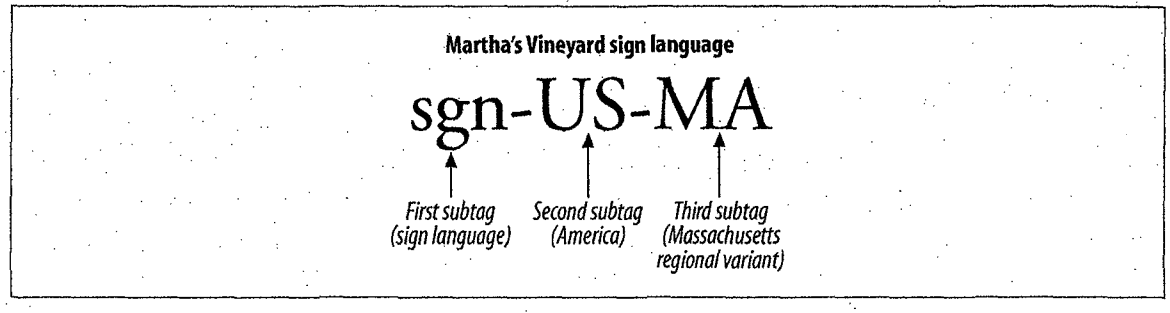


Figure 16-9. Language tags are separated into subtags

Capitalization

All tags are case-insensitive—the tags "en" and "eN" are equivalent. However, lowercasing conventionally is used to represent general languages, while uppercasing is used to signify particular countries. For example, "fr" means all languages classified as French, while "FR" signifies the country France.†

IANA Language Tag Registrations

The values of the first and second language subtags are defined by various standards documents and their maintaining organizations. The IANA‡ administers the list of standard language tags, using the rules outlined in RFC 3066.

If a language tag is composed of standard country and language values, the tag doesn't have to be specially registered. Only those language tags that can't be composed out of the standard country and language values need to be registered specially with the

* Describes the unique dialect spoken by "shredders."

† This convention is recommended by ISO standard 3166.

‡ See *http://www.iana.org* and RFC 2860.

IANA.* The following sections outline the RFC 3066 standards for the first and second subtags.

First Subtag: Namespace

The first subtag usually is a standardized *language* token, chosen from the ISO 639 set of language standards. But it also can be the letter "i" to identify IANA-registered names, or "x" for private, extension names. Here are the rules:

If the first subtag has:

- Two characters, it is a language code from the ISO 639† and 639-1 standards
- Three characters, it is a language code listed in the ISO 639-2‡ standard and extensions
- The letter "i," the language tag is explicitly IANA-registered
- The letter "x," the language tag is a private, nonstandard, extension subtag

The ISO 639 and 639-2 names are summarized in Appendix G. A few examples are shown here in Table 16-5.

Table 16-5. Sample ISO 639 and 639-2 language codes

Language	ISO 639	ISO 639-2
Arabic	ar	ara
Chinese	zh	chi/zho
Dutch	nl	dut/nla
English	en	eng
French	fr	fra/fre
German	de	deu/ger
Greek (Modern)	el	ell/gre
Hebrew	he	heb
Italian	it	ita
Japanese	ja	jpn
Korean	ko	kor
Norwegian	no	nor
Russian	ru	rus
Spanish	es	esl/spa

* At the time of writing, only 21 language tags have been explicitly registered with the IANA, including Cantonese ("zh-yue"), New Norwegian ("no-nyn"), Luxembourgish ("i-lux"), and Klingon ("i-klingon"). The hundreds of remaining spoken languages in use on the Internet have been composed from standard components.

† See ISO standard 639, "Codes for the representation of names of languages."

‡ See ISO 639-2, "Codes for the representation of names of languages—Part 2: Alpha-3 code."

Table 16-5. Sample ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Swedish	sv	sve/swe
Turkish	tr	tur

Second Subtag: Namespace

The second subtag usually is a standardized *country* token, chosen from the ISO 3166 set of country code and region standards. But it may also be another string, which you may register with the IANA. Here are the rules:

If the second subtag has:

- Two characters, it's a country/region defined by ISO 3166*
- Three to eight characters, it may be registered with the IANA
- One character, it is illegal

Some of the ISO 3166 country codes are shown in Table 16-6. The complete list of country codes can be found in Appendix G.

Table 16-6. Sample ISO 3166 country codes

Country	Code
Brazil	BR
Canada	CA
China	CN
France	FR
Germany	DE
Holy See (Vatican City State)	VA
Hong Kong	HK
India	IN
Italy	IT
Japan	JP
Lebanon	LB
Mexico	MX
Pakistan	PK
Russian Federation	RU
United Kingdom	GB
United States	US

* The country codes AA, QM–QZ, XA–XZ and ZZ are reserved by ISO 3166 as user-assigned codes. These must not be used to form language tags.

Remaining Subtags: Namespace

There are no rules for the third and following subtags, apart from being up to eight characters (letters and digits).

Configuring Language Preferences

You can configure language preferences in your browser profile.

Netscape Navigator lets you set language preferences through Edit → Preferences... → Languages..., and Microsoft Internet Explorer lets you set languages through Tools → Internet Options... → Languages.

Language Tag Reference Tables

Appendix G contains convenient reference tables for language tags:

- IANA-registered language tags are shown in Table G-1.
- ISO 639 language codes are shown in Table G-2.
- ISO 3166 country codes are shown in Table G-3.

Internationalized URIs

Today, URIs don't provide much support for internationalization. With a few (poorly defined) exceptions, today's URIs are comprised of a subset of US-ASCII characters. There are efforts underway that might let us include a richer set of characters in the hostnames and paths of URLs, but right now, these standards have not been widely accepted or deployed. Let's review today's practice.

Global Transcribability Versus Meaningful Characters

The URI designers wanted everyone around the world to be able to share URIs with each other—by email, by phone, by billboard, even over the radio. And they wanted URIs to be easy to use and remember. These two goals are in conflict.

To make it easy for folks around the globe to enter, manipulate, and share URIs, the designers chose a very limited set of common characters for URIs (basic Latin alphabet letters, digits, and a few special characters). This small repertoire of characters is supported by most software and keyboards around the world.

Unfortunately, by restricting the character set, the URI designers made it much harder for people around the globe to create URIs that are easy to use and remember. The majority of world citizens don't even recognize the Latin alphabet, making it nearly impossible to remember URIs as abstract patterns.

The URI authors felt it was more important to ensure transcribability and sharability of resource identifiers than to have them consist of the most meaningful characters. So we have URIs that (today) essentially consist of a restricted subset of ASCII characters.

URI Character Repertoire

The subset of US-ASCII characters permitted in URIs can be divided into *reserved*, *unreserved*, and *escape* character classes. The unreserved character classes can be used generally within any component of URIs that allow them. The reserved characters have special meanings in many URIs, so they shouldn't be used in general. See Table 16-7 for a list of the unreserved, reserved, and escape characters.

Table 16-7. URI character syntax

Character class	Character repertoire
Unreserved	[A-Za-z0-9] | "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
Reserved	";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
Escape	"%" <HEX> <HEX>

Escaping and Unescaping

URI "escapes" provide a way to safely insert reserved characters and other unsupported characters (such as spaces) inside URIs. An escape is a three-character sequence, consisting of a percent character (%) followed by two hexadecimal digit characters. The two hex digits represent the code for a US-ASCII character.

For example, to insert a space (ASCII 32) in a URL, you could use the escape "%20", because 20 is the hexadecimal representation of 32. Similarly, if you wanted to include a percent sign and have it not be treated as an escape, you could enter "%25", where 25 is the hexadecimal value of the ASCII code for percent.

Figure 16-10 shows how the conceptual characters for a URI are turned into code bytes for the characters, in the current character set. When the URI is needed for processing, the escapes are undone, yielding the underlying ASCII code bytes.

Internally, HTTP applications should transport and forward URIs with the escapes in place. HTTP applications should unescape the URIs only when the data is needed. And, more importantly, the applications should ensure that no URI ever is unescaped twice, because percent signs that might have been encoded in an escape will themselves be unescaped, leading to loss of data.

Escaping International Characters

Note that escape values should be in the range of US-ASCII codes (0–127). Some applications attempt to use escape values to represent iso-8859-1 extended characters (128–255)—for example, web servers might erroneously use escapes to code

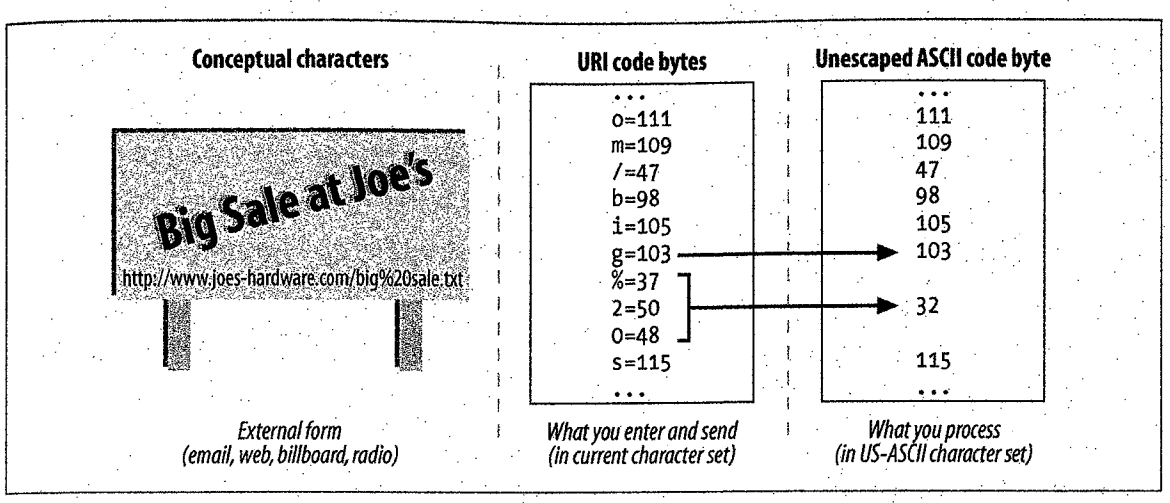


Figure 16-10. URI characters are transported as escaped code bytes but processed unescaped

filenames that contain international characters. This is incorrect and may cause problems with some applications.

For example, the filename *Sven Ölssen.html* (containing an umlaut) might be encoded by a web server as *Sven%20%D6lssen.html*. It's fine to encode the space with %20, but is technically illegal to encode the Ö with %D6, because the code D6 (decimal 214) falls outside the range of ASCII. ASCII defines only codes up to 0x7F (decimal 127).

Modal Switches in URIs

Some URIs also use sequences of ASCII characters to represent characters in other character sets. For example, iso-2022-jp encoding might be used to insert "ESC (J" to shift into JIS-Roman and "ESC (B" to shift back to ASCII. This works in some local circumstances, but the behavior is not well defined, and there is no standardized scheme to identify the particular encoding used for the URL. As the authors of RFC 2396 say:

> For original character sequences that contain non-ASCII characters, however, the situation is more difficult. Internet protocols that transmit octet sequences intended to represent character sequences are expected to provide some way of identifying the charset used, if there might be more than one [RFC2277].
>
> However, there is currently no provision within the generic URI syntax to accomplish this identification. An individual URI scheme may require a single charset, define a default charset, or provide a way to indicate the charset used. It is expected that a systematic treatment of character encoding within URI will be developed as a future modification of this specification.

Currently, URIs are not very international-friendly. The goal of URI portability outweighed the goal of language flexibility. There are efforts currently underway to internationalize URIs, but in the near term, HTTP applications should stick with ASCII. It's been around since 1968, so it can't be all that bad.

Other Considerations

This section discusses a few other things you should keep in mind when writing international HTTP applications.

Headers and Out-of-Spec Data

HTTP headers must consist of characters from the US-ASCII character set. However, not all clients and servers implement this correctly, so you may on occasion receive illegal characters with code values larger than 127.

Many HTTP applications use operating-system and library routines for processing characters (for example, the Unix ctype character classification library). Not all of these libraries support character codes outside of the ASCII range (0–127).

In some circumstances (generally, with older implementations), these libraries may return improper results or crash the application when given non-ASCII characters. Carefully read the documentation for your character classification libraries before using them to process HTTP messages, in case the messages contain illegal data.

Dates

The HTTP specification clearly defines the legal GMT date formats, but be aware that not all web servers and clients follow the rules. For example, we have seen web servers send invalid HTTP Date headers with months expressed in local languages.

HTTP applications should attempt to be tolerant of out-of-spec dates, and not crash on receipt, but they may not always be able to interpret all dates sent. If the date is not parseable, servers should treat it conservatively.

Domain Names

DNS doesn't currently support international characters in domain names. There are standards efforts under way to support multilingual domain names, but they have not yet been widely deployed.

For More Information

The very success of the World Wide Web means that HTTP applications will continue to exchange more and more content in different languages and character sets. For more information on the important but slightly complex topic of multilingual multimedia, please refer to the following sources.

Appendixes

- IANA-registered charset tags are listed in Table H-1.
- IANA-registered language tags are shown in Table G-1.
- ISO 639 language codes are shown in Table G-2.
- ISO 3166 country codes are shown in Table G-3.

Internet Internationalization

http://www.w3.org/International/
: "Making the WWW Truly World Wide"—the W3C Internationalization and Localization web site.

http://www.ietf.org/rfc/rfc2396.txt
: RFC 2396, "Uniform Resource Identifiers (URI): Generic Syntax," is the defining document of URIs. This document includes sections describing character set restrictions for international URIs.

CJKV Information Processing
: Ken Lunde, O'Reilly & Associates, Inc. CJKV is the bible of Asian electronic character processing. Asian character sets are varied and complex, but this book provides an excellent introduction to the standards technologies for large character sets.

http://www.ietf.org/rfc/rfc2277.txt
: RFC 2277, "IETF Policy on Character Sets and Languages," documents the current policies being applied by the Internet Engineering Steering Group (IESG) toward the standardization efforts in the Internet Engineering Task Force (IETF) in order to help Internet protocols interchange data in multiple languages and characters.

International Standards

http://www.iana.org/numbers.htm
: The Internet Assigned Numbers Authority (IANA) contains repositories of registered names and numbers. The "Protocol Numbers and Assignments Directory" contains records of registered character sets for use on the Internet. Because much work on international communications falls under the domain of the ISO, and not the Internet community, the IANA listings are not exhaustive.

http://www.ietf.org/rfc/rfc3066.txt
: RFC 3066, "Tags for the Identification of Languages," describes language tags, their values, and how to construct them.

"Codes for the representation of names of languages"
ISO 639:1988 (E/F), The International Organization for Standardization, first edition.

"Codes for the representation of names of languages—Part 2: Alpha-3 code"
ISO 639-2:1998, Joint Working Group of ISO TC46/SC4 and ISO TC37/SC2, first edition.

"Codes for the representation of names of countries"
ISO 3166:1988 (E/F), The International Organization for Standardization, third edition.

CHAPTER 17

Content Negotiation and Transcoding

Often, a single URL may need to correspond to different resources. Take the case of a web site that wants to offer its content in multiple languages. If a site such as Joe's Hardware has both French- and English-speaking users, it might want to offer its web site in both languages. However, if this is the case, when one of Joe's customers requests "http://www.joes-hardware.com," which version should the server send? French or English?

Ideally, the server will send the English version to an English speaker and the French version to a French speaker—a user could go to Joe's Hardware's home page and get content in the language he speaks. Fortunately, HTTP provides *content-negotiation* methods that allow clients and servers to make just such determinations. Using these methods, a single URL can correspond to different resources (e.g., a French and English version of the same web page). These different versions are called *variants*.

Servers also can make other types of decisions about what content is best to send to a client for a particular URL. In some cases, servers even can automatically generate customized pages—for instance, a server can convert an HTML page into a WML page for your handheld device. These kinds of dynamic content transformations are called *transcodings*. They are done in response to content negotiation between HTTP clients and servers.

In this chapter, we will discuss content negotiation and how web applications go about their content-negotiation duties.

Content-Negotiation Techniques

There are three distinct methods for deciding which page at a server is the right one for a client: present the choice to the client, decide automatically at the server, or ask an intermediary to select. These three techniques are called client-driven negotiation, server-driven negotiation, and transparent negotiation, respectively (see Table 17-1).

In this chapter, we will look at the mechanics of each technique as well as their advantages and disadvantages.

Table 17-1. Summary of content-negotiation techniques

Technique	How it works	Advantages	Drawbacks
Client-driven	Client makes a request, server sends list of choices to client, client chooses.	Easiest to implement at server side. Client can make best choice.	Adds latency: at least two requests are needed to get the correct content.
Server-driven	Server examines client's request headers and decides what version to serve.	Quicker than client-driven negotiation. HTTP provides a q-value mechanism to allow servers to make approximate matches and a Vary header for servers to tell downstream devices how to evaluate requests.	If the decision is not obvious (headers don't match up), the server must guess.
Transparent	An intermediate device (usually a proxy cache) does the request negotiation on the client's behalf.	Offloads the negotiation from the web server. Quicker than client-driven negotiation.	No formal specifications for how to do transparent negotiation.

Client-Driven Negotiation

The easiest thing for a server to do when it receives a client request is to send back a response listing the available pages and let the client decide which one it wants to see. This, of course, is the easiest to implement at the server and is likely to result in the best copy being selected (provided that the list has enough information to allow the client to pick the right copy). The disadvantage is that two requests are needed for each page—one to get the list and a second to get the selected copy. This is a slow and tedious process, and it's likely to become annoying to the client.

Mechanically, there are actually two ways for servers to present the choices to the client for selection: by sending back an HTML document with links to the different versions of the page and descriptions of each of the versions, or by sending back an HTTP/1.1 response with the 300 Multiple Choices response code. The client browser may receive this response and display a page with the links, as in the first method, or it may pop up a dialog window asking the user to make a selection. In any case, the decision is made manually at the client side by the browser user.

In addition to the increased latency and annoyance of multiple requests per page, this method has another drawback: it requires multiple URLs—one for the main page and one for each specific page. So, if the original request was for *www.joes-hardware.com*, Joe's server may respond with a page that has links to *www.joes-hardware.com/english* and *www.joes-hardware.com/french*. Should clients now bookmark the original main page or the selected ones? Should they tell their friends about the great web site at *www.joes-hardware.com* or tell only their English-speaking friends about the web site at *www.joes-hardware.com/english*?

Server-Driven Negotiation

Client-driven negotiation has several drawbacks, as discussed in the previous section. Most of these drawbacks center around the increased communication between the client and server to decide on the best page in response to a request. One way to reduce this extra communication is to let the server decide which page to send back—but to do this, the client must send enough information about its preferences to allow the server to make an informed decision. The server gets this information from the client's request headers.

There are two mechanisms that HTTP servers use to evaluate the proper response to send to a client:

- Examining the set of content-negotiation headers. The server looks at the client's Accept headers and tries to match them with corresponding response headers.
- Varying on other (non–content-negotiation) headers. For example, the server could send responses based on the client's User-Agent header.

These two mechanisms are explained in more detail in the following sections.

Content-Negotiation Headers

Clients may send their preference information using the set of HTTP headers listed in Table 17-2.

Table 17-2. Accept headers

Header	Description
Accept	Used to tell the server what media types are okay to send
Accept-Language	Used to tell the server what languages are okay to send
Accept-Charset	Used to tell the server what charsets are okay to send
Accept-Encoding	Used to tell the server what encodings are okay to send

Notice how similar these headers are to the entity headers discussed in Chapter 15. However, there is a clear distinction between the purposes of the two types of headers. As mentioned in Chapter 15, entity headers are like shipping labels—they specify attributes of the message body that are necessary during the transfer of messages from the server to the client. Content-negotiation headers, on the other hand, are used by clients and servers to exchange preference information and to help choose between different versions of a document, so that the one most closely matching the client's preferences is served.

Servers match clients' Accept headers with the corresponding entity headers, listed in Table 17-3.

Table 17-3. Accept and matching document headers

Accept header	Entity header
Accept	Content-Type
Accept-Language	Content-Language
Accept-Charset	Content-Type
Accept-Encoding	Content-Encoding

Note that because HTTP is a stateless protocol (meaning that servers do not keep track of client preferences across requests), clients must send their preference information with every request.

If both clients sent Accept-Language header information specifying the language in which they were interested, the server could decide which copy of *www.joes-hardware.com* to send back to each client. Letting the server automatically pick which document to send back reduces the latency associated with the back-and-forth communication required by the client-driven model.

However, say that one of the clients prefers Spanish. Which version of the page should the server send back? English or French? The server has just two choices: either guess, or fall back on the client-driven model and ask the client to choose. However, if the Spaniard happens to understand some English, he might choose the English page—it wouldn't be ideal, but it would do. In this case, the Spaniard needs the ability to pass on more information about his preferences, conveying that he does have minimal knowledge of English and that, in a pinch, English will suffice.

Fortunately, HTTP does provide a mechanism for letting clients like our Spaniard give richer descriptions of their preferences, using *quality values* ("q values" for short).

Content-Negotiation Header Quality Values

The HTTP protocol defines quality values to allow clients to list multiple choices for each category of preference and associate an order of preference with each choice. For example, clients can send an Accept-Language header of the form:

```
Accept-Language: en;q=0.5, fr;q=0.0, nl;q=1.0, tr;q=0.0
```

Where the q values can range from 0.0 to 1.0 (with 0.0 being the lowest preference and 1.0 being the highest). The header above, then, says that the client prefers to receive a Dutch (nl) version of the document, but an English (en) version will do. Under no circumstances does the client want a French (fr) or Turkish (tr) version, though. Note that the order in which the preferences are listed is not important; only the q values associated with them are.

Occasionally, the server may not have any documents that match any of the client's preferences. In this case, the server may change or transcode the document to match the client's preferences. This mechanism is discussed later in this chapter.

Varying on Other Headers

Servers also can attempt to match up responses with other client request headers, such as User-Agent. Servers may know that old versions of a browser do not support JavaScript, for example, and may therefore send back a version of the page that does not contain JavaScript.

In this case, there is no q-value mechanism to look for approximate "best" matches. The server either looks for an exact match or simply serves whatever it has (depending on the implementation of the server).

Because caches must attempt to serve correct "best" versions of cached documents, the HTTP protocol defines a Vary header that the server sends in responses; the Vary header tells caches (and clients, and any downstream proxies) which headers the server is using to determine the best version of the response to send. The Vary header is discussed in more detail later in this chapter.

Content Negotiation on Apache

Here is an overview of how the Apache web server supports content negotiation. It is up to the web site content provider—Joe, for example—to provide different versions of Joe's index page. Joe must put all his index page files in the appropriate directory on the Apache server corresponding to his web site. There are two ways to enable content negotiation:

- In the web site directory, create a *type-map file* for each URI in the web site that has variants. The type-map file lists all the variants and the content-negotiation headers to which they correspond.
- Enable the MultiViews directive, which causes Apache to create type-map files for the directory automatically.

Using type-map files

The Apache server needs to know what type-map files look like. To configure this, set a handler in the server configuration file that specifies the file suffix for type-map files. For example:

```
AddHandler type-map .var
```

This line indicates that files with the extension *.var* are type-map files.

Here is a sample type-map file:

```
URI: joes-hardware.html

URI: joes-hardware.en.html
Content-type: text/html
Content-language: en
```

```
URI: joes-hardware.fr.de.html
Content-type: text/html;charset=iso-8859-2
Content-language: fr, de
```

From this type-map file, the Apache server knows to send *joes-hardware.en.html* to clients requesting English and *joes-hardware.fr.de.html* to clients requesting French. Quality values also are supported; see the Apache server documentation.

Using MultiViews

To use MultiViews, you must enable it for the directory containing the web site, using an Options directive in the appropriate section of the *access.conf* file (*<Directory>*, *<Location>*, or *<Files>*).

If MultiViews is enabled and a browser requests a resource named *joes-hardware*, the server looks for all files with "joes-hardware" in the name and creates a type-map file for them. Based on the names, the server guesses the appropriate content-negotiation headers to which the files correspond. For example, a French-language version of *joes-hardware* should contain *.fr*.

Server-Side Extensions

Another way to implement content negotiation at the server is by server-side extensions, such as Microsoft's Active Server Pages (ASP). See Chapter 8 for an overview of server-side extensions.

Transparent Negotiation

Transparent negotiation seeks to move the load of server-driven negotiation away from the server, while minimizing message exchanges with the client by having an intermediary proxy negotiate on behalf of the client. The proxy is assumed to have knowledge of the client's expectations and be capable of performing the negotiations on its behalf (the proxy has received the client's expectations in the request for content). To support transparent content negotiation, the server must be able to tell proxies what request headers the server examines to determine the best match for the client's request. The HTTP/1.1 specification does not define any mechanisms for transparent negotiation, but it does define the Vary header. Servers send Vary headers in their responses to tell intermediaries what request headers they use for content negotiation.

Caching proxies can store different copies of documents accessed via a single URL. If servers communicate their decision-making processes to caches, the caches can negotiate with clients on behalf of the servers. Caches also are great places to transcode content, because a general-purpose transcoder deployed in a cache can transcode content from any server, not just one. Transcoding of content at a cache is illustrated in Figure 17-3 and discussed in more detail later in the chapter.

Caching and Alternates

Caching of content assumes that the content can be reused later. However, caches must employ much of the decision-making logic that servers do when sending back a response, to ensure that they send back the correct cached response to a client request.

The previous section described the Accept headers sent by clients and the corresponding entity headers that servers match them up against in order to choose the best response to each request. Caches must use these same headers to decide which cached response to send back.

Figure 17-1 illustrates both a correct and incorrect sequence of operations involving a cache. The first request results in the cache forwarding the request to the server and storing the response. The second response is looked up by the cache, and a document matching the URL is found. This document, however, is in French, and the requestor wants a Spanish document. If the cache just sends back the French document to the requestor, it will be behaving incorrectly.

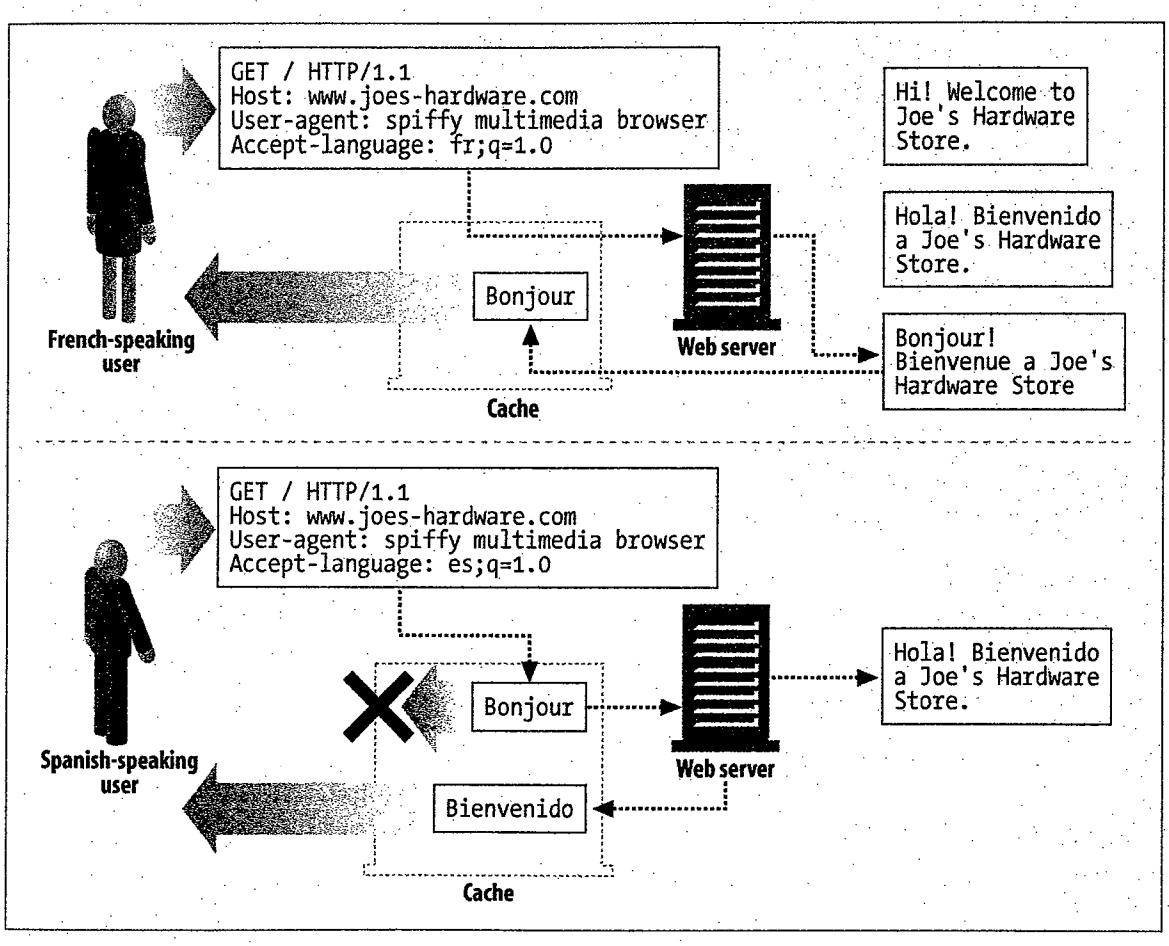


Figure 17-1. Caches use content-negotiation headers to send back correct responses to clients

The cache must therefore forward the second request to the server as well, and store both the response and an "alternate" response for that URL. The cache now has two

different documents for the same URL, just as the server does. These different versions are called *variants* or *alternates*. Content negotiation can be thought of as the process of selecting, from the variants, the best match for a client request.

The Vary Header

Here's a typical set of request and response headers from a browser and server:

```
GET http://www.joes-hardware.com/ HTTP/1.0
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.73 [en] (WinNT; U)
Host: www.joes-hardware.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en, pdf
Accept-Charset: iso-8859-1, *, utf-8

HTTP/1.1 200 OK
Date: Sun, 10 Dec 2000 22:13:40 GMT
Server: Apache/1.3.12 OpenSSL/0.9.5a (Unix) FrontPage/4.0.4.3
Last-Modified: Fri, 05 May 2000 04:42:52 GMT
Etag: "1b7ddf-48-3912514c"
Accept-Ranges: Bytes
Content-Length: 72
Connection: close
Content-Type: text/html
```

What happens, however, if the server's decision was based on headers other than the Accept headers, such as the User-Agent header? This is not as radical as it may sound. Servers may know that old versions of a browser do not support JavaScript, for example, and may therefore send back a version of the page that does not have JavaScript in it. If servers are using other headers to make their decisions about which pages to send back, caches must know what those headers are, so that they can perform parallel logic in choosing which cached page to send back.

The HTTP Vary response header lists all of the client request headers that the server considers to select the document or generate custom content (in addition to the regular content-negotiation headers). For example, if the served document depends on the User-Agent header, the Vary header must include "User-Agent".

When a new request arrives, the cache finds the best match using the content-negotiation headers. Before it can serve this document to the client, however, it must see whether the server sent a Vary header in the cached response. If a Vary header is present, the header values for the headers in the new request must match the header values in the old, cached request. Because servers may vary their responses based on client request headers, caches must store both the client request headers and the corresponding server response headers with each cached variant, in order to implement transparent negotiation. This is illustrated in Figure 17-2.

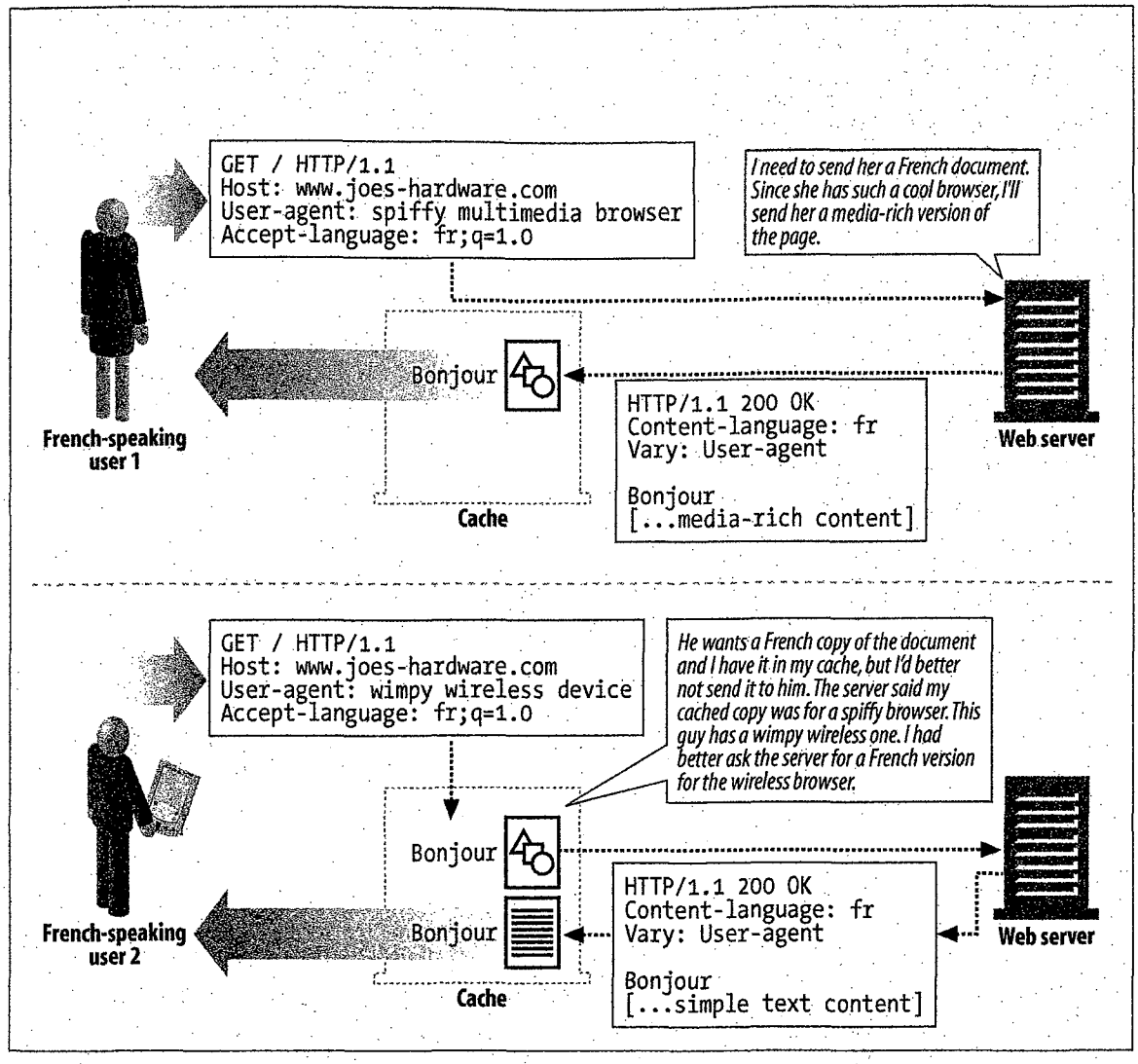


Figure 17-2. If servers vary on specific request headers, caches must match those request headers in addition to the regular content-negotiation headers before sending back cached responses

If a server's Vary header looked like this, the huge number of different User-Agent and Cookie values could generate many variants:

```
Vary: User-Agent, Cookie
```

A cache would have to store each document version corresponding to each variant. When the cache does a lookup, it first does content matching with the content-negotiation headers, then matches the request's variant with cached variants. If there is no match, the cache fetches the document from the origin server.

Transcoding

We have discussed in some detail the mechanism by which clients and servers can choose between a set of documents for a URL and send the one that best matches the

client's needs. These mechanisms rely on the presence of documents that match the client's needs—whether they match the needs perfectly or not so well.

What happens, however, when a server does not have a document that matches the client's needs at all? The server may have to respond with an error, but theoretically, the server may be able to transform one of its existing documents into something that the client can use. This option is called *transcoding*.

Table 17-4 lists some hypothetical transcodings.

Table 17-4. Hypothetical transcodings

Before	After
HTML document	WML document
High-resolution image	Low-resolution image
Image in 64K colors	Black-and-white image
Complex page with frames	Simple text page without frames or images
HTML page with Java applets	HTML page without Java applets
Page with ads	Page with ads removed

There are three categories of transcoding: format conversion, information synthesis, and content injection.

Format Conversion

Format conversion is the transformation of data from one format to another to make it viewable by a client. A wireless device seeking to access a document typically viewed by a desktop client may be able do so with an HTML-to-WML conversion. A client accessing a web page over a slow link that is not very interested in high-resolution images may be able to view an image-rich page more easily if the images are reduced in size and resolution by converting them from color to black and white and shrinking them.

Format conversion is driven by the content-negotiation headers listed in Table 17-2, although it may also be driven by the User-Agent header. Note that content transformation or transcoding is different from content encoding or transfer encoding, in that the latter two typically are used for more efficient or safe transport of content, whereas the former is used to make content viewable on the access device.

Information Synthesis

The extraction of key pieces of information from a document—known as *information synthesis*—can be a useful transcoding process. A simple example of this is the generation of an outline of a document based on section headings, or the removal of advertisements and logos from a page.

More sophisticated technologies that categorize pages based on keywords in content also are useful in summarizing the essence of a document. This technology often is used by automatic web page–classification systems, such as web-page directories at portal sites.

Content Injection

The two categories of transcodings described so far typically reduce the amount of content in web documents, but there is another category of transformations that increases the amount of content: *content-injection* transcodings. Examples of content-injection transcodings are automatic ad generators and user-tracking systems.

Imagine the appeal (and offence) of an ad-insertion transcoder that automatically adds advertisements to each HTML page as it goes by. Transcoding of this type has to be dynamic—it must be done on the fly in order to be effective in adding ads that currently are relevant or somehow have been targeted for a particular user. User-tracking systems also can be built to add content to pages dynamically, for the purpose of collecting statistics about how the page is viewed and how clients surf the Web.

Transcoding Versus Static Pregeneration

An alternative to transcodings is to build different copies of web pages at the web server—for example, one with HTML, one with WML, one with high-resolution images, one with low-resolution images, one with multimedia content, and one without. This, however, is not a very practical technique, for many reasons: any small change in a page requires multiple pages to be modified, more space is necessary to store all the different versions of each page, and it's harder to catalog pages and program web servers to serve the right ones. Some transcodings, such as ad insertion (especially targeted ad insertion), cannot be done statically—the ad inserted will depend upon the user requesting the page.

An on-the-fly transformation of a single root page can be an easier solution than static pregeneration. It can come, however, at the cost of increased latency in serving the content. Some of this computation can, however, be done by a third party, thereby offloading the computation from the web server—the transformation can be done by an external agent at a proxy or cache. Figure 17-3 illustrates transcoding at a proxy cache.

Next Steps

The story of content negotiation does not end with the Accept and Content headers, for a couple of reasons:

- Content negotiation in HTTP incurs some performance limits. Searching through many variants for appropriate content, or trying to "guess" the best match, can

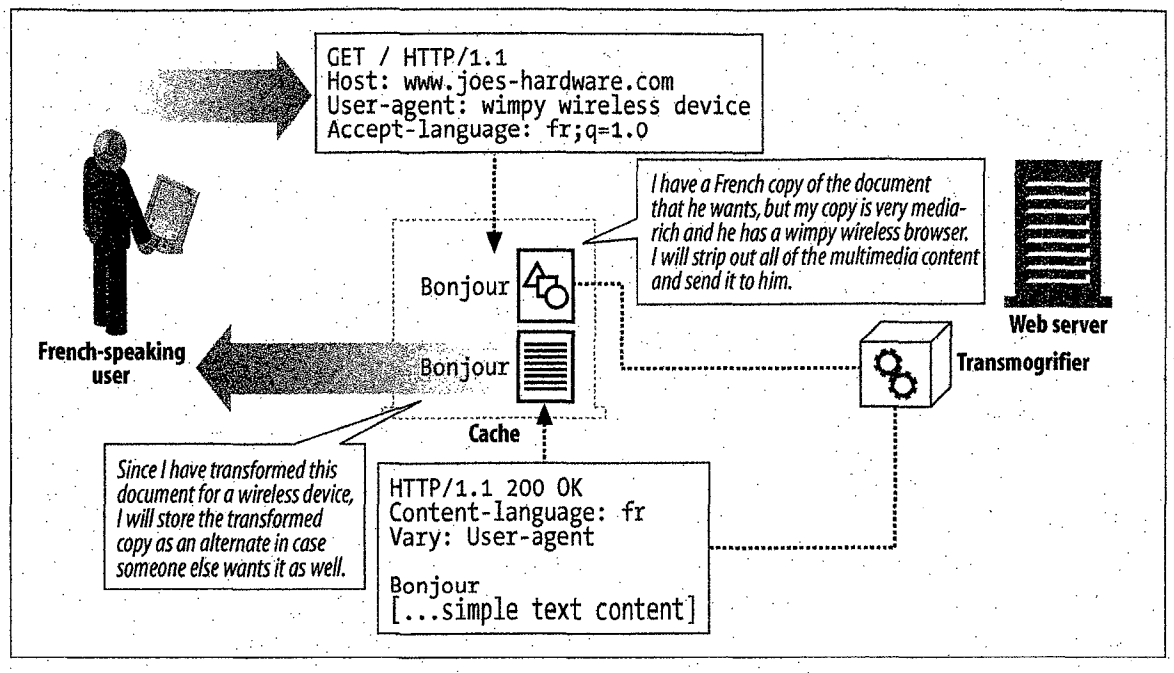


Figure 17-3. Content transformation or transcoding at a proxy cache

be costly. Are there ways to streamline and focus the content-negotiation protocol? RFCs 2295 and 2296 attempt to address this question for transparent HTTP content negotiation.

- HTTP is not the only protocol that needs to do content negotiation. Streaming media and fax are two other examples where client and server need to discuss the best answer to the client's request. Can a general content-negotiation protocol be developed on top of TCP/IP application protocols? The Content Negotiation Working Group was formed to tackle this question. The group is now closed, but it contributed several RFCs. See the next section for a link to the group's web site.

For More Information

The following Internet drafts and online documentation can give you more details about content negotiation:

http://www.ietf.org/rfc/rfc2616.txt

RFC 2616, "Hypertext Transfer Protocol—HTTP/1.1," is the official specification for HTTP/1.1, the current version of the HTTP protocol. The specification is a well-written, well-organized, detailed reference for HTTP, but it isn't ideal for readers who want to learn the underlying concepts and motivations of HTTP or the differences between theory and practice. We hope that this book fills in the underlying concepts, so you can make better use of the specification.

http://search.ietf.org/rfc/rfc2295.txt

RFC 2295, "Transparent Content Negotiation in HTTP," is a memo describing a transparent content-negotiation protocol on top of HTTP. The status of this memo remains experimental.

http://search.ietf.org/rfc/rfc2296.txt

RFC 2296, "HTTP Remote Variant Selection Algorithm—RVSA 1.0," is a memo describing an algorithm for the transparent selection of the "best" content for a particular HTTP request. The status of this memo remains experimental.

http://search.ietf.org/rfc/rfc2936.txt

RFC 2936, "HTTP MIME Type Handler Detection," is a memo describing an approach for determining the actual MIME type handlers that a browser supports. This approach can help if the Accept header is not specific enough.

http://www.imc.org/ietf-medfree/index.htm

This is a link to the Content Negotiation (CONNEG) Working Group, which looked into transparent content negotiation for HTTP, fax, and print. This group is now closed.

PART V

Content Publishing and Distribution

Part V talks all about the technology for publishing and disseminating web content:

- Chapter 18, *Web Hosting*, discusses the ways people deploy servers in modern web hosting environments, HTTP support for virtual web hosting, and how to replicate content across geographically distant servers.
- Chapter 19, *Publishing Systems*, discusses the technologies for creating web content and installing it onto web servers.
- Chapter 20, *Redirection and Load Balancing*, surveys the tools and techniques for distributing incoming web traffic among a collection of servers.
- Chapter 21, *Logging and Usage Tracking*, covers log formats and common questions.

CHAPTER 18

Web Hosting

When you place resources on a public web server, you make them available to the Internet community. These resources can be as simple as text files or images, or as complicated as real-time driving maps or e-commerce shopping gateways. It's critical that this rich variety of resources, owned by different organizations, can be conveniently published to web sites and placed on web servers that offer good performance at a fair price.

The collective duties of storing, brokering, and administering content resources is called *web hosting*. Hosting is one of the primary functions of a web server. You need a server to hold, serve, log access to, and administer your content. If you don't want to manage the required hardware and software yourself, you need a hosting service, or *hoster*. Hosters rent you serving and web-site administration services and provide various degrees of security, reporting, and ease of use. Hosters typically pool web sites on heavy-duty web servers for cost-efficiency, reliability, and performance.

This chapter explains some of the most important features of web hosting services and how they interact with HTTP applications. In particular, this chapter covers:

- How different web sites can be "virtually hosted" on the same server, and how this affects HTTP
- How to make web sites more reliable under heavy traffic
- How to make web sites load faster

Hosting Services

In the early days of the World Wide Web, individual organizations purchased their own computer hardware, built their own computer rooms, acquired their own network connections, and managed their own web server software.

As the Web quickly became mainstream, everyone wanted a web site, but few people had the skills or time to build air-conditioned server rooms, register domain

names, or purchase network bandwidth. To save the day, many new businesses emerged, offering professionally managed web hosting services. Many levels of service are available, from physical facilities management (providing space, air conditioning, and wiring) to full-service web hosting, where all the customer does is provide the content.

This chapter focuses on what the hosting web server provides. Much of what makes a web site work—as well as, for example, its ability to support different languages and its ability to do secure e-commerce transactions—depends on what capabilities the hosting web server supports.

A Simple Example: Dedicated Hosting

Suppose that Joe's Hardware Online and Mary's Antique Auction both want fairly high-volume web sites. Irene's ISP has racks and racks full of identical, high-performance web servers that it can lease to Joe and Mary, instead of having Joe and Mary purchase their own servers and maintain the server software.

In Figure 18-1, both Joe and Mary sign up for the *dedicated web hosting* service offered by Irene's ISP. Joe leases a dedicated web server that is purchased and maintained by Irene's ISP. Mary gets a different dedicated server from Irene's ISP. Irene's ISP gets to buy server hardware in volume and can select hardware that is reliable, time-tested, and low-cost. If either Joe's Hardware Online or Mary's Antique Auction grows in popularity, Irene's ISP can offer Joe or Mary additional servers immediately.

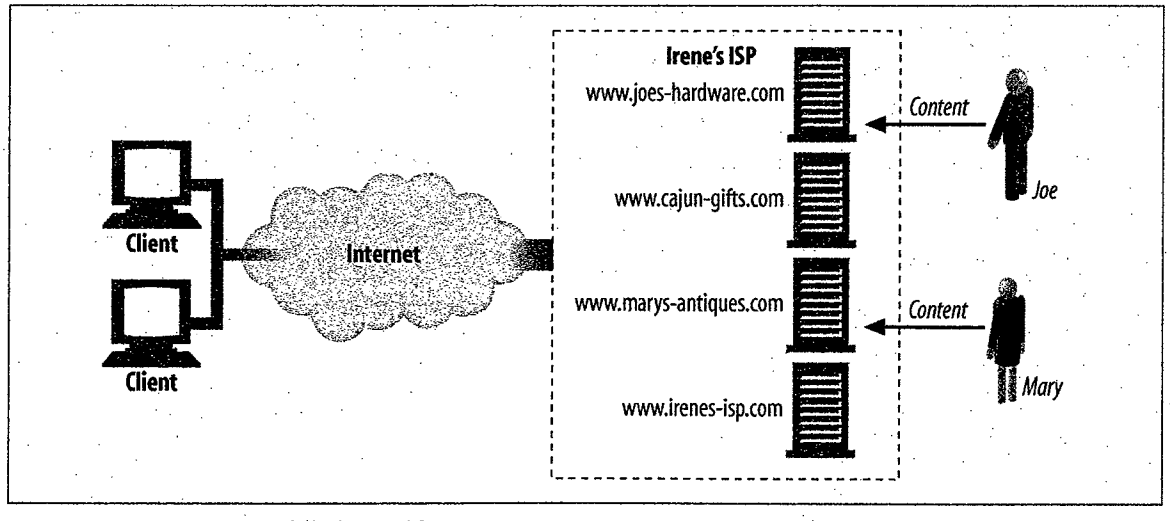


Figure 18-1. Outsourced dedicated hosting

In this example, browsers send HTTP requests for *www.joes-hardware.com* to the IP address of Joe's server and requests for *www.marys-antiques.com* to the (different) IP address of Mary's server.

Virtual Hosting

Many folks want to have a web presence but don't have high-traffic web sites. For these people, providing a dedicated web server may be a waste, because they're paying many hundreds of dollars a month to lease a server that is mostly idle!

Many web hosters offer lower-cost web hosting services by sharing one computer between several customers. This is called *shared hosting* or *virtual hosting*. Each web site appears to be hosted by a different server, but they really are hosted on the same physical server. From the end user's perspective, virtually hosted web sites should be indistinguishable from sites hosted on separate dedicated servers.

For cost efficiency, space, and management reasons, a virtual hosting company wants to host tens, hundreds, or thousands of web sites on the same server—but this does not necessarily mean that 1,000 web sites are served from only one PC. Hosters can create banks of replicated servers (called *server farms*) and spread the load across the farm of servers. Because each server in the farm is a clone of the others, and hosts many virtual web sites, administration is much easier. (We'll talk more about server farms in Chapter 20.)

When Joe and Mary started their businesses, they might have chosen virtual hosting to save money until their traffic levels made a dedicated server worthwhile (see Figure 18-2).

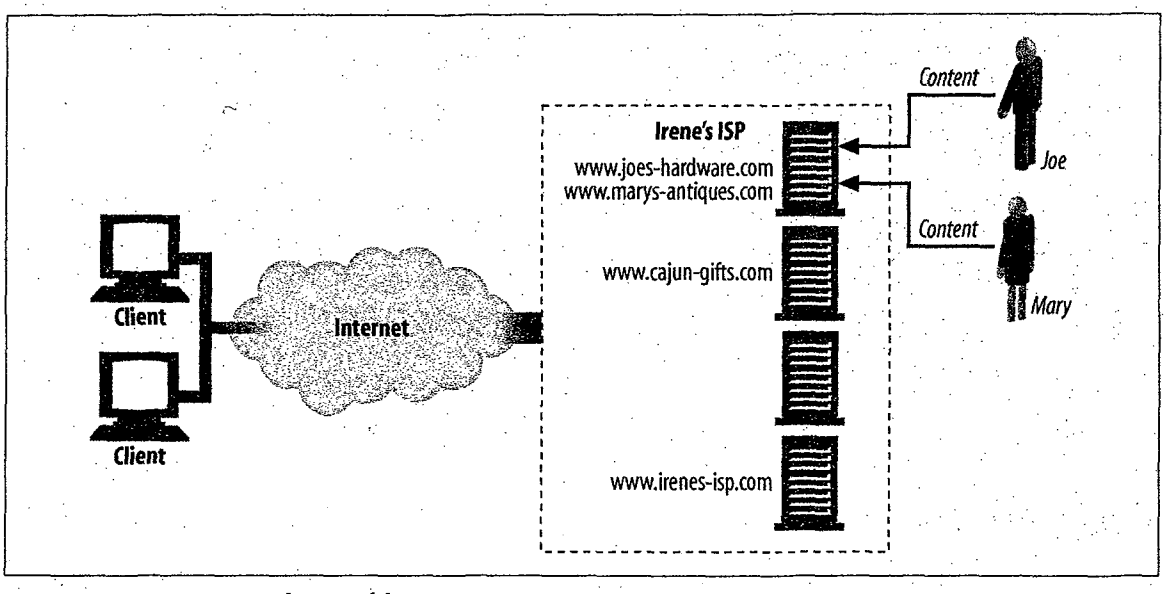


Figure 18-2. Outsourced virtual hosting

Virtual Server Request Lacks Host Information

Unfortunately, there is a design flaw in HTTP/1.0 that makes virtual hosters pull their hair out. The HTTP/1.0 specification didn't give any means for shared web servers to identify which of the virtual web sites they're hosting is being accessed.

Recall that HTTP/1.0 requests send only the path component of the URL in the request message. If you try to get *http://www.joes-hardware.com/index.html*, the browser connects to the server *www.joes-hardware.com*, but the HTTP/1.0 request says "GET /index.html", with no further mention of the hostname. If the server is virtually hosting multiple sites, this isn't enough information to figure out what virtual web site is being accessed. For example, in Figure 18-3:

- If client A tries to access *http://www.joes-hardware.com/index.html*, the request "GET /index.html" will be sent to the shared web server.
- If client B tries to access *http://www.marys-antiques.com/index.html*, the identical request "GET /index.html" will be sent to the shared web server.

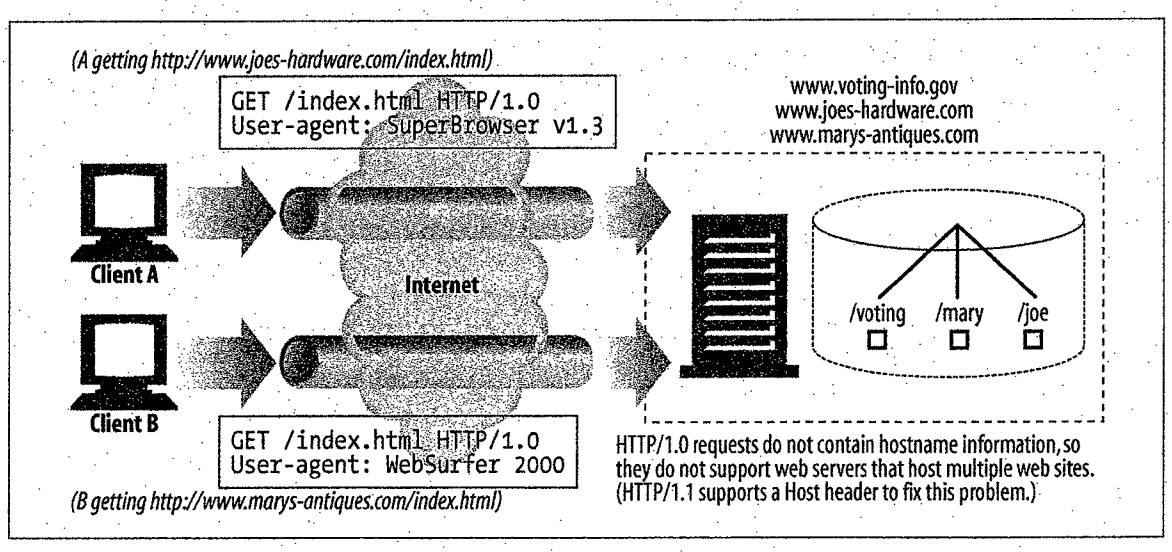


Figure 18-3. HTTP/1.0 server requests don't contain hostname information

As far as the web server is concerned, there is not enough information to determine which web site is being accessed! The two requests look the same, even though they are for totally different documents (from different web sites). The problem is that the web site host information has been stripped from the request.

As we saw in Chapter 6, HTTP surrogates (reverse proxies) and intercepting proxies also need site-specifying information.

Making Virtual Hosting Work

The missing host information was an oversight in the original HTTP specification, which mistakenly assumed that each web server would host exactly one web site. HTTP's designers didn't provide support for virtually hosted, shared servers. For this reason, the hostname information in the URL was viewed as redundant and stripped away; only the path component was required to be sent.

Because the early specifications did not make provisions for virtual hosting, web hosters needed to develop workarounds and conventions to support shared virtual hosting. The problem could have been solved simply by requiring all HTTP request

messages to send the full URL instead of just the path component. HTTP/1.1 does require servers to handle full URLs in the request lines of HTTP messages, but it will be a long time before all legacy applications are upgraded to this specification. In the meantime, four techniques have emerged:

Virtual hosting by URL path
: Adding a special path component to the URL so the server can determine the site.

Virtual hosting by port number
: Assigning a different port number to each site, so requests are handled by separate instances of the web server.

Virtual hosting by IP address
: Dedicating different IP addresses for different virtual sites and binding all the IP addresses to a single machine. This allows the web server to identify the site name by IP address.

Virtual hosting by Host header
: Many web hosters pressured the HTTP designers to solve this problem. Enhanced versions of HTTP/1.0 and the official version of HTTP/1.1 define a Host request header that carries the site name. The web server can identify the virtual site from the Host header.

Let's take a closer look at each technique.

Virtual hosting by URL path

You can use brute force to isolate virtual sites on a shared server by assigning them different URL paths. For example, you could give each logical web site a special path prefix:

- Joe's Hardware store could be *http://www.joes-hardware.com/joe/index.html*.
- Mary's Antiques store could be *http://www.marys-antiques.com/mary/index.html*.

When the requests arrive at the server, the hostname information is not present in the request, but the server can tell them apart based on the path:

- The request for Joe's hardware is "GET /joe/index.html".
- The request for Mary's antiques is "GET /mary/index.html".

This is not a good solution. The "/joe" and "/mary" prefixes are redundant and confusing (we already mentioned "joe" in the hostname). Worse, the common convention of specifying *http://www.joes-hardware.com* or *http://www.joes-hardware.com/index.html* for the home page won't work.

In general, URL-based virtual hosting is a poor solution and seldom is used.

Virtual hosting by port number

Instead of changing the pathname, Joe and Mary could each be assigned a different port number on the web server. Instead of port 80, for example, Joe could get 82 and

Mary could have 83. But this solution has the same problem: an end user would expect to find the resources without having to specify a nonstandard port in the URL.

Virtual hosting by IP address

A much better approach (in common use) is virtual IP addressing. Here, each virtual web site gets one or more unique IP addresses. The IP addresses for all of the virtual web sites are attached to the same shared server. The server can look up the destination IP address of the HTTP connection and use that to determine what web site the client thinks it is connected to.

Say a hoster assigned the IP address 209.172.34.3 to *www.joes-hardware.com*, assigned 209.172.34.4 to *www.marys-antiques.com*, and tied both IP addresses to the same physical server machine. The web server could then use the destination IP address to identify which virtual site is being requested, as shown in Figure 18-4:

- Client A fetches *http://www.joes-hardware.com/index.html*.
- Client A finds the IP address for *www.joes-hardware.com*, getting 209.172.34.3.
- Client A opens a TCP connection to the shared web server at 209.172.34.3.
- Client A sends the request "GET /index.html HTTP/1.0".
- Before the web server serves a response, it notes the actual destination IP address (209.172.34.3), determines that this is a virtual IP address for Joe's web site, and fulfills the request from the */joe* subdirectory. The page */joe/index.html* is returned.

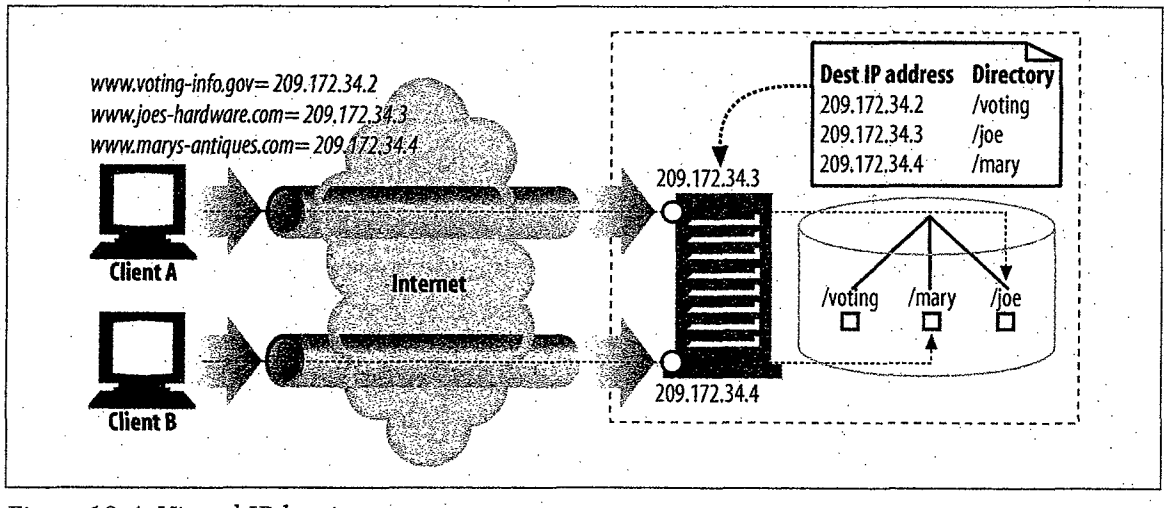


Figure 18-4. Virtual IP hosting

Similarly, if client B asks for *http://www.marys-antiques.com/index.html*:

- Client B finds the IP address for *www.marys-antiques.com*, getting 209.172.34.4.
- Client B opens a TCP connection to the web server at 209.172.34.4.
- Client B sends the request "GET /index.html HTTP/1.0".
- The web server determines that 209.172.34.4 is Mary's web site and fulfills the request from the */mary* subdirectory, returning the document */mary/index.html*.

Virtual IP hosting works, but it causes some difficulties, especially for large hosters:

- Computer systems usually have a limit on how many virtual IP addresses can be bound to a machine. Hosters that want hundreds or thousands of virtual sites to be hosted on a shared server may be out of luck.
- IP addresses are a scarce commodity. Hosters with many virtual sites might not be able to obtain enough virtual IP addresses for the hosted web sites.
- The IP address shortage is made worse when hosters replicate their servers for additional capacity. Different virtual IP addresses may be needed on each replicated server, depending on the load-balancing architecture, so the number of IP addresses needed can multiply by the number of replicated servers.

Despite the address consumption problems with virtual IP hosting, it is used widely.

Virtual hosting by Host header

To avoid excessive address consumption and virtual IP limits, we'd like to share the same IP address among virtual sites, but still be able to tell the sites apart. But as we've seen, because most browsers send just the path component of the URL to servers, the critical virtual hostname information is lost.

To solve this problem, browser and server implementors extended HTTP to provide the original hostname to servers. But browsers couldn't just send a full URL, because that would break many servers that expected to receive only a path component. Instead, the hostname (and port) is passed in a Host extension header in all requests.

In Figure 18-5, client A and client B both send Host headers that carry the original hostname being accessed. When the server gets the request for */index.html*, it can use the Host header to decide which resources to use.

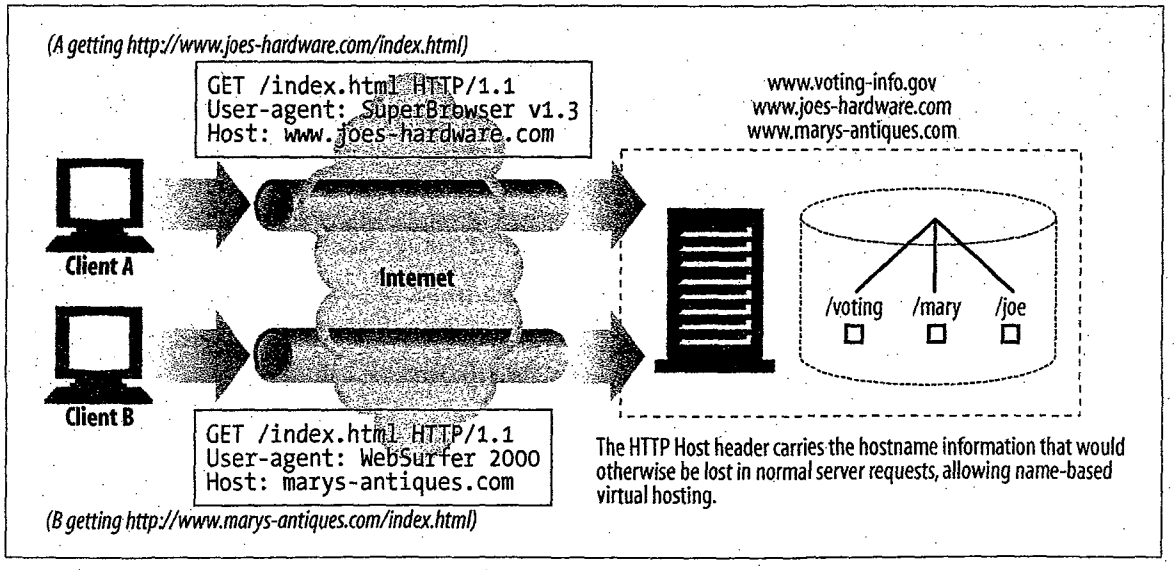


Figure 18-5. Host headers distinguish virtual host requests

Host headers were first introduced with HTTP/1.0+, a vendor-extended superset of HTTP/1.0. Host headers are required for HTTP/1.1 compliance. Host headers are supported by most modern browsers and servers, but there are still a few clients and servers (and robots) that don't support them.

HTTP/1.1 Host Headers

The Host header is an HTTP/1.1 request header, defined in RFC 2068. Virtual servers are so common that most HTTP clients, even if they are not HTTP/1.1-compliant, implement the Host header.

Syntax and usage

The Host header specifies the Internet host and port number for the resource being requested, as obtained from the original URL:

```
Host = "Host" ":" host [ ":" port ]
```

In particular:

- If the Host header does not contain a port, the default port for the scheme is assumed.
- If the URL contains an IP address, the Host header should contain the same address.
- If the URL contains a hostname, the Host header must contain the same name.
- If the URL contains a hostname, the Host header should *not* contain the IP address equivalent to the URL's hostname, because this will break virtually hosted servers, which layer multiple virtual sites over a single IP address.
- If the URL contains a hostname, the Host header should not contain another alias for this hostname, because this also will break virtually hosted servers.
- If the client is using an explicit proxy server, the client must include the name and port of the *origin* server in the Host header, *not* the proxy server. In the past, several web clients had bugs where the outgoing Host header was set to the hostname of the proxy, when the client's proxy setting was enabled. This incorrect behavior causes proxies and origin servers to misbehave.
- Web clients must include a Host header field in all request messages.
- Web proxies must add Host headers to request messages before forwarding them.
- HTTP/1.1 web servers must respond with a 400 status code to any HTTP/1.1 request message that lacks a Host header field.

Here is a sample HTTP request message used to fetch the home page of *www.joes-hardware.com*, along with the required Host header field:

```
GET http://www.joes-hardware.com/index.html HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Host: www.joes-hardware.com
```

Missing Host headers

A small percentage of old browsers in use do not send Host headers. If a virtual hosting server is using Host headers to determine which web site to serve, and no Host header is present, it probably will either direct the user to a default web page (such as the web page of the ISP) or return an error page suggesting that the user upgrade her browser.

Interpreting Host headers

An origin server that isn't virtually hosted, and doesn't allow resources to differ by the requested host, may ignore the Host header field value. But any origin server that does differentiate resources based on the host must use the following rules for determining the requested resource on an HTTP/1.1 request:

1. If the URL in the HTTP request message is absolute (i.e., contains a scheme and host component), the value in the Host header is ignored in favor of the URL.
2. If the URL in the HTTP request message doesn't have a host, and the request contains a Host header, the value of the host/port is obtained from the Host header.
3. If no valid host can be determined through Steps 1 or 2, a 400 Bad Response response is returned to the client.

Host headers and proxies

Some browser versions send incorrect Host headers, especially when configured to use proxies. For example, when configured to use a proxy, some older versions of Apple and PointCast clients mistakenly sent the name of the proxy instead of the origin server in the Host header.

Making Web Sites Reliable

There are several times during which web sites commonly break:

- Server downtime
- Traffic spikes: suddenly everyone wants to see a particular news broadcast or rush to a sale. Sudden spikes can overload a web server, slowing it down or stopping it completely.
- Network outages or losses

This section presents some ways of anticipating and dealing with these common problems.

Mirrored Server Farms

A server farm is a bank of identically configured web servers that can cover for each other. The content on each server in the farm can be mirrored, so that if one has a problem, another can fill in.

Often, mirrored servers follow a hierarchical relationship. One server might act as the "content authority"—the server that contains the original content (perhaps a server to which the content authors post). This server is called the *master origin server*. The mirrored servers that receive content from the master origin server are called *replica origin servers*. One simple way to deploy a server farm is to use a network switch to distribute requests to the servers. The IP address for each of the web sites hosted on the servers is the IP address of the switch.

In the mirrored server farm shown in Figure 18-6, the master origin server is responsible for sending content to the replica origin servers. To the outside world, the IP address for this content is the IP address of the switch. The switch is responsible for sending requests to the servers.

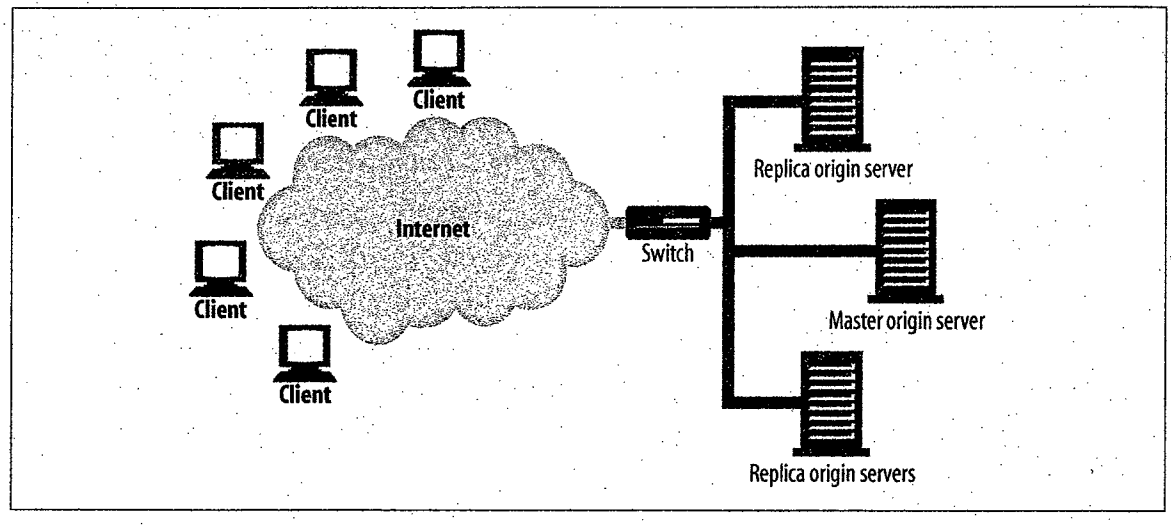


Figure 18-6. Mirrored server farm

Mirrored web servers can contain copies of the exact same content at different locations. Figure 18-7 illustrates four mirrored servers, with a master server in Chicago and replicas in New York, Miami, and Little Rock. The master server serves clients in the Chicago area and also has the job of propagating its content to the replica servers.

In the Figure 18-7 scenario, there are a couple of ways that client requests would be directed to a particular server:

HTTP redirection
: The URL for the content could resolve to the IP address of the master server, which could then send redirects to replica servers.

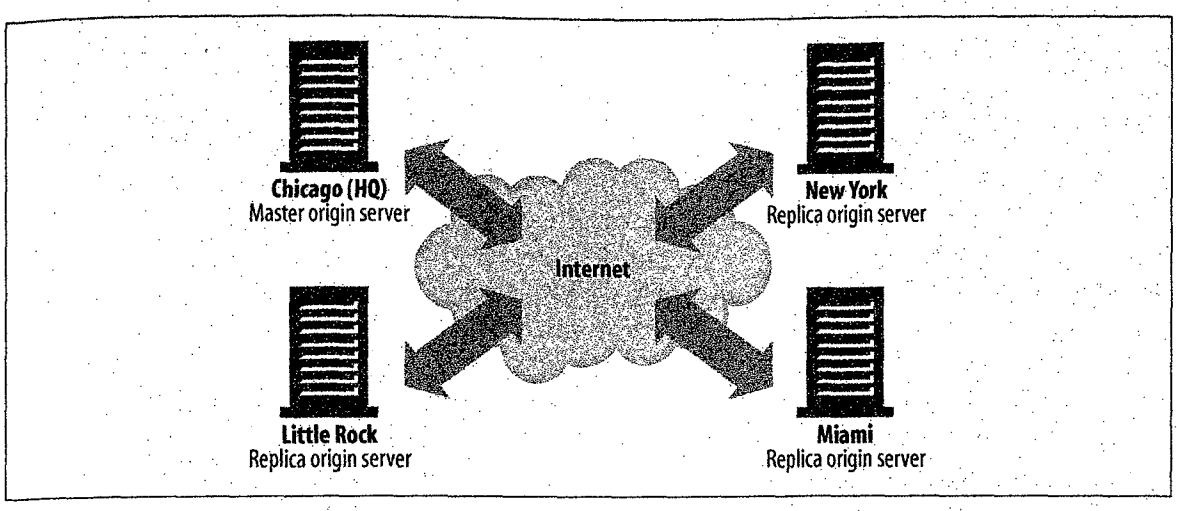


Figure 18-7. Dispersed mirrored servers

DNS redirection

The URL for the content could resolve to four IP addresses, and the DNS server could choose the IP address that it sends to clients.

See Chapter 20 for more details.

Content Distribution Networks

A *content distribution network* (CDN) is simply a network whose purpose is the distribution of specific content. The nodes of the network can be web servers, surrogates, or proxy caches.

Surrogate Caches in CDNs

Surrogate caches can be used in place of replica origin servers in Figures 18-6 and 18-7. Surrogates, also known as reverse proxies, receive server requests for content just as mirrored web servers do. They receive server requests on behalf of a specific set of origin servers (this is possible because of the way IP addresses for content are advertised; there usually is a working relationship between origin server and surrogate, and surrogates expect to receive requests aimed at specific origin servers).

The difference between a surrogate and a mirrored server is that surrogates typically are demand-driven. They do not store entire copies of the origin server content; they store whatever content their clients request. The way content is distributed in their caches depends on the requests that they receive; the origin server does not have the responsibility to update their content. For easy access to "hot" content (content that is in high demand), some surrogates have "prefetching" features that enable them to pull content in advance of user requests.

An added complexity in CDNs with surrogates is the possibility of cache hierarchies.

Proxy Caches in CDNs

Proxy caches also can be deployed in configurations similar to those in Figures 18-6 and 18-7. Unlike surrogates, traditional proxy caches can receive requests aimed at any web servers (there need not be any working relationship or IP address agreement between a proxy cache and an origin server). As with surrogates, however, proxy cache content typically is demand-driven and is not expected to be an exact duplicate of the origin server content. Some proxy caches also can be preloaded with hot content.

Demand-driven proxy caches can be deployed in other kinds of configurations—in particular, interception configurations, where a layer-2 or -3 device (switch or router) intercepts web traffic and sends it to a proxy cache (see Figure 18-8).

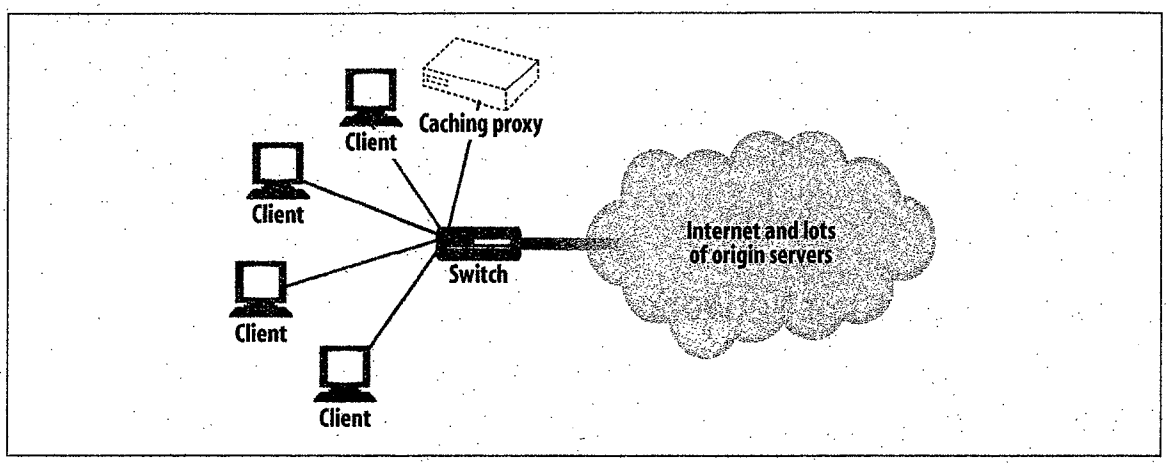


Figure 18-8. Client requests intercepted by a switch and sent to a proxy

An interception configuration depends on being able to set up the network between clients and servers so that all of the appropriate HTTP requests are physically channeled to the cache. (See Chapter 20). The content is distributed in the cache according to the requests it receives.

Making Web Sites Fast

Many of the technologies mentioned in the previous section also help web sites load faster. Server farms and distributed proxy caches or surrogate servers distribute network traffic, avoiding congestion. Distributing the content brings it closer to end users, so that the travel time from server to client is lower. The key to speed of resource access is how requests and responses are directed from client to server and back across the Internet. See Chapter 20 for details on redirection methods.

Another approach to speeding up web sites is encoding the content for fast transportation. This can mean, for example, compressing the content, assuming that the receiving client can uncompress it. See Chapter 15 for details.

For More Information

See Part III, *Identification, Authorization, and Security*, for details on how to make web sites secure. The following Internet drafts and documentation can give you more details about web hosting and content distribution:

http://www.ietf.org/rfc/rfc3040.txt
: RFC 3040, "Internet Web Replication and Caching Taxonomy," is a reference for the vocabulary of web replication and caching applications.

http://search.ietf.org/internet-drafts/draft-ietf-cdi-request-routing-reqs-00.txt
: "Request-Routing Requirements for Content Internetworking."

Apache: The Definitive Guide
: Ben Laurie and Peter Laurie, O'Reilly & Associates, Inc. This book describes how to run the open source Apache web server.

CHAPTER 19

Publishing Systems

How do you create web pages and get them onto a web server? In the dark ages of the Web (let's say, 1995), you might have hand-crafted your HTML in a text editor and manually uploaded the content to the web server using FTP. This procedure was painful, difficult to coordinate with coworkers, and not particularly secure.

Modern-day publishing tools make it much more convenient to create, publish, and manage web content. Today, you can interactively edit web content as you'll see it on the screen and publish that content to servers with a single click, while being notified of any files that have changed.

Many of the tools that support remote publishing of content use extensions to the HTTP protocol. In this chapter, we explain two important technologies for web-content publishing based on HTTP: FrontPage and DAV.

FrontPage Server Extensions for Publishing Support

FrontPage (commonly referred to as FP) is a versatile web authoring and publishing toolkit provided by Microsoft Corp. The original idea for FrontPage (FrontPage 1.0) was conceived in 1994, at Vermeer Technologies, Inc., and was dubbed the first product to combine web site management and creation into a single, unified tool. Microsoft purchased Vermeer and shipped FrontPage 1.1 in 1996. The latest version, FrontPage Version 2002, is the sixth version in the line and a core part of the Microsoft Office suite.

FrontPage Server Extensions

As part of the "publish anywhere" strategy, Microsoft released a set of server-side software called FrontPage Server Extensions (FPSE). These server-side components integrate with the web server and provide the necessary translation between the web site and the client running FrontPage (and other clients that support these extensions).

Our primary interest lies in the publishing protocol between the FP clients and FPSE. This protocol provides an example of designing extensions to the core services available in HTTP without changing HTTP semantics.

The FrontPage publishing protocol implements an RPC layer on top of the HTTP POST request. This allows the FrontPage client to send commands to the server to update documents on the web site, perform searches, collaborate amongst the web authors, etc. Figure 19-1 gives an overview of the communication.

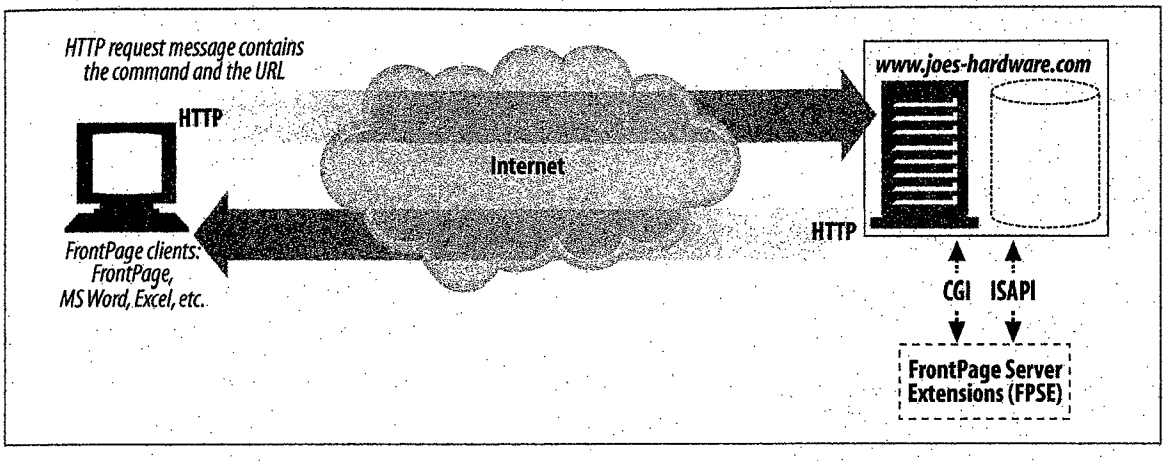


Figure 19-1. FrontPage publishing architecture

The web server sees POST requests addressed to the FPSE (implemented as a set of CGI programs, in the case of a non-Microsoft IIS server) and directs those requests accordingly. As long as intervening firewalls and proxy servers are configured to allow the POST method, FrontPage can continue communicating with the server.

FrontPage Vocabulary

Before we dive deeper into the RPC layer defined by FPSE, it may help to establish the common vocabulary:

Virtual server

One of the multiple web sites running on the same server, each with a unique domain name and IP address. In essence, a virtual server allows a single web server to host multiple web sites, each of which appears to a browser as being hosted by its own web server. A web server that supports virtual servers is called a *multi-hosting* web server. A machine that is configured with multiple IP addresses is called a *multi-homed* server (for more details, please refer to "Virtual Hosting" in Chapter 18).

Root web

The default, top-level content directory of a web server, or, in a multi-hosting environment, the top-level content directory of a virtual web server. To access the root web, it is enough to specify the URL of the server without specifying a page name. There can be only one root web per web server.

Subweb

A named subdirectory of the root web or another subweb that is a complete FPSE extended web. A subweb can be a complete independent entity with the ability to specify its own administration and authoring permissions. In addition, subwebs may provide scoping for methods such as searches.

The FrontPage RPC Protocol

The FrontPage client and FPSE communicate using a proprietary RPC protocol. This protocol is layered on top of HTTP POST by embedding the RPC methods and their associated variables in the body of the POST request.

To start the process, the client needs to determine the location and the name of the target programs on the server (the part of the FPSE package that can execute the POST request). It then issues a special GET request (see Figure 19-2).

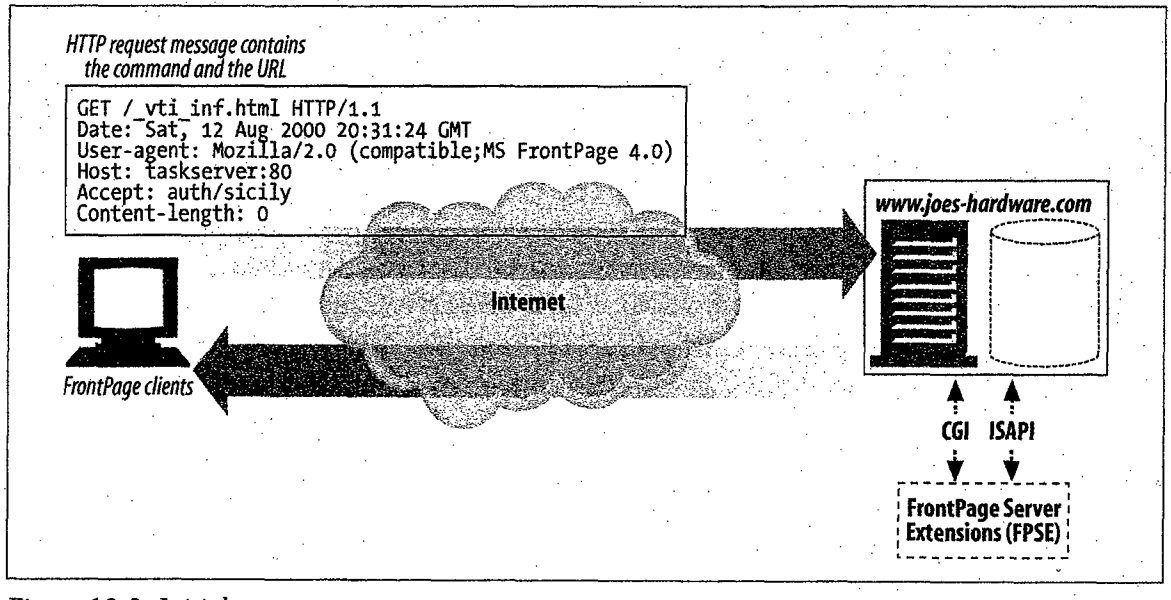


Figure 19-2. Initial request

When the file is returned, the FrontPage client reads the response and finds the values associated with `FPShtmlScriptUrl`, `FPAuthorScriptUrl`, and `FPAdminScriptUrl`. Typically, this may look like:

```
FPShtmlScriptUrl="_vti_bin/_vti_rpc/shtml.dll"
FPAuthorScriptUrl="_vti_bin/_vti_aut/author.dll"
FPAdminScriptUrl="_vti_bin/_vti_adm/admin.dll"
```

`FPShtmlScriptUrl` tells the client where to POST requests for "browse time" commands (e.g., getting the version of FPSE) to be executed.

`FPAuthorScriptUrl` tells the client where to POST requests for "authoring time" commands to be executed. Similarly, `FPAdminScriptUrl` tells FrontPage where to POST requests for administrative actions.

Now that we know where the various programs are located, we are ready to send a request.

Request

The body of the POST request contains the RPC command, in the form of "method=<command>" and any required parameters. For example, consider the RPC message requesting a list of documents, as follows:

```
POST /_vti_bin/_vti_aut/author.dll HTTP/1.1
Date:  Sat, 12 Aug 2000 20:32:54 GMT
User-Agent: MSFrontPage/4.0
.....

<BODY>
method=list+documents%3a4%2e0%2e2%2e3717&service%5fname=&listHiddenDocs=false&listExp
lorerDocs=false&listRecurse=false&listFiles=true&listFolders=true&listLinkInfo=true&l
istIncludeParent=true&listDerived=false
&listBorders=false&listChildWebs=true&initialUrl=&folderList=%5b%3bTW%7c12+Aug+2000+2
0%3a33%3a04+%2d0000%5d
```

The body of the POST command contains the RPC command being sent to the FPSE. As with CGI programs, the spaces in the method are encoded as plus sign (+) characters. All other nonalphanumeric characters in the method are encoded using *%XX* format, where the *XX* stands for the ASCII representation of the character. Using this notation, a more readable version of the body would look like the following:

```
method=list+documents:4.0.1.3717
&service_name=
&listHiddenDocs=false
&listExplorerDocs=false
.....
```

Some of the elements listed are:

service_name
: The URL of the web site on which the method should act. Must be an existing folder or one level below an existing folder.

listHiddenDocs
: Shows the hidden documents in a web if its value is "true". The "hidden" documents are designated by URLs with path components starting with "_".

listExploreDocs
: If the value is "true", lists the task lists.

Response

Most RPC protocol methods have return values. Most common return values are for successful methods and errors. Some methods also have a third subsection, "Sample Return Code." FrontPage properly interprets the codes to provide accurate feedback to the user.

Continuing with our example, the FPSE processes the "list+documents" request and returns the necessary information. A sample response follows:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 12 Aug 2000 22:49:50 GMT
Content-type: application/x-vermeer-rpc
X-FrontPage-User-Name: IUSER_MINSTAR

<html><head><title>RPC packet</title></head>
<body>
<p>method=list documents: 4.0.2.3717
<p>document_list=
<ul>
    <li>document_name=help.gif
<\ul>
```

As you can see from the response, a formatted list of documents available on the web server is returned to the FP client. You can find the complete list of commands and responses at the Microsoft web site.

FrontPage Security Model

Any publishing system directly accessing web server content needs to be very conscious of the security implications of its actions. For the most part, FPSE depends on the web server to provide the security.

The FPSE security model defines three kinds of users: administrators, authors, and browsers, with administrators having complete control. All permissions are cumulative; i.e., all administrators may author and browse the FrontPage web. Similarly, all authors have browsing permissions.

The list of administrators, authors, and browsers is defined for a given FPSE extended web. All of the subwebs may inherit the permissions from the root web or set their own. For non-IIS web servers, all the FPSE programs are required to be stored in directories marked "executable" (the same restriction as for any other CGI program). *Fpsrvadm*, the FrontPage server administrator utility, may be used for this purpose. On IIS servers, the integrated Windows security model prevails.

On non-IIS servers, web server access-control mechanisms specify the users who are allowed to access a given program. On Apache and NCSA web servers, the file is named *.htaccess*; on Netscape servers, it is named *.nsconfig*. The access file associates users, groups, and IP addresses with various levels of permissions: GET (read), POST (execute), etc. For example, for a user to be an author on an Apache web server, the *.htaccess* file should permit that user to POST to *author.exe*. These access-specification files often are defined on a per-directory basis, providing greater flexibility in defining the permissions.

On IIS servers, the permissions are checked against the ACLs for a given root or subroot. When IIS gets a request, it first logs on and impersonates the user, then sends the request to one of the three extension dynamic link libraries (DLLs). The DLL checks the impersonation credentials against the ACL defined for the destination folder. If the check is successful, the requested operation is executed by the extension DLL. Otherwise, a "permission denied" message is sent back to the client. Given the tight integration of Windows security with IIS, the User Manager may be used to define fine-grained control.

In spite of this elaborate security model, enabling FPSE has gained notoriety as a nontrivial security risk. In most cases, this is due to sloppy practices adopted by web site administrators. However, the earlier versions of FPSE did have severe security loopholes and thus contributed to the general perception of security risk. This problem also was exacerbated by the arcane practices needed to fully implement a tight security model.

WebDAV and Collaborative Authoring

Web Distributed Authoring and Versioning (WebDAV) adds an extra dimension to web publishing—collaboration. Currently, the most common practice of collaboration is decidedly low-tech: predominantly email, sometimes combined with distributed fileshares. This practice has proven to be very inconvenient and error-prone, with little or no control over the process. Consider an example of launching a multinational, multilingual web site for an automobile manufacturer. It's easy to see the need for a robust system with secure, reliable publishing primitives, along with collaboration primitives such as locking and versioning.

WebDAV (published as RFC 2518) is focused on extending HTTP to provide a suitable platform for collaborative authoring. It currently is an IETF effort with support from various vendors, including Adobe, Apple, IBM, Microsoft, Netscape, Novell, Oracle, and Xerox.

WebDAV Methods

WebDAV defines a set of new HTTP methods and modifies the operational scope of a few other HTTP methods. The new methods added by WebDAV are:

PROPFIND
: Retrieves the properties of a resource.

PROPPATCH
: Sets one or more properties on one or many resources.

MKCOL
: Creates collections.

COPY
: Copies a resource or a collection of resources from a given source to a given destination. The destination need not be on the same machine.

MOVE
: Moves a resource or a collection of resources from a given source to a given destination. The destination need not be on the same machine.

LOCK
: Locks a resource or multiple resources.

UNLOCK
: Unlocks a previously locked resource.

HTTP methods modified by WebDAV are DELETE, PUT, and OPTIONS. Both the new and the modified methods are discussed in detail later in this chapter.

WebDAV and XML

WebDAV's methods generally require a great deal of information to be associated with both requests and responses. HTTP usually communicates this information in message headers. However, transporting necessary information in headers alone imposes some limitations, including the difficulties of selective application of header information to multiple resources in a request, to represent hierarchy, etc.

To solve this problem, WebDAV embraces the Extensible Markup Language (XML), a meta-markup language that provides a format for describing structured data. XML provides WebDAV with:

- A method of formatting instructions describing how data is to be handled
- A method of formatting complex responses from the server
- A method of communicating customized information about the collections and resources handled
- A flexible vehicle for the data itself
- A robust solution for most of the internationalization issues

Traditionally, the schema definition for XML documents is kept in a Document Type Definition (DTD) file that is referenced within the XML document itself. Therefore, when trying to interpret an XML document, the DOCTYPE definition entity gives the name of the DTD file associated with the XML document in question.

WebDAV defines an explicit XML namespace, "DAV:". Without going into many details, an XML namespace is a collection of names of elements or attributes. The namespace qualifies the embedded names uniquely across the domain, thus avoiding any name collisions.

The complete XML schema is defined in the WebDAV specification, RFC 2518. The presence of a predefined schema allows the parsing software to make assumptions on the XML schema without having to read in DTD files and interpret them correctly.

WebDAV Headers

WebDAV does introduce several HTTP headers to augment the functionality of the new methods. This section provides a brief overview; see RFC 2518 for more information. The new headers are:

DAV

Used to communicate the WebDAV capabilities of the server. All resources supported by WebDAV are required to return this header in the response to the OPTIONS request. See "The OPTIONS method" for more details.

```
DAV = "DAV" ":" "1" ["," "2"] ["," 1#extend]
```

Depth

The crucial element for extending WebDAV to grouped resources with multiple levels of hierarchy (for more detailed explanation about collections, please refer to "Collections and Namespace Management").

```
Depth = "Depth" ":" ("0" | "1" | "infinity")
```

Let's look at a simple example. Consider a directory *DIR_A* with files *file_1.html* and *file_2.html*. If a method uses Depth: 0, the method applies to the *DIR_A* directory alone, and Depth: 1 applies to the *DIR_A* directory and its files, *file_1.html* and *file_2.html*.

The Depth header modifies many WebDAV-defined methods. Some of the methods that use the Depth header are LOCK, COPY, and MOVE.

Destination

Defined to assist the COPY or MOVE methods in identifying the destination URI.

```
Destination = "Destination" ":" absoluteURI
```

If

The only defined state token is a lock token (see "The LOCK Method"). The If header defines a set of conditionals; if they all evaluate to false, the request will fail. Methods such as COPY and PUT conditionalize the applicability by specifying preconditions in the If header. In practice, the most common precondition to be satisfied is the prior acquisition of a lock.

```
If = "If" ":" (1*No-tag-list | 1*Tagged-list)
   No-tag-list = List
   Tagged-list = Resource 1*List
   Resource = Coded-URL
   List = "(" 1*(["Not"](State-token | "[" entity-tag "]")) ")"
   State-token = Coded-URL
   Coded-URL = "<" absoluteURI ">"
```

Lock-Token

Used by the UNLOCK method to specify the lock that needs to be removed. A response to a LOCK method also has a Lock-Token header, carrying the necessary information about the lock token.

```
Lock-Token = "Lock-Token" ":" Coded-URL
```

Overwrite

Used by the COPY and MOVE methods to designate whether the destination should be overwritten. See the discussion of the COPY and MOVE methods later in this chapter for more details.

```
Overwrite = "Overwrite" ":" ("T" | "F")
```

Timeout

A request header used by a client to specify a desired lock timeout value. For more information, refer to the section "Lock refreshes and the Timeout header."

```
TimeOut = "Timeout" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other)
DAVTimeOutVal = 1*digit
Other = "Extend" field-value
```

Now that we have sketched the intent and implementation of WebDAV, let's look more closely at the functions provided.

WebDAV Locking and Overwrite Prevention

By definition, collaboration requires more than one person working on a given document. The inherent problem associated with collaboration is illustrated in Figure 19-3.

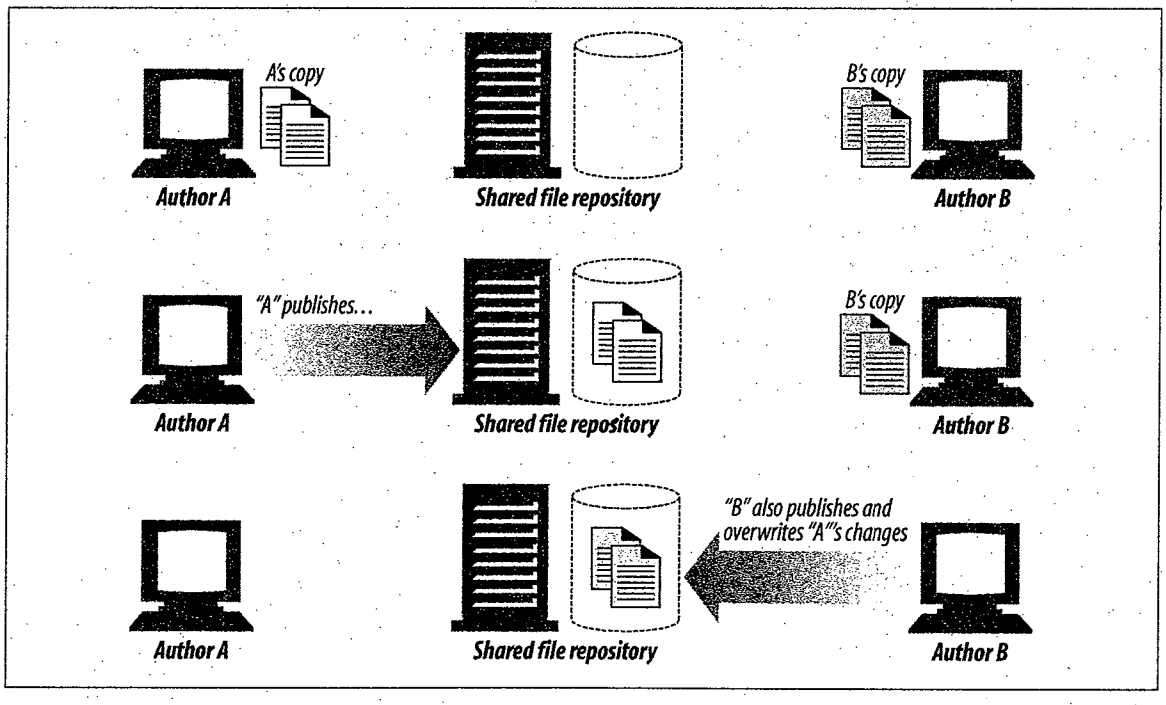


Figure 19-3. Lost update problem

In this example, authors A and B are jointly writing a specification. A and B independently make a set of changes to the document. A pushes the updated document to the repository, and at a later point, B posts her own version of the document into the repository. Unfortunately, because B never knew about A's changes, she never merged her version with A's version, resulting in A's work being lost.

To ameliorate the problem, WebDAV supports the concept of locking. Locking alone will not fully solve the problem. Versioning and messaging support are needed to complete the solution.

WebDAV supports two types of locks:

- Exclusive write locking of a resource or a collection
- Shared write locking of a resource or a collection

An *exclusive* write lock guarantees write privileges only to the lock owner. This type of locking completely eliminates potential conflicts. A *shared* write lock allows a group of people to work on a given document. This type of locking works well in an environment where all the authors are aware of each other's activities. WebDAV provides a property discovery mechanism, via PROPFIND, to determine the support for locking and the types of locks supported.

WebDAV has two new methods to support locking: LOCK and UNLOCK.

To accomplish locking, there needs to be a mechanism for identifying the author. WebDAV requires digest authentication (discussed in Chapter 13).

When a lock is granted, the server returns a token that is unique across the domain to the client. The specification refers to this as the opaquelocktoken lock token URI scheme. When the client subsequently wants to perform a write, it connects to the server and completes the digest authentication sequence. Once the authentication is complete, the WebDAV client presents the lock token, along with the PUT request. Thus, the combination of the correct user and the lock token is required to complete the write.

The LOCK Method

A powerful feature of WebDAV is its ability to lock multiple resources with a single LOCK request. WebDAV locking does not require the client to stay connected to the server.

For example, here's a simple LOCK request:

```
LOCK /ch-publish.fm HTTP/1.1
Host: minstar
Content-Type: text/xml
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Content-Length: 201

<?xml version="1.0"?>
<a:lockinfo xmlns:a="DAV:">
     <a:lockscope><a:exclusive/></a:lockscope>
     <a:locktype><a:write/></a:locktype>
     <a:owner><a:href>AuthorA</a:href></a:owner>
</a:lockinfo>
```

The XML being submitted has the <lockinfo> element as its base element. Within the <lockinfo> structure, there are three subelements:

<locktype>
: Indicates the type of lock. Currently there is only one, "write."

<lockscope>
: Indicates whether this is an exclusive lock or a shared lock.

<owner>
: Field is set with the person who holds the current lock.

Here's a successful response to our LOCK request:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 20:56:18 GMT
Content-Type: text/xml
Content-Length: 419

<?xml version="1.0"?>
<a:prop xmlns:a="DAV:">
<a:lockdiscovery><a:activelock>
<a:locktype><a:write/></a:locktype>
<a:lockscope><a:exclusive/></a:lockscope>
<a:owner xmlns:a="DAV:"><a:href>AutherA</a:href></a:owner>
<a:locktoken><a:href>opaquelocktoken:*****</a:href></a:locktoken>
<a:depth>0</a:depth>
<a:timeout>Second-180</a:timeout>
</a:activelock></a:lockdiscovery>
</a:prop>
```

The <lockdiscovery> element acts as a container for information about the lock. Embedded in the <lockdiscovery> element is an <activelock> subelement that holds the information sent with the request (<locktype>, <lockscope>, and <owner>). In addition, <activelock> has the following subelements:

<locktoken>
: Uniquely identifies the lock in a URI scheme called `opaquelocktoken`. Given the stateless nature of HTTP, this token is used to identify the ownership of the lock in future requests.

<depth>
: Mirrors the value of the Depth header.

<timeout>
: Indicates the timeout associated with the lock. In the above response (Figure 19-3), the timeout value is 180 seconds.

The opaquelocktoken scheme

The `opaquelocktoken` scheme is designed to provide a unique token across all resources for all times. To guarantee uniqueness, the WebDAV specification mandates the use of the universal unique identifier (UUID) mechanism, as described in ISO-11578.

When it comes to actual implementation, there is some leeway. The server has the choice of generating a UUID for each LOCK request, or generating a single UUID and maintaining the uniqueness by appending extra characters at the end. For performance considerations, the latter choice is better. However, if the server chooses to implement the latter choice, it is required to guarantee that none of the added extensions will ever be reused.

The <lockdiscovery> XML element

The <lockdiscovery> XML element provides a mechanism for active lock discovery. If others try to lock the file while a lock is in place, they will receive a <lockdiscovery> XML element that indicates the current owner. The <lockdiscovery> element lists all outstanding locks along with their properties.

Lock refreshes and the Timeout header

To refresh a lock, a client needs to resubmit a lock request with the lock token in the If header. The timeout value returned may be different from the earlier timeout values.

Instead of accepting the timeout value given by the server, a client may indicate the timeout value required in the LOCK request. This is done through the Timeout header. The syntax of the Timeout header allows the client to specify a few options in a comma-separated list. For example:

```
Timeout : Infinite, Second-86400
```

The server is not obligated to honor either of the options. However, it is required to provide the lock expiration time in the <timeout> XML element. In all cases, lock timeout is only a guideline and is not necessarily binding on the server. The administrator may do a manual reset, or some other extraordinary event may cause the server to reset the lock. The clients should avoid taking lengthy locks.

In spite of these primitives, we may not completely solve the "lost update problem" illustrated in Figure 19-3. To completely solve it, a cooperative event system with a versioning control is needed.

The UNLOCK Method

The UNLOCK method removes a lock on a resource, as follows:

```
UNLOCK /ch-publish.fm HTTP/1.1
Host: minstar.inktomi.com
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Lock-Token:
opaquelocktoken:*********

HTTP/1.1 204 OK
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 20:56:18 GMT
```

As with most resource management requests, WebDAV has two requirements for UNLOCK to succeed: prior completion of a successful digest authentication sequence, and matching the lock token that is sent in the Lock-Token header.

If the unlock is successful, a 204 No Content status code is returned to client. Table 19-1 summarizes the possible status codes with the LOCK and UNLOCK methods.

Table 19-1. Status codes for LOCK and UNLOCK methods

Status code	Defined by	Method	Effect
200 OK	HTTP	LOCK	Indicates successful locking.
201 Created	HTTP	LOCK	Indicates that a lock on a nonexistent resource succeeded by creating the resource.
204 No Content	HTTP	UNLOCK	Indicates successful unlocking.
207 Multi-Status	WebDAV	LOCK	The request was for locking multiple resources. Not all status codes returned were the same. Hence, they are all encapsulated in a 207 response.
403 Forbidden	HTTP	LOCK	Indicates that the client does not have permission to lock the resource.
412 Precondition Failed	HTTP	LOCK	Either the XML sent with the LOCK command indicated a condition to be satisfied and the server failed to complete the required condition, or the lock token could not be enforced.
422 Unprocessable Property	WebDAV	LOCK	Inapplicable semantics—an example may be specifying a nonzero Depth for a resource that is not a collection.
423 Locked	WebDAV	LOCK	Already locked.
424 Failed Dependency	WebDAV	UNLOCK	UNLOCK specifies other actions and their success as a condition for the unlocking. This error is returned if the dependency fails to complete.

Properties and META Data

Properties describe information about the resource, including the author's name, modification date, content rating, etc. META tags in HTML do provide a mechanism to embed this information as part of the content; however, many resources (such as any binary data) have no capability for embedding META data.

A distributed collaborative system such as WebDAV adds more complexity to the property requirement. For example, consider an author property: when a document gets edited, this property needs to be updated to reflect the new authors. WebDAV terms such dynamically modifiable properties "live" properties. The more permanent, static properties, such as Content-Type, are termed "dead" properties.

To support discovery and modification of properties, WebDAV extends HTTP to include two new methods, PROPFIND and PROPPATCH. Examples and corresponding XML elements are described in the following sections.

The PROPFIND Method

The PROPFIND (property find) method is used for retrieving the properties of a given file or a group of files (also known as a "collection"). PROPFIND supports three types of operations:

- Request all properties and their values.
- Request a selected set of properties and values.
- Request all property names.

Here's the scenario where all the properties and their values are requested:

```
PROPFIND /ch-publish.fm HTTP/1.1
Host: minstar.inktomi.com
User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT)
Depth: 0
Cache-Control: no-cache
Connection: Keep-Alive
Content-Length: 0
```

The <propfind> request element specifies the properties to be returned from a PROPFIND method. The following list summarizes a few XML elements that are used with PROPFIND requests:

<allprop>
: Requires all property names and values to be returned. To request all properties and their values, a WebDAV client may either send an <allprop> XML subelement as part of the <propfind> element, or submit a request with no body.

<propname>
: Specifies the set of property names to be returned.

<prop>
: A subelement of the <propfind> element. Specifies a specific property whose value is to be returned. For example: "<a:prop> <a:owner />..... </a:prop>".

Here's a response to a sample PROPFIND request:

```
HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
.....

<?xml version="1.0"?>
<a:multistatusxmlns:b="urn:uuid:********/" xmlns:c="xml:" xmlns:a="DAV:">
<a:response>
  <a:href>http://minstar/ch-publish.fm </a:href>
  <a:propstat>
    <a:status>HTTP/1.1 2000K</a:status>
    <a:prop>
      <a:getcontentlength b:dt="int">1155</a:getcontentlength>
      .....
      .....
```

```
        <a:ishidden b:dt="boolean">0</a:ishidden>
        <a:iscollection b:dt="boolean">0</a:iscollection>
      </a:prop>
    </a:propstat>
  </a:response></a:multistatus>
```

In this example, the server responds with a 207 Multi-Status code. WebDAV uses the 207 response for PROPFIND and a few other WebDAV methods that act simultaneously on multiple resources and potentially have different responses for each resource.

A few XML elements in the response need to be defined:

<multistatus>
: A container for multiple responses.

<href>
: Identifies the resource's URI.

<status>
: Contains the HTTP status code for the particular request.

<propstat>
: Groups one <status> element and one <prop> element. The <prop> element may contain one or more property name/value pairs for the given resource.

In the sample response listed above, the response is for one URI, *http://minstar/ch-publish.fm*. The <propstat> element embeds one <status> element and one <prop> element. For this URI, the server returned a 200 OK response, as defined by the <status> element. The <prop> element has several subelements; only some are listed in the example.

One instant application of PROPFIND is the support for directory listing. Given the expressability of a PROPFIND request, one single call can retrieve the entire hierarchy of the collection with all the properties of individual entities.

The PROPPATCH Method

The PROPPATCH method provides an atomic mechanism to set or remove multiple properties on a given resource. The atomicity will guarantee that either all of the requests are successful or none of them made it.

The base XML element for the PROPPATCH method is <propertyupdate>. It acts as a container for all the properties that need updating. The XML elements <set> and <remove> are used to specify the operation:

<set>
: Specifies the property values to be set. The <set> contains one or more <prop> subelements, which in turn contains the name/value pairs of the properties to be set for the resource. If the property already exists, the value is replaced.

<remove>

Specifies the properties that are to be removed. Unlike with <set>, only the names of the properties are listed in the <prop> container.

This trivial example sets and removes the "owner" property:

```
<d:propertyupdate xmlns:d="DAV:" xmlns:o="http://name-space/scheme/">
 <d:set>
   <d:prop>
     <o:owner>Author A</o:owner>
   </d:prop>
 </d:set>

 <d:remove>
   <d:prop>
     <o:owner/>
   </d:prop>
 </d:remove>
</d:propertyupdate>
```

The response to PROPPATCH requests is very similar to that for PROPFIND requests. For more information, refer to RFC 2518.

Table 19-2 summarizes the status codes for the PROPFIND and PROPPATCH methods.

Table 19-2. Status codes for PROPFIND and PROPPATCH methods

Status code	Defined by	Methods	Effect
200 OK	HTTP	PROPFIND, PROPPATCH	Command success.
207 Multi-Status	WEBDAV	PROPFIND, PROPPATCH	When acting on one or more resources (or a collection), the status for each object is encapsulated into one 207 response. This is a typical success response.
401 Unauthorized	HTTP	PROPATCH	Requires authorization to complete the property modification operation.
403 Forbidden	HTTP	PROPFIND, PROPPATCH	For PROPFIND, the client is not allowed to access the property. For PROPPATCH, the client may not change the property.
404 Not Found	HTTP	PROPFIND	No such property.
409 Conflict	HTTP	PROPPATCH	Conflict of update semantics—for example, trying to update a read-only property.
423 Locked	WebDAV	PROPPATCH	Destination resource is locked and there is no lock token or the lock token does not match.
507 Insufficient Storage	WebDAV	PROPPATCH	Not enough space for registering the modified property.

Collections and Namespace Management

A collection refers to a logical or physical grouping of resources in a predefined hierachy. A classic example of a collection is a directory. Like directories in a filesystem,

collections act as containers of other resources, including other collections (equivalent to directories on the filesystem).

WebDAV uses the XML namespace mechanism. Unlike traditional namespaces, XML namespace partitions allow for precise structural control while preventing any namespace collisions.

WebDAV provides five methods for manipulating the namespace: DELETE, MKCOL, COPY, MOVE, and PROPFIND. PROPFIND was discussed previously in this chapter, but let's talk about the other methods.

The MKCOL Method

The MKCOL method allows clients to create a collection at the indicated URL on the server. At first sight, it may seem rather redundant to define an entire new method just for creating a collection. Overlaying on top of a PUT or POST method seems like a perfect alternative. The designers of the WebDAV protocol did consider these alternatives and still chose to define a new method. Some of the reasons behind that decision are:

- To have a PUT or a POST create a collection, the client needs to send some extra "semantic glue" along with the request. While this certainly is feasible, defining an ad hoc protocol may become tedious and error-prone.
- Most of the access-control mechanisms are based on the type of methods—only a few are allowed to create and delete resources in the repository. If we overload other methods, these access-control mechanisms will not work.

For example, a request might be:

```
MKCOL /publishing HTTP/1.1
Host: minstar
Content-Length: 0
Connection: Keep-Alive
```

And the response might be:

```
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Fri, 10 May 2002 23:20:36 GMT
Location: http://minstar/publishing/
Content-Length: 0
```

Let us examine a few pathological cases:

- Suppose the collection already exists. If a MKCOL /colA request is made and colA already exists (i.e., namespace conflict), the request will fail with a 405 Method Not Allowed status code.
- If there are no write permissions, the MKCOL request will fail with a 403 Forbidden status code.

- If a request such as MKCOL /colA/colB is made and colA does not exist, the request will fail with a 409 Conflict status code.

Once the file or collection is created, we can delete it with the DELETE method.

The DELETE Method

We already saw the DELETE method in Chapter 3. WebDAV extends the semantics to cover collections.

If we need to delete a directory, the Depth header is needed. If the Depth header is not specified, the DELETE method assumes the Depth header to be set to infinity—that is, all the files in the directory and any subdirectories thereof are deleted. The response also has a Content-Location header identifying the collection that just got deleted. The request might read:

```
DELETE /publishing HTTP/1.0
Host: minstar
```

And the response might read:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 14 May 2002 16:41:44 GMT
Content-Location: http://minstar/publishing/
Content-Type: text/xml
Content-Length: 0
```

When removing collections, there always is a chance that a file in the collection is locked by someone else and can't be deleted. In such a case, the collection itself can't be deleted, and the server replies with a 207 Multi-Status status code. The request might read:

```
DELETE /publishing HTTP/1.0
Host: minstar
```

And the response might read:

```
HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
Content-Location: http://minstar/publishing/
.............
<?xml version="1.0"?>
<a:multistatus xmlns:a="DAV:">
<a:response>
<a:href>http://minstar/index3/ch-publish.fm</a:href>
<a:status> HTTP/1.1 423 Locked </a:status>
</a:response>
</a:multistatus>
```

In this transaction, the <status> XML element contains the status code 423 Locked, indicating that the resource *ch-publish.fm* is locked by another user.

The COPY and MOVE Methods

As with MKCOL, there are alternatives to defining new methods for COPY and MOVE operations. One such alternative for the COPY method is to do a GET request on the source, thus downloading the resource, and then to upload it back to the server with a PUT request. A similar scenario could be envisioned for MOVE (with the additional DELETE operation). However, this process does not scale well—consider all the issues involved in managing a COPY or MOVE operation on a multilevel collection.

Both the COPY and MOVE methods use the request URL as the source and the contents of the Destination HTTP header as the target. The MOVE method performs some additional work beyond that of the COPY method: it copies the source URL to the destination, checks the integrity of the newly created URI, and then deletes the source. The request might read:

```
{COPY,MOVE} /publishing HTTP/1.1
Destination: http://minstar/pub-new
Depth: infinity
Overwrite: T
Host: minstar
```

And the response might read:

```
HTTP/1.1 201 Created
Server: Microsoft-IIS/5.0
Date: Wed, 15 May 2002 18:29:53 GMT
Location: http://minstar.inktomi.com/pub-new/
Content-Type: text/xml
Content-Length: 0
```

When acting on a collection, the behavior of COPY or MOVE is affected by the Depth header. In the absence of the Depth header, infinity is assumed (i.e., by default, the entire structure of the source directory will be copied or moved). If the Depth is set to zero, the method is applied just to the resource. If we are doing a copy or a move of a collection, only a collection with properties identical to those of the source is created at the destination—no internal members of the collection are copied or moved.

For obvious reasons, only a Depth value of infinity is allowed with the MOVE method.

Overwrite header effect

The COPY and MOVE methods also may use the Overwrite header. The Overwrite header can be set to either `T` or `F`. If it's set to `T` and the destination exists, a DELETE with a Depth value of infinity is performed on the destination resource before a COPY or MOVE operation. If the Overwrite flag is set to `F` and the destination resource exists, the operation will fail.

COPY/MOVE of properties

When a collection or an element is copied, all of its properties are copied by default. However, a request may contain an optional XML body that supplies additional information for the operation. You can specify that all properties must be copied successfully for the operation to succeed, or define which properties must be copied for the operation to succeed.

A couple of pathological cases to consider are:

- Suppose COPY or MOVE is applied to the output of a CGI program or other script that generates content. To preserve the semantics, if a file generated by a CGI script is to be copied or moved, WebDAV provides "src" and "link" XML elements that point to the location of the program that generated the page.
- The COPY and MOVE methods may not be able to completely duplicate all of the live properties. For example, consider a CGI program. If it is copied away from the *cgi-bin* directory, it may no longer be executed. The current specification of WebDAV makes COPY and MOVE a "best effort" solution, copying all the static properties and the appropriate live properties.

Locked resources and COPY/MOVE

If a resource currently is locked, both COPY and MOVE are prohibited from moving or duplicating the lock at the destination. In both cases, if the destination is to be created under an existing collection with its own lock, the duplicated or moved resource is added to the lock. Consider the following example:

```
COPY /publishing HTTP/1.1
Destination: http://minstar/archived/publishing-old
```

Let's assume that */publishing* and */archived* already are under two different locks, lock1 and lock2. When the COPY operation completes, */publishing* continues to be under the scope of lock1, while, by virtue of moving into a collection that's already locked by lock2, *publishing-old* gets added to lock2. If the operation was a MOVE, just *publishing-old* gets added to lock2.

Table 19-3 lists most of the possible status codes for the MKCOL, DELETE, COPY, and MOVE methods.

Table 19-3. Status codes for the MKCOL, DELETE, COPY, and MOVE methods

Status code	Defined by	Methods	Effect
102 Processing	WebDAV	MOVE, COPY	If the request takes longer than 20 seconds, the server sends this status code to keep clients from timing out. This usually is seen with a COPY or MOVE of a large collection.
201 Created	HTTP	MKCOL, COPY, MOVE	For MKCOL, a collection has been created. For COPY and MOVE, a resource/collection was copied or moved successfully.

Table 19-3. Status codes for the MKCOL, DELETE, COPY, and MOVE methods (continued)

Status code	Defined by	Methods	Effect
204 No Content	HTTP	DELETE, COPY, MOVE	For DELETE, a standard success response. For COPY and MOVE, the resource was copied over successfully or moved to replace an existing entity.
207 Multi-Status	WebDAV	MKCOL, COPY, MOVE	For MKCOL, a typical success response. For COPY and MOVE, if an error is associated with a resource other than the request URI, the server returns a 207 response with the XML body detailing the error.
403 Forbidden	HTTP	MKCOL, COPY, MOVE	For MKCOL, the server does not allow creation of a collection at the specified location. For COPY and MOVE, the source and destination are the same.
409 Conflict	HTTP	MKCOL, COPY, MOVE	In all cases, the methods are trying to create a collection or a resource when an intermediate collection does not exist—for example, trying to create colA/colB when colA does not exist.
412 Precondition Failed	HTTP	COPY, MOVE	Either the Overwrite header is set to F and the destination exists, or the XML body specifies a certain requirement (such as keeping the "liveness" property) and the COPY or MOVE methods are not able to retain the property.
415 Unsupported Media Type	HTTP	MKCOL	The server does not support or understand the creation of the request entity type.
422 Unprocessable Entity	WebDAV	MKCOL	The server does not understand the XML body sent with the request.
423 Locked	WebDAV	DELETE, COPY, MOVE	The source or the destination resource is locked, or the lock token supplied with the method does not match.
502 Bad Gateway	HTTP	COPY, MOVE	The destination is on a different server and permissions are missing.
507 Insufficient Storage	WebDAV	MKCOL COPY	There is not enough free space to create the resource.

Enhanced HTTP/1.1 Methods

WebDAV modifies the semantics of the HTTP methods DELETE, PUT, and OPTIONS. Semantics for the GET and HEAD methods remain unchanged. Operations performed by POST always are defined by the specific server implementation, and WebDAV does not modify any of the POST semantics. We already covered the DELETE method, in "Collections and Namespace Management." We'll discuss the PUT and OPTIONS methods here.

The PUT method

Though PUT is not defined by WebDAV, it is the only way for an author to transport the content to a shared site. We discussed the general functionality of PUT in Chapter 3. WebDAV modifies its behavior to support locking.

Consider the following example:

```
PUT /ch-publish.fm HTTP/1.1
Accept: */*
If:<http://minstar/index.htm>(<opaquelocktoken:********>)
User-Agent: DAV Client (C)
Host: minstar.inktomi.com
Connection: Keep-Alive
Cache-Control: no-cache
Content-Length: 1155
```

To support locking, WebDAV adds an If header to the PUT request. In the above transaction, the semantics of the If header state that if the lock token specified with the If header matches the lock on the resource (in this case, *ch-publish.fm*), the PUT operation should be performed. The If header also is used with a few other methods, such as PROPPATCH, DELETE, MOVE, LOCK, UNLOCK, etc.

The OPTIONS method

We discussed OPTIONS in Chapter 3. This usually is the first request a WebDAV-enabled client makes. Using the OPTIONS method, the client tries to establish the capability of the WebDAV server. Consider a transaction in which the request reads:

```
OPTIONS /ch-publish.fm HTTP/1.1
Accept: */*
Host: minstar.inktomi.com
```

And the response reads:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
MS-Author-Via: DAV
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL,PROPFIND,
PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, COPY, MOVE, PROPFIND,PROPPATCH,
SEARCH, LOCK, UNLOCK
```

There are several interesting headers in the response to the OPTIONS method. A slightly out-of-order examination follows:

- The DAV header carries the information about DAV compliance classes. There are two classes of compliance:

  Class 1 compliance
  : Requires the server to comply with all MUST requirements in all sections of RFC 2518. If the resource complies only at the Class 1 level, it will send 1 with the DAV header.

  Class 2 compliance
  : Meets all the Class 1 requirements and adds support for the LOCK method. Along with LOCK, Class 2 compliance requires support for the Timeout and

Lock-Token headers and the <supportedlock> and <lockdiscovery> XML elements. A value of 2 in the DAV header indicates Class 2 compliance.

In the above example, the DAV header indicates both Class 1 and Class 2 compliance.

- The Public header lists all methods supported by this particular server.
- The Allow header usually contains a subset of the Public header methods. It lists only those methods that are allowed on this particular resource (*ch-publish.fm*).
- The DASL header provides the type of query grammar used in the SEARCH method. In this case, it is sql. More details about the DASL header are provided at *http://www.webdav.org*.

Version Management in WebDAV

It may be ironic, given the "V" in "DAV," but versioning is a feature that did not make the first cut. In a multi-author, collaborative environment, version management is critical. In fact, to completely fix the lost update problem (illustrated in Figure 19-3), locking and versioning are essential. Some of the common features associated with versioning are the ability to store and access previous document versions and the ability to manage the change history and any associated annotations detailing the changes.

Versioning was added to WebDAV in RFC 3253.

Future of WebDAV

WebDAV is well supported today. Working implementations of clients include IE 5.x and above, Windows Explorer, and Microsoft Office. On the server side, implementations include IIS5.x and above, Apache with mod_dav, and many others. Both Windows XP and Mac OS 10.x provide support for WebDAV out of the box; thus, any applications written to run on these operating systems are WebDAV-enabled natively.

For More Information

For more information, refer to:

http://officeupdate.microsoft.com/frontpage/wpp/serk/

Microsoft FrontPage 2000 Server Extensions Resource Kit.

http://www.ietf.org/rfc/rfc2518.txt?number=2518

"HTTP Extensions for Distributed Authoring—WEBDAV," by Y. Goland, J. Whitehead, A. Faizi, S. Carter, and D. Jensen.

http://www.ietf.org/rfc/rfc3253.txt?number=3253

"Versioning Extensions to WebDAV," by G. Clemm, J. Amsden, T. Ellison, C. Kaler, and J. Whitehead.

http://www.ics.uci.edu/pub/ietf/webdav/intro/webdav_intro.pdf

"WEBDAV: IETF Standard for Collaborative Authoring on the Web," by J. Whitehead and M. Wiggins.

http://www.ics.uci.edu/~ejw/http-future/whitehead/http_pos_paper.html

"Lessons from WebDAV for the Next Generation Web Infrastructure," by J. Whitehead.

http://www.microsoft.com/msj/0699/dav/davtop.htm

"Distributed Authoring and Versioning Extensions for HTTP Enable Team Authoring," by L. Braginski and M. Powell.

http://www.webdav.org/dasl/protocol/draft-dasl-protocol-00.html

"DAV Searching & Locating," by S. Reddy, D. Lowry, S. Reddy, R. Henderson, J. Davis, and A. Babich.

CHAPTER 20

Redirection and Load Balancing

HTTP does not walk the Web alone. The data in an HTTP message is governed by many protocols on its journey. HTTP cares only about the endpoints of the journey—the sender and the receiver—but in a world with mirrored servers, web proxies, and caches, the destination of an HTTP message is not necessarily straightforward.

This chapter is about redirection technologies—network tools, techniques, and protocols that determine the final destination of an HTTP message. Redirection technologies usually determine whether the message ends up at a proxy, a cache, or a particular web server in a server farm. Redirection technologies may send your messages to places a client didn't explicitly request.

In this chapter, we'll take a look at the following redirection techniques, how they work, and what their load-balancing capabilities are (if any):

- HTTP redirection
- DNS redirection
- Anycast routing
- Policy routing
- IP MAC forwarding
- IP address forwarding
- The Web Cache Coordination Protocol (WCCP)
- The Intercache Communication Protocol (ICP)
- The Hyper Text Caching Protocol (HTCP)
- The Network Element Control Protocol (NECP)
- The Cache Array Routing Protocol (CARP)
- The Web Proxy Autodiscovery Protocol (WPAD)

Why Redirect?

Redirection is a fact of life in the modern Web because HTTP applications always want to do three things:

- Perform HTTP transactions reliably
- Minimize delay
- Conserve network bandwidth

For these reasons, web content often is distributed in multiple locations. This is done for reliability, so that if one location fails, another is available; it is done to lower response times, because if clients can access a nearer resource, they receive their requested content faster; and it's done to lower network congestion, by spreading out target servers. You can think of redirection as a set of techniques that help to find the "best" distributed content.

The subject of load balancing is included because redirection and load balancing coexist. Most redirection deployments include some form of load balancing; that is, they are capable of spreading incoming message load among a set of servers. Conversely, any form of load balancing involves redirection, because incoming messages must somehow be somehow among the servers sharing the load.

Where to Redirect

Servers, proxies, caches, and gateways all appear to clients as servers, in the sense that a client sends them an HTTP request, and they process it. Many redirection techniques work for servers, proxies, caches, and gateways because of their common, server-like traits. Other redirection techniques are specially designed for a particular class of endpoint and are not generally applicable. We'll see general techniques and specialized techniques in later sections of this chapter.

Web servers handle requests on a per-IP basis. Distributing requests to duplicate servers means that each request for a specific URL should be sent to an optimal web server (the one nearest to the client, or the least-loaded one, or some other optimization). Redirecting to a server is like sending all drivers in search of gasoline to the nearest gas station.

Proxies tend to handle requests on a per-protocol basis. Ideally, all HTTP traffic in the neighborhood of a proxy should go through the proxy. For instance, if a proxy cache is near various clients, all requests ideally will flow through the proxy cache, because the cache will store popular documents and serve them directly, avoiding longer and more expensive trips to the origin servers. Redirecting to a proxy is like siphoning off traffic on a main access road (no matter where it is headed) to a local shortcut.

Overview of Redirection Protocols

The goal of redirection is to send HTTP messages to available web servers as quickly as possible. The direction that an HTTP message takes on its way through the Internet is affected by the HTTP applications and routing devices it passes from, through, and toward. For example:

- The browser application that creates the client's message could be configured to send it to a proxy server.
- DNS resolvers choose the IP address that is used for addressing the message. This IP address can be different for different clients in different geographical locations.
- As the message passes through networks, it is divided into addressed packets; switches and routers examine the TCP/IP addressing on the packets and make decisions about routing the packets on that basis.
- Web servers can bounce requests back to different web servers with HTTP redirects.

Browser configuration, DNS, TCP/IP routing, and HTTP all provide mechanisms for redirecting messages. Notice that some methods, such as browser configuration, make sense only for redirecting traffic to proxies, while others, such as DNS redirection, can be used to send traffic to any server.

Table 20-1 summarizes the redirection methods used to redirect messages to servers, each of which is discussed later in this chapter.

Table 20-1. General redirection methods

Mechanism	How it works	Basis for rerouting	Limitations
HTTP redirection	Initial HTTP request goes to a first web server that chooses a "best" web server to serve the content. The first web server sends the client an HTTP redirect to the chosen server. The client resends the request to the chosen server.	Many options, from round-robin load balancing, to minimizing latency, to choosing the shortest path.	Can be slow—every transaction involves the extra redirect step. Also, the first server must be able to handle the request load.
DNS redirection	DNS server decides which IP address, among several, to return for the hostname in the URL.	Many options, from round-robin load balancing, to minimizing latency, to choosing the shortest path.	Need to configure DNS server.
Anycast addressing	Several servers use the same IP address. Each server masquerades as a backbone router. The other routers send packets addressed to the shared IP to the nearest server (believing they are sending packets to the nearest router).	Routers use built-in shortest-path routing capabilities.	Need to own/configure routers. Risks address conflicts. Established TCP connections can break if routing changes and packets associated with a connection get sent to different servers.

Table 20-1. General redirection methods (continued)

Mechanism	How it works	Basis for rerouting	Limitations
IP MAC forwarding	A network element such as a switch or router reads a packet's destination address; if the packet should be redirected, the switch gives the packet the destination MAC address of a server or proxy.	Save bandwidth and improve QOS. Load balance.	Server or proxy must be one hop away.
IP address forwarding	Layer-4 switch evaluates a packet's destination port and changes the IP address of a redirect packet to that of a proxy or mirrored server.	Save bandwidth and improve QOS. Load balance.	IP address of the client can be lost to the server/proxy.

Table 20-2 summarizes the redirection methods used to redirect messages to proxy servers.

Table 20-2. Proxy and cache redirection techniques

Mechanism	How it works	Basis for rerouting	Limitations
Explicit browser configuration	Web browser is configured to send HTTP messages to a nearby proxy, usually a cache. The configuration can be done by the end user or by a service that manages the browser.	Save bandwidth and improve QOS. Load balance.	Depends on ability to configure the browser.
Proxy auto-configuration (PAC)	Web browser retrieves a PAC file from a configuration server. The PAC file tells the browser what proxy to use for each URL.	Save bandwidth and improve QOS. Load balance.	Browser must be configured to query the configuration server.
Web Proxy Autodiscovery Protocol (WPAD)	Web browser asks a configuration server for the URL of a PAC file. Unlike PAC alone, the browser does not have to be configured with a specific configuration server.	The configuration server bases the URL on information in client HTTP request headers. Load balance.	Only a few browsers support WPAD.
Web Cache Coordination Protocol (WCCP)	Router evaluates a packet's destination address and encapsulates redirect packets with the IP address of a proxy or mirrored server. Works with many existing routers. Packet can be encapsulated, so the client's IP address is not lost.	Save bandwidth and improve QOS. Load balance.	Must use routers that support WCCP. Some topological limitations.
Internet Cache Protocol (ICP)	A proxy cache can query a group of sibling caches for requested content. Also supports cache hierarchies.	Obtaining content from a sibling or parent cache is faster than applying to the origin server.	False cache hits can arise because only the URL is used to request content.
Cache Array Routing Protocol (CARP)	A proxy cache hashing protocol. Allows a cache to forward a request to a parent cache. Unlike with ICP, the content on the caches is disjoint, and the group of caches acts as a single large cache.	Obtaining content from a nearby peer cache is faster than applying to the origin server.	CARP cannot support sibling relationships. All CARP clients must agree on the configuration; otherwise, different clients will send the same URI to different parents, reducing hit ratios.

Table 20-2. Proxy and cache redirection techniques (continued)

Mechanism	How it works	Basis for rerouting	Limitations
Hyper Text Caching Protocol (HTCP)	Participating proxy caches can query a group of sibling caches for requested content. Supports HTTP 1.0 and 1.1 headers to fine-tune cache queries.	Obtaining content from a sibling or parent cache is faster than applying to the origin server.	

General Redirection Methods

In this section, we will delve deeper into the various redirection methods that are commonly used for both servers and proxies. These techniques can be used to redirect traffic to a different (presumably more optimal) server or to vector traffic through a proxy. Specifically, we'll cover HTTP redirection, DNS redirection, anycast addressing, IP MAC forwarding, and IP address forwarding.

HTTP Redirection

Web servers can send short redirect messages back to clients, telling them to try someplace else. Some web sites use HTTP redirection as a simple form of load balancing; the server that handles the redirect (the redirecting server) finds the least-loaded content server available and redirects the browser to that server. For widely distributed web sites, determining the "best" available server gets more complicated, taking into account not only the servers' load but the Internet distance between the browser and the server. One advantage of HTTP redirection over some other forms of redirection is that the redirecting server knows the client's IP address; in theory, it may be able to make a more informed choice.

Here's how HTTP redirection works. In Figure 20-1a, Alice sends a request to *www.joes-hardware.com*:

```
GET /hammers.html HTTP/1.0
Host: www.joes-hardware.com
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

In Figure 20-1b, instead of sending back a web page body with HTTP status code 200, the server sends back a redirect message with status code 302:

```
HTTP/1.0 302 Redirect
Server: Stronghold/2.4.2 Apache/1.3.6
Location: http://161.58.228.45/hammers.html
```

Now, in Figure 20-1c, the browser resends the request using the redirected URL, this time to host 161.58.228.45:

```
GET /hammers.html HTTP/1.0
Host: 161.58.228.45
User-Agent: Mozilla/4.51 [en] (X11; U; IRIX 6.2 IP22)
```

Another client could get redirected to a different server. In Figure 20-1d–f, Bob's request gets redirected to 161.58.228.46.

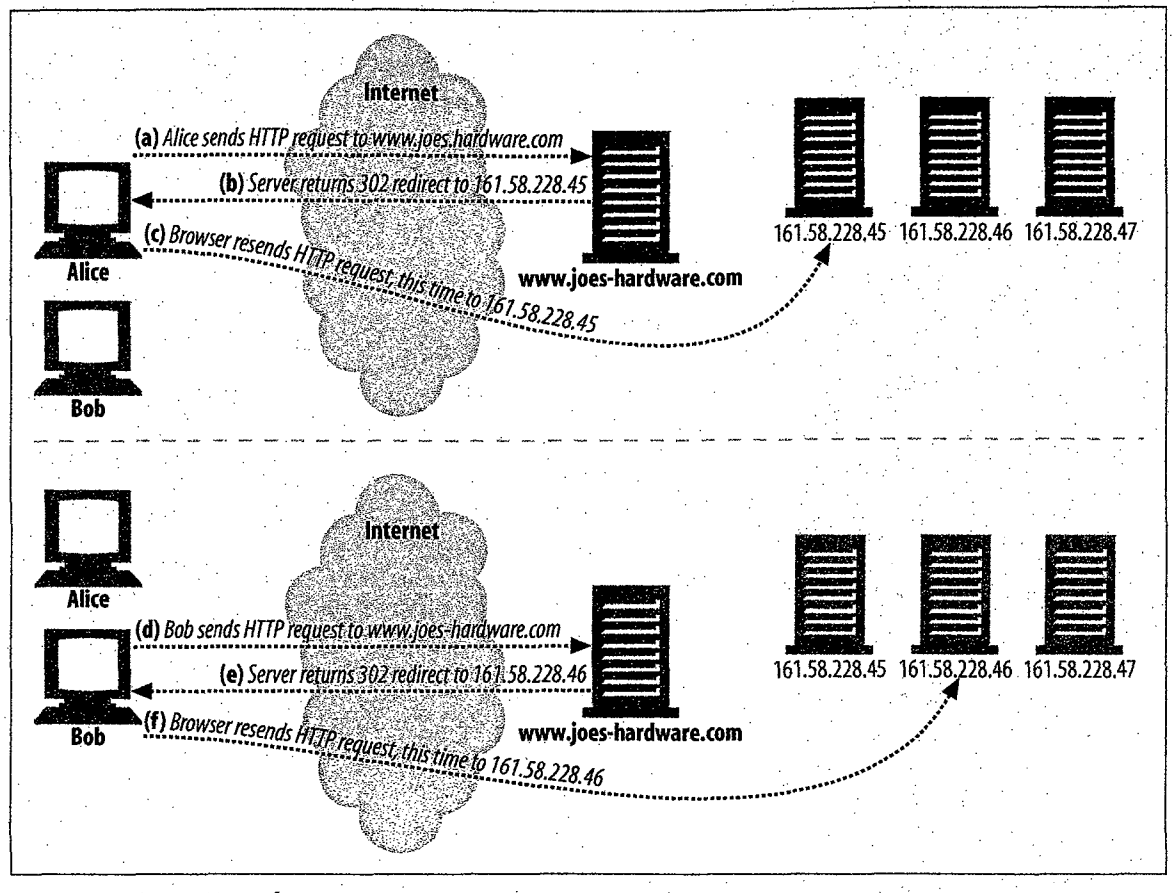


Figure 20-1. HTTP redirection

HTTP redirection can vector requests across servers, but it has several disadvantages:

- A significant amount of processing power is required from the original server to determine which server to redirect to. Sometimes almost as much server horsepower is required to issue the redirect as would be to serve up the page itself.
- User delays are increased, because two round trips are required to access pages.
- If the redirecting server is broken, the site will be broken.

Because of these weaknesses, HTTP redirection usually is used in combination with some of the other redirection techniques.

DNS Redirection

Every time a client tries to access Joe's Hardware's web site, the domain name *www.joes-hardware.com* must be resolved to an IP address. The DNS resolver may be the client's own operating system, a DNS server in the client's network, or a more remote DNS server. DNS allows several IP addresses to be associated to a single domain, and DNS resolvers can be configured or programmed to return varying IP addresses. The basis on which the resolver returns the IP address can run from the simple (round robin) to the complex (such as checking the load on several servers and returning the IP address of the least-loaded server).

In Figure 20-2, Joe runs four servers for *www.joes-hardware.com*. The DNS server has to decide which of four IP addresses to return for *www.joes-hardware.com*. The easiest DNS decision algorithm is a simple round robin.

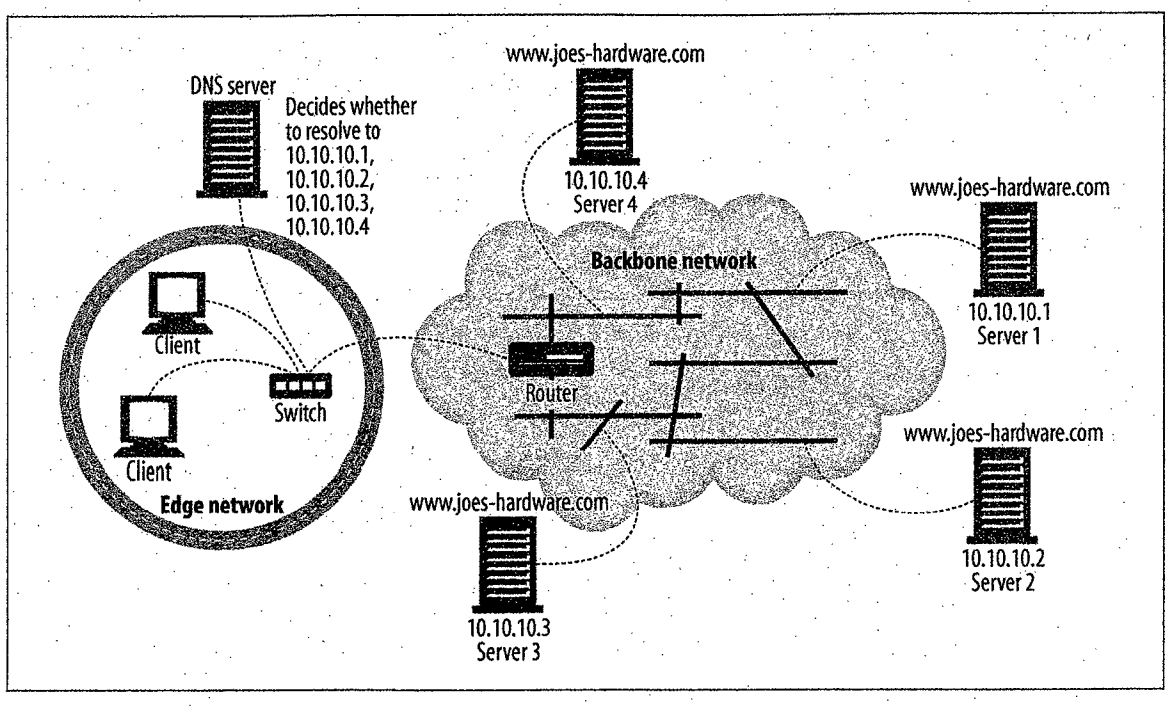


Figure 20-2. DNS-based redirection

For a run-through of the DNS resolution process, see the DNS reference listed at the end of this chapter.

DNS round robin

One of the most common redirection techniques also is one of the simplest. DNS round robin uses a feature of DNS hostname resolution to balance load across a farm of web servers. It is a pure load-balancing strategy, and it does not take into account any factors about the location of the client relative to the server or the current stress on the server.

Let's look at what CNN.com really does. In early May of 2000, we used the *nslookup* Unix tool to find the IP addresses associated with CNN.com. Example 20-1 shows the results.*

Example 20-1. IP addresses for www.cnn.com

```
% nslookup www.cnn.com
Name:    cnn.com
```

* DNS results as of May 7, 2000 and resolved from Northern California. The particular values likely will change over time, and some DNS systems return different values based on client location.

Example 20-1. IP addresses for www.cnn.com (continued)

```
Addresses:  207.25.71.5, 207.25.71.6, 207.25.71.7, 207.25.71.8
            207.25.71.9, 207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23
            207.25.71.24, 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
            207.25.71.29, 207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245
            207.25.71.246
Aliases:  www.cnn.com
```

The web site *www.cnn.com* actually is a farm of 20 distinct IP addresses! Each IP address might typically translate to a different physical server.

Multiple addresses and round-robin address rotation

Most DNS clients just use the first address of the multi-address set. To balance load, most DNS servers rotate the addresses each time a lookup is done. This address rotation often is called *DNS round robin*.

For example, three consecutive DNS lookups of *www.cnn.com* might return rotated lists of IP addresses like those shown in Example 20-2.

Example 20-2. Rotating DNS address lists

```
% nslookup www.cnn.com
Name:    cnn.com
Addresses:  207.25.71.5, 207.25.71.6, 207.25.71.7, 207.25.71.8
            207.25.71.9, 207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23
            207.25.71.24, 207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28
            207.25.71.29, 207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245
            207.25.71.246

% nslookup www.cnn.com
Name:    cnn.com
Addresses:  207.25.71.6, 207.25.71.7, 207.25.71.8, 207.25.71.9
            207.25.71.12, 207.25.71.20, 207.25.71.22, 207.25.71.23, 207.25.71.24
            207.25.71.25, 207.25.71.26, 207.25.71.27, 207.25.71.28, 207.25.71.29
            207.25.71.30, 207.25.71.82, 207.25.71.199, 207.25.71.245, 207.25.71.246
            207.25.71.5

% nslookup www.cnn.com
Name:    cnn.com
Addresses:  207.25.71.7, 207.25.71.8, 207.25.71.9, 207.25.71.12
            207.25.71.20, 207.25.71.22, 207.25.71.23, 207.25.71.24, 207.25.71.25
            207.25.71.26, 207.25.71.27, 207.25.71.28, 207.25.71.29, 207.25.71.30
            207.25.71.82, 207.25.71.199, 207.25.71.245, 207.25.71.246, 207.25.71.5
            207.25.71.6
```

In Example 20-2:

- The first address of the first DNS lookup is 207.25.71.5.
- The first address of the second DNS lookup is 207.25.71.6.
- The first address of the third DNS lookup is 207.25.71.7.

DNS round robin for load balancing

Because most DNS clients just use the first address, the DNS rotation serves to balance load among servers. If DNS did not rotate the addresses, most clients would always send load to the first client.

Figure 20-3 shows how DNS round-robin rotation acts to balance load:

- When Alice tries to connect to *www.cnn.com*, she looks up the IP address using DNS and gets back 207.25.71.5 as the first IP address. Alice connects to the web server 207.25.71.5 in Figure 20-3c.
- When Bob subsequently tries to connect to *www.cnn.com*, he also looks up the IP address using DNS, but he gets back a different result because the address list has been rotated one position, based on Alice's previous request. Bob gets back 207.25.71.6 as the first IP address, and he connects to this server in Figure 20-3f.

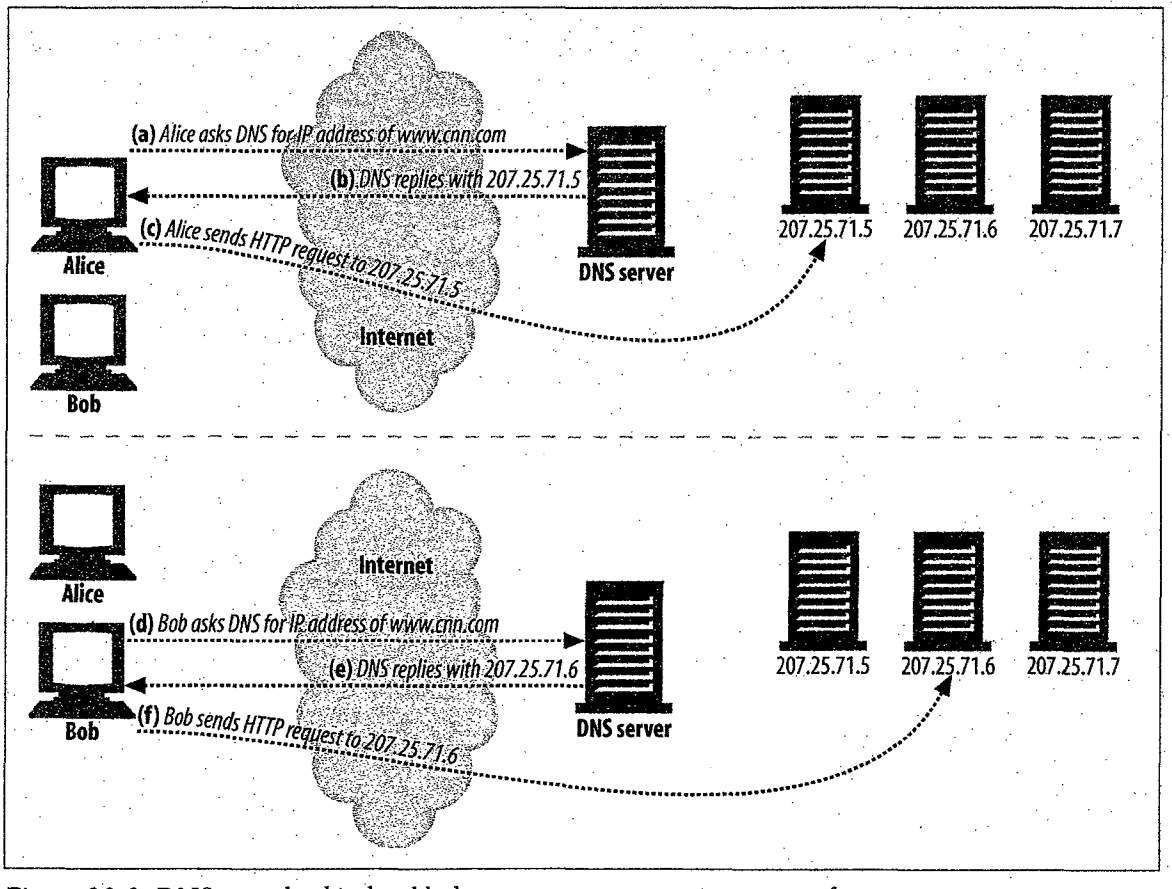


Figure 20-3. DNS round robin load balances across servers in a server farm

The impact of DNS caching

DNS address rotation spreads the load around, because each DNS lookup to a server gets a different ordering of server addresses. However, this load balancing isn't perfect, because the results of the DNS lookup may be memorized and reused by applications, operating systems, and some primitive child DNS servers. Many web browsers

perform a DNS lookup for a host but then use the same address over and over again, to eliminate the cost of DNS lookups and because some servers prefer to keep talking to the same client. Furthermore, many operating systems perform the DNS lookup automatically, and cache the result, but don't rotate the addresses. Consequently, DNS round robin generally doesn't balance the load of a single client—one client typically will be stuck to one server for a long period of time.

But, even though DNS doesn't deal out the transactions of a single client across server replicas, it does a decent job of spreading the aggregate load of multiple clients. As long as there is a modestly large number of clients with similar demand, the load will be relatively well distributed across servers.

Other DNS-based redirection algorithms

We've already discussed how DNS rotates address lists with each request. However, some enhanced DNS servers use other techniques for choosing the order of the addresses:

Load-balancing algorithms
: Some DNS servers keep track of the load on the web servers and place the least-loaded web servers at the front of the list.

Proximity-routing algorithms
: DNS servers can attempt to direct users to nearby web servers, when the farm of web servers is geographically dispersed.

Fault-masking algorithms
: DNS servers can monitor the health of the network and route requests away from service interruptions or other faults.

Typically, the DNS server that runs sophisticated server-tracking algorithms is an authoritative server that is under the control of the content provider (see Figure 20-4).

Several distributed hosting services use this DNS redirection model. One drawback of the model for services that look for nearby servers is that the only information that the authoritative DNS server uses to make its decision is the IP address of the local DNS server, not the IP address of the client.

Anycast Addressing

In anycast addressing, several geographically dispersed web servers have the exact same IP address and rely on the "shortest-path" routing capabilities of backbone routers to send client requests to the server nearest to the client. One way this method can work is for each web server to advertise itself as a router to a neighboring backbone router. The web server talks to its neighboring backbone router using a router communication protocol. When the backbone router receives packets aimed at the anycast address, it looks (as it usually would) for the nearest "router" that

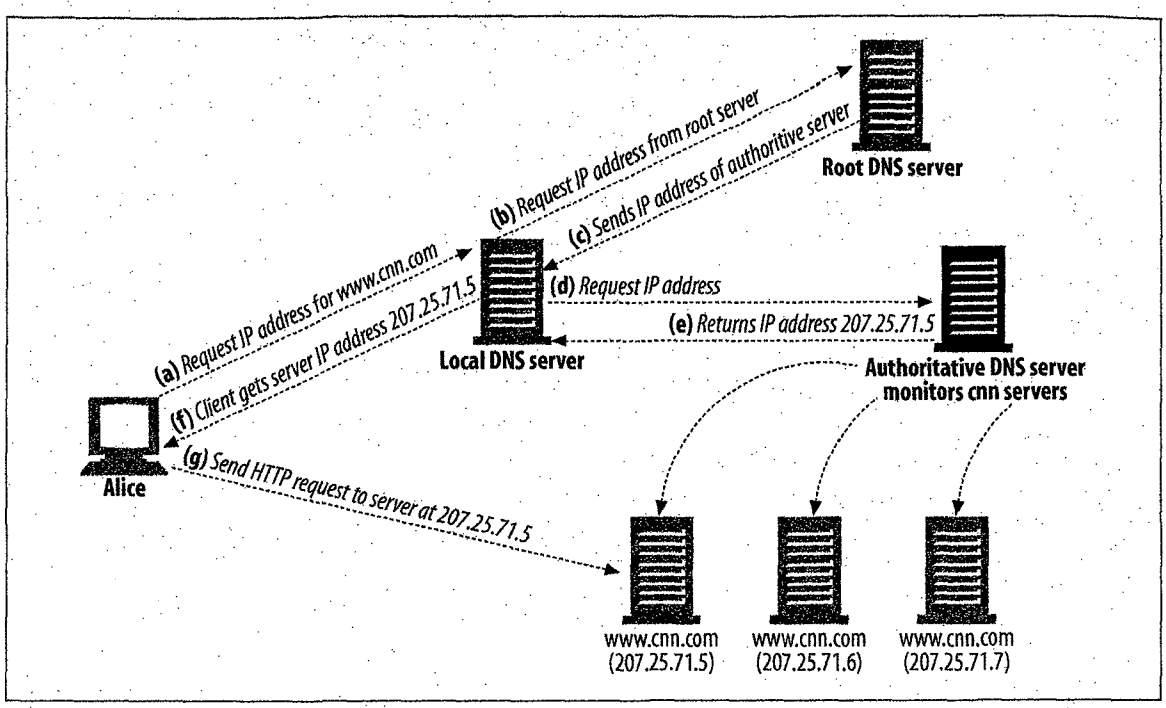


Figure 20-4. DNS request involving authoritative server

accepts that IP address. Because the server will have advertised itself as a router for that address, the backbone router will send the server the packet.

In Figure 20-5, three servers front the same IP address, 10.10.10.1. The Los Angeles (LA) server advertises this address to the LA router, the New York (NY) server advertises the same address to the NY router, and so on. The servers communicate with the routers using a router protocol. The routers automatically route client requests aimed at 10.10.10.1 to the nearest server that advertises the address. In Figure 20-5, a request for the IP address 10.10.10.1 will be routed to server 3.

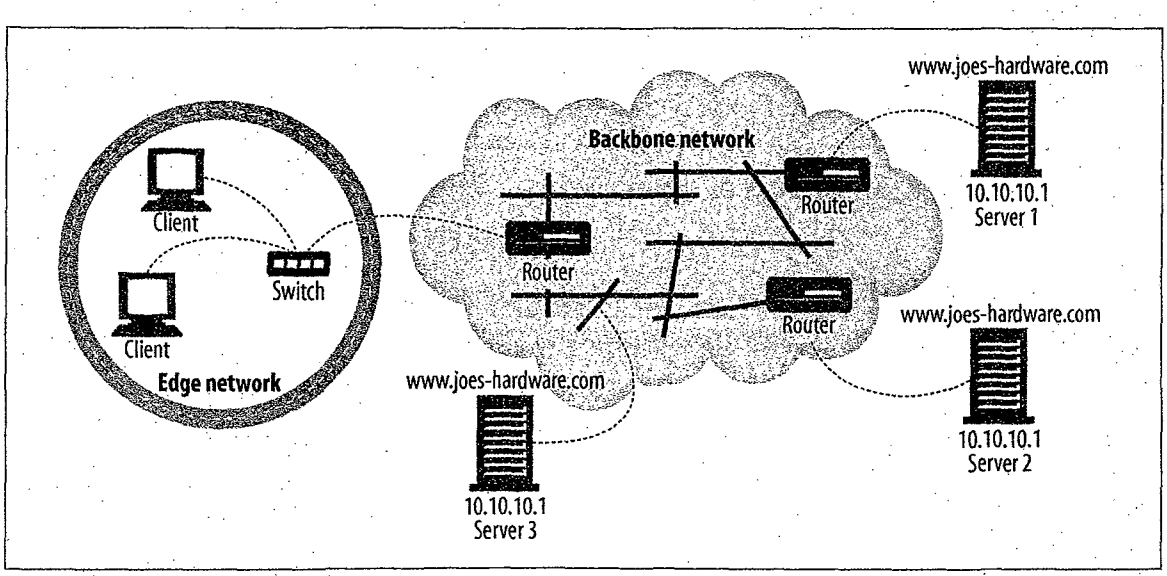


Figure 20-5. Distributed anycast addressing

Anycast addressing is still an experimental technique. For distributed anycast to work, the servers must "speak router language" and the routers must be able to handle possible address conflicts, because Internet addressing basically assumes one server for one address. (If done improperly, this can lead to serious problems known as "route leaks.") Distributed anycast is an emerging technology and might be a solution for content providers who control their own backbone networks.

IP MAC Forwarding

In Ethernet networks, HTTP messages are sent in the form of addressed data packets. Each packet has a layer-4 address, consisting of the source and destination IP address and TCP port numbers; this is the address to which layer 4–aware devices pay attention. Each packet also has a layer-2 address, the Media Access Control (MAC) address, to which layer-2 devices (commonly switches and hubs) pay attention. The job of layer-2 devices is to receive packets with particular incoming MAC addresses and forward them to particular outgoing MAC addresses.

In Figure 20-6, for example, the switch is programmed to send all traffic from MAC address "MAC3" to MAC address "MAC4."

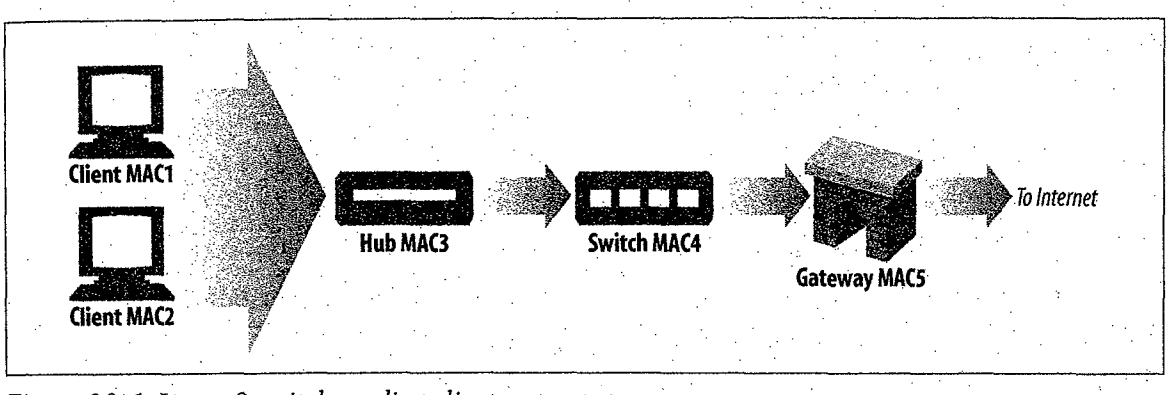


Figure 20-6. Layer-2 switch sending client requests to a gateway

A layer 4–aware switch is able to examine the layer-4 addressing (IP addresses and TCP port numbers) and make routing decisions based on this information. For example, a layer-4 switch could send all port 80–destined web traffic to a proxy. In Figure 20-7, the switch is programmed to send all port 80 traffic from MAC3 to MAC6 (a proxy cache). All other MAC3 traffic goes to MAC5.

Typically, if the requested HTTP content is in the cache and is fresh, the proxy cache serves it; otherwise, the proxy cache sends an HTTP request to the origin server for the content, on the client's behalf. The switch sends port 80 requests from the proxy (MAC6) to the Internet gateway (MAC5).

Layer-4 switches that support MAC forwarding usually can forward requests to several proxy caches and balance the load among them. Likewise, HTTP traffic also can be forwarded to alternate HTTP servers.

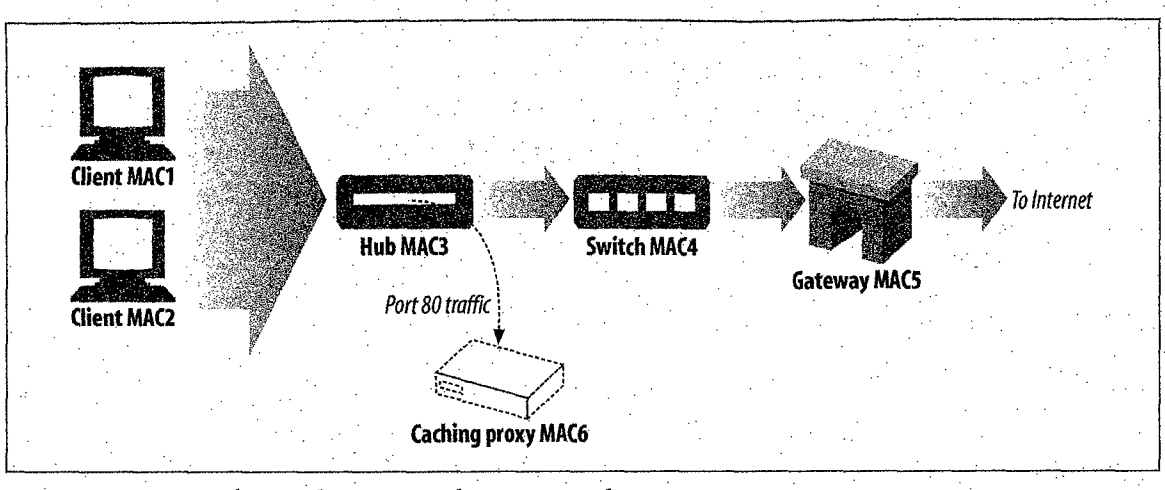


Figure 20-7. MAC forwarding using a layer-4 switch

Because MAC address forwarding is point-to-point only, the server or proxy has to be located one hop away from the switch.

IP Address Forwarding

In IP address forwarding, a switch or other layer 4–aware device examines TCP/IP addressing on incoming packets and routes packets accordingly by changing the destination IP address, instead of the destination MAC address. An advantage over MAC forwarding is that the destination server need not be one hop away; it just needs to be located upstream from the switch, and the usual layer-3 end-to-end Internet routing gets the packet to the right place. This type of forwarding also is called Network Address Translation (NAT).

There is a catch, however: routing symmetry. The switch that accepts the incoming TCP connection from the client is managing that connection; the switch must send the response back to the client on that TCP connection. Therefore, any response from the destination server or proxy must return to the switch (see Figure 20-8).

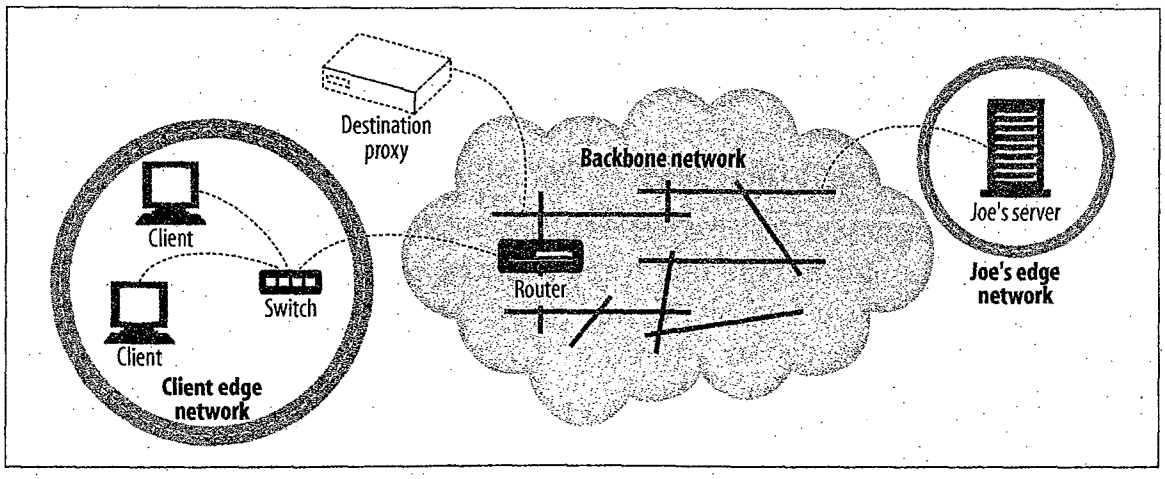


Figure 20-8. A switch doing IP forwarding to a caching proxy or mirrored web server

Two ways to control the return path of the response are:

- Change the source IP address of the packet to the IP address of the switch. That way, regardless of the network configuration between the switch and server, the response packet goes to the switch. This is called *full NAT*, where the IP forwarding device translates both destination and source IP addresses. Figure 20-9 shows the effect of full NAT on a TCP/IP datagram. The consequence is that the client IP address is unknown to the web server, which might want it for authentication or billing purposes, for example.
- If the source IP address remains the client's IP address, make sure (from a hardware perspective) that no routes exist directly from server to client (bypassing the switch). This sometimes is called *half NAT*. The advantage here is that the server obtains the client IP address, but the disadvantage is the requirement of some control of the entire network between client and server.

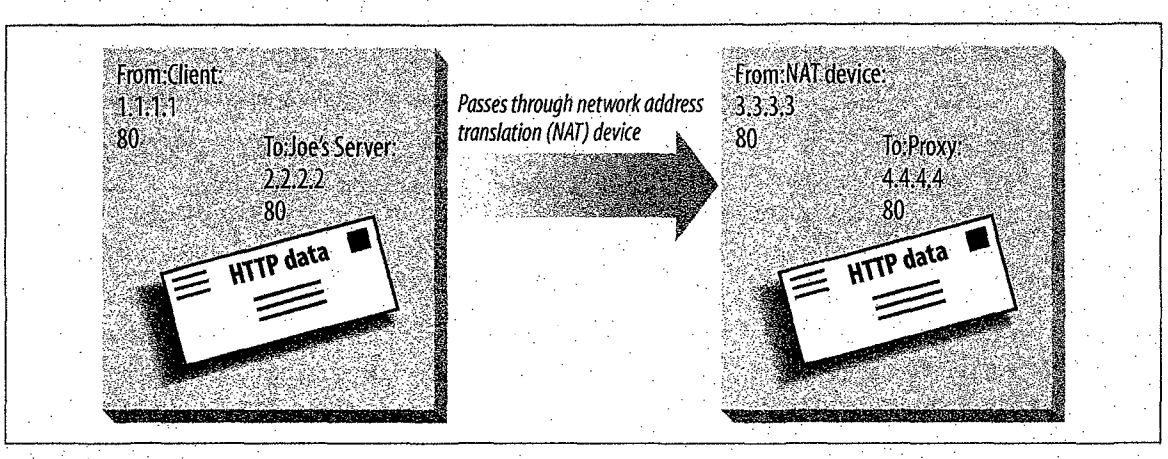


Figure 20-9. Full NAT of a TCP/IP datagram

Network Element Control Protocol

The Network Element Control Protocol (NECP) allows network elements (NEs)—devices such as routers and switches that forward IP packets—to talk with server elements (SEs)—devices such as web servers and proxy caches that serve application layer requests. NECP does not explicitly support load balancing; it only offers a way for an SE to send an NE load-balancing information so that the NE can load balance as it sees fit. Like WCCP, NECP offers several ways to forward packets: MAC forwarding, GRE encapsulation, and NAT.

NECP supports the idea of exceptions. The SE can decide that it cannot service particular source IP addresses, and send those addresses to the NE. The NE can then forward requests from those IP addresses to the origin server.

Messages

The NECP messages are described in Table 20-3.

Table 20-3. NECP messages

Message	Who sends it	Meaning
NECP_NOOP		No operation—do nothing.
NECP_INIT	SE	SE initiates communication with NE. SE sends this message to NE after opening TCP connection with NE. SE must know which NE port to connect to.
NECP_INIT_ACK	NE	Acknowledges NECP_INIT.
NECP_KEEPALIVE	NE or SE	Asks if peer is alive.
NECP_KEEPALIVE_ACK	NE or SE	Answers keep-alive message.
NECP_START	SE	SE says "I am here and ready to accept network traffic." Can specify a port.
NECP_START_ACK	NE	Acknowledges NECP_START.
NECP_STOP	SE	SE tells NE "stop sending me traffic."
NECP_STOP_ACK	NE	NE acknowledges stop.
NECP_EXCEPTION_ADD	SE	SE says to add one or more exceptions to NE's list. Exceptions can be based on source IP, destination IP, protocol (above IP), or port.
NECP_EXCEPTION_ADD_ACK	NE	Confirms EXCEPTION_ADD.
NECP_EXCEPTION_DEL	SE	Asks NE to delete one or more exceptions from its list.
NECP_EXCEPTION_DEL_ACK	NE	Confirms EXCEPTION_DEL.
NECP_EXCEPTION_RESET	SE	Asks NE to delete entire exception list.
NECP_EXCEPTION_RESET_ACK	NE	Confirms EXCEPTION_RESET.
NECP_EXCEPTION_QUERY	SE	Queries NE's entire exception list.
NECP_EXCEPTION_RESP	NE	Responds to exception query.

Proxy Redirection Methods

So far, we have talked about general redirection methods. Content also may need to be accessed through various proxies (potentially for security reasons), or there might be a proxy cache in the network that a client should take advantage of (because it likely will be much faster to retrieve the cached content than it would be to go directly to the origin server).

But how do clients such as web browsers know to go to a proxy? There are three ways to determine this: by explicit browser configuration, by dynamic automatic configuration, and by transparent interception. We will discuss these three techniques in this section.

A proxy can, in turn, redirect client requests to a different proxy. For example, a proxy cache that does not have the content in its cache may choose to redirect the client to another cache. As this results in the response coming from a location different from the one from which the client requested the resource, we also will discuss several protocols used for peer proxy-cache redirection: the Internet Cache Protocol (ICP), the Cache Array Routing Protocol (CARP), and the Hyper Text Caching Protocol (HTCP).

Explicit Browser Configuration

Most browsers can be configured to contact a proxy server for content—there is a pull-down menu where the user can enter the proxy's name or IP address and port number. The browser then contacts the proxy for all requests. Rather than relying on users to correctly configure their browsers to use proxies, some service providers require users to download preconfigured browsers. These browsers know the address of the proxy to contact.

Explicit browser configuration has two main disadvantages:

- Browsers configured to use proxies do not contact the origin server even if the proxy is not responding. If the proxy is down or if the browser is incorrectly configured, the user experiences connectivity problems.
- It is difficult to make changes in network architecture and propagate those changes to all end users. If a service provider wants to add more proxies or take some out of service, browser users have to change their proxy settings.

Proxy Auto-configuration

Explicit configuration of browsers to contact specific proxies can restrict changes in network architecture, because it depends on users to intervene and reconfigure their browsers. An automatic configuration methodology that allows browsers to dynamically configure themselves to contact the correct proxy server solves this problem. Such a methodology exists; it is called the Proxy Auto-configuration (PAC) protocol. PAC was defined by Netscape and is supported by the Netscape Navigator and Microsoft Internet Explorer browsers.

The basic idea behind PAC is to have browsers retrieve a special file, called the PAC file, which specifies the proxy to contact for each URL. The browser must be configured to contact a specific server for the PAC file. The browser then fetches the PAC file every time it is restarted.

The PAC file is a JavaScript file, which must define the function:

```
function FindProxyForURL(url, host)
```

Browsers call this function for every requested URL, as follows:

```
return_value = FindProxyForURL(url_of_request, host_in_url);
```

where the return value is a string specifying where the browser should request this URL. The return value can be a list of the names of proxies to contact (for example, "PROXY proxy1.domain.com; PROXY proxy2.domain.com") or the string "DIRECT", which means that the browser should go directly to the origin server, bypassing any proxies.

The sequence of operations that illustrate the request for and response to a browser's request for the PAC file are illustrated in Figure 20-10. In this example, the server

sends back a PAC file with a JavaScript program. The JavaScript program has a function called "FindProxyForURL" that tells the browser to contact the origin server directly if the host in the requested URL is in the "netscape.com" domain, and to go to "proxy1.joes-cache.com" for all other requests. The browser calls this function for each URL it requests and connects according to the results returned by the function.

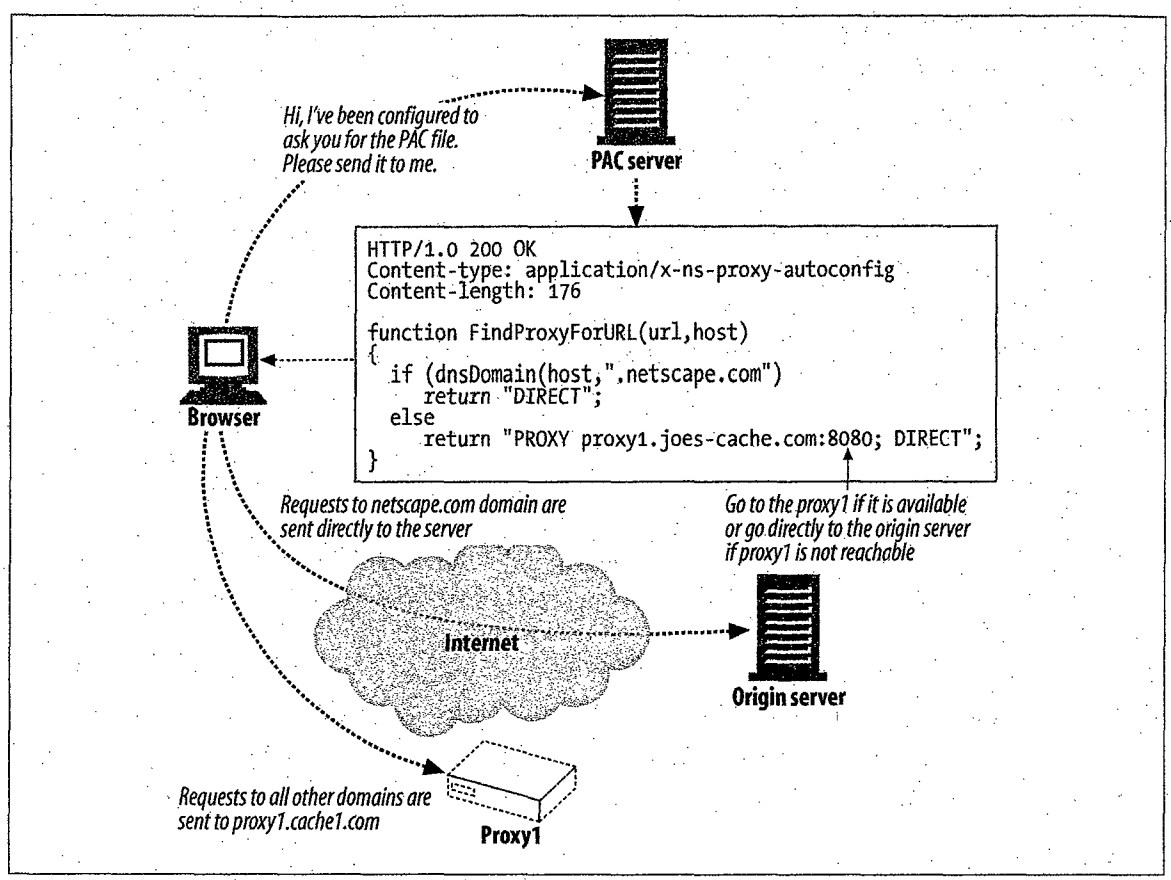


Figure 20-10. Proxy auto-configuration

The PAC protocol is quite powerful: the JavaScript program can ask the browser to choose a proxy based on any of a number of parameters related to the hostname, such as the DNS address and subnet, and even the day of week or time of day. PAC allows browsers automatically to contact the right proxy with changes in network architecture, as long as the PAC file is updated at the server to reflect changes to the proxy locations. The main drawback with PAC is that the browser must be configured to know which server to fetch the PAC file from, so it is not a completely automatic configuration system. WPAD, discussed in the next section, addresses this problem.

PAC, like preconfigured browsers, is used by some major ISPs today.

Web Proxy Autodiscovery Protocol

The Web Proxy Autodiscovery Protocol (WPAD) aims to provide a way for web browsers to find and use nearby proxies, without requiring the end user to manually

configure a proxy setting and without relying on transparent traffic interception. The general problem of defining a web proxy autodiscovery protocol is complicated by the existence of many discovery protocols to choose from and the differences in proxy-use configurations in different browsers.

This section contains an abbreviated and slightly reorganized version of the WPAD Internet draft. The draft currently is being developed as part of the Web Intermediaries Working Group of the IETF.

PAC file autodiscovery

WPAD enables HTTP clients to locate a PAC file and use the PAC file to discover the name of an appropriate proxy server. WPAD does not directly determine the name of the proxy server, because that would circumvent the additional capabilities provided by PAC files (load balancing, request routing to an array of servers, automated failover to backup proxy servers, and so on).

As shown in Figure 20-11, the WPAD protocol discovers a PAC file URL, also known as a *configuration URL* (CURL). The PAC file executes a JavaScript program that returns the address of an appropriate proxy server.

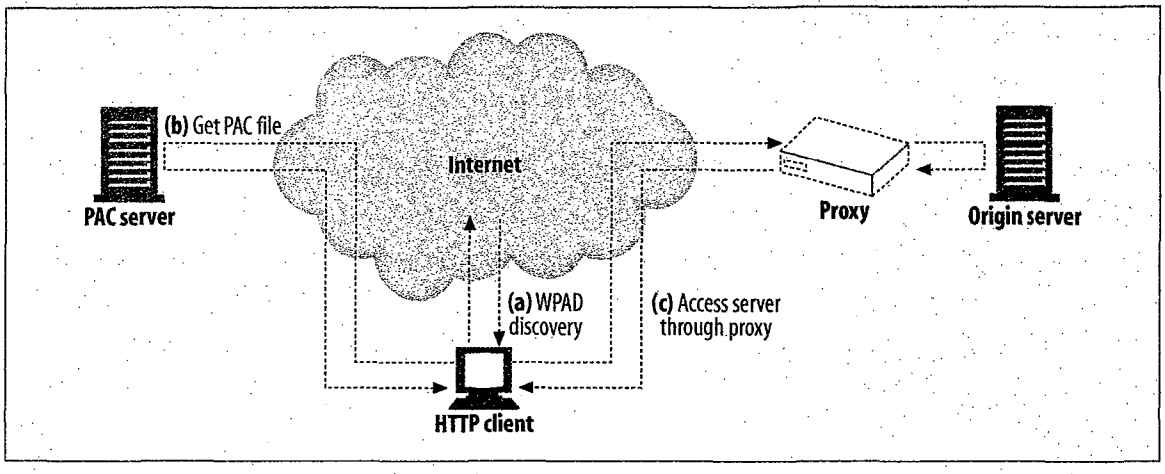


Figure 20-11. WPAD determines the PAC URL, which determines the proxy server

An HTTP client that implements the WPAD protocol:

- Uses WPAD to find the PAC file CURL
- Fetches the PAC file (a.k.a. configuration file, or CFILE) corresponding to the CURL
- Executes the PAC file to determine the proxy server
- Sends HTTP requests to the proxy server returned by the PAC file

WPAD algorithm

WPAD uses a series of resource-discovery techniques to determine the proper PAC file CURL. Multiple discovery techniques are specified, because not all organizations

can use all techniques. WPAD clients attempt each technique, one by one, until they succeed in obtaining a CURL.

The current WPAD specification defines the following techniques, in order:

- DHCP (Dynamic Host Configuration Protocol)
- SLP (Service Location Protocol)
- DNS well-known hostnames
- DNS SRV records
- DNS service URLs in TXT records

Of these five mechanisms, only the DHCP and DNS well-known hostname techniques are required for WPAD clients. We present more details in subsequent sections.

The WPAD client sends a series of resource-discovery requests, using the discovery mechanisms mentioned above, in order. Clients attempt only mechanisms that they support. Whenever a discovery attempt succeeds, the client uses the information obtained to construct a PAC CURL.

If a PAC file is retrieved successfully at that CURL, the process completes. If not, the client resumes where it left off in the predefined series of resource-discovery requests. If, after trying all discovery mechanisms, no PAC file is retrieved, the WPAD protocol fails and the client is configured to use no proxy server.

The client tries DHCP first, followed by SLP. If no PAC file is retrieved, the client moves on to the DNS-based mechanisms.

The client cycles through the DNS SRV, well-known hostnames, and DNS TXT record methods multiple times. Each time, the DNS query QNAME is made less and less specific. In this manner, the client can locate the most specific configuration information possible, but still can fall back on less specific information. Every DNS lookup has the QNAME prefixed with "wpad" to indicate the resource type being requested.

Consider a client with hostname *johns-desktop.development.foo.com*. This is the sequence of discovery attempts a complete WPAD client would perform:

- DHCP
- SLP
- DNS A lookup on "QNAME=wpad.development.foo.com"
- DNS SRV lookup on "QNAME=wpad.development.foo.com"
- DNS TXT lookup on "QNAME=wpad.development.foo.com"
- DNS A lookup on "QNAME=wpad.foo.com"
- DNS SRV lookup on "QNAME=wpad.foo.com"
- DNS TXT lookup on "QNAME=wpad.foo.com"

Refer to the WPAD specification to get detailed pseudocode that addresses the entire sequence of operations. The following sections discuss the two required mechanisms, DHCP and DNS A lookup. For more details about the reminder of the CURL discovery methods, refer to the WPAD specification.

CURL discovery using DHCP

For this mechanism to work, the CURLs must be stored on DHCP servers that WPAD clients can query. The WPAD client obtains the CURL by sending a DHCP query to a DHCP server. The CURL is contained in DHCP option code 252 (if the DHCP server is configured with this information). All WPAD client implementations are required to support DHCP. The DHCP protocol is detailed in RFC 2131. See RFC 2132 for a list of existing DHCP options.

If the WPAD client already has conducted DHCP queries during its initialization, the DHCP server might already have supplied that value. If the value is not available through a client OS API, the client sends a DHCPINFORM message to query the DHCP server to obtain the value.

The DHCP option code 252 for WPAD is of type STRING and is of arbitrary size. This string contains a URL that points to an appropriate PAC file. For example:

```
"http://server.domain/proxyconfig.pac"
```

DNS A record lookup

For this mechanism to work, the IP addresses of suitable proxy servers must be stored on DNS servers that the WPAD clients can query. The WPAD client obtains the CURL by sending an A record lookup to a DNS server. The result of a successful lookup contains an IP address for an appropriate proxy server.

WPAD client implementations are required to support this mechanism. This should be straightforward, as only basic DNS lookup of A records is required. See RFC 2219 for a description of using well-known DNS aliases for resource discovery. For WPAD, the specification uses "well known alias" of "wpad" for web proxy autodiscovery.

The client performs the following DNS lookup:

```
QNAME=wpad.TGTDOM., QCLASS=IN, QTYPE=A
```

A successful lookup contains an IP address from which the WPAD client constructs the CURL.

Retrieving the PAC file

Once a candidate CURL is created, the WPAD client usually makes a GET request to the CURL. When making requests, WPAD clients are required to send Accept headers with appropriate CFILE format information that they are capable of handling. For example:

```
Accept: application/x-ns-proxy-autoconfig
```

In addition, if the CURL results in a redirect, the clients are required to follow the redirect to its final destination.

When to execute WPAD

The web proxy autodiscovery process is required to occur at least as frequently as one of the following:

- Upon startup of the web client—WPAD is performed only for the start of the first instance. Subsequent instances inherit the settings.
- Whenever there is an indication from the networking stack that the IP address of the client host has changed.

A web client can use either option, depending on what makes sense in its environment. In addition, the client must attempt a discovery cycle upon expiration of a previously downloaded PAC file in accordance with HTTP expiration. It's important that the client obey the timeouts and rerun the WPAD process when the PAC file expires.

Optionally, the client also may implement rerunning the WPAD process on failure of the currently configured proxy if the PAC file does not provide an alternative.

Whenever the client decides to invalidate the current PAC file, it must rerun the entire WPAD protocol to ensure it discovers the currently correct CURL. Specifically, there is no provision in the protocol to do an If-Modified-Since conditional fetch of the PAC file.

A number of network round trips might be required during the WPAD protocol broadcast and/or multicast communications. The WPAD protocol should not be invoked at a more frequent rate than specified above (such as per-URL retrieval).

WPAD spoofing

The IE 5 implementation of WPAD enabled web clients to detect proxy settings automatically, without user intervention. The algorithm used by WPAD prepends the hostname "wpad" to the fully qualified domain name and progressively removes subdomains until it either finds a WPAD server answering the hostname or reaches the third-level domain. For instance, web clients in the domain *a.b.microsoft.com* would query *wpad.a.b.microsoft*, *wpad.b.microsoft.com*, then *wpad.microsoft.com*.

This exposed a security hole, because in international usage (and certain other configurations), the third-level domain may not be trusted. A malicious user could set up a WPAD server and serve proxy configuration commands of her choice. Subsequent versions of IE (5.01 and later) rectified the problem.

Timeouts

WPAD goes through multiple levels of discovery, and clients must make sure that each phase is time-bound. When possible, limiting each phase to 10 seconds is

considered reasonable, but implementors may choose a different value that is more appropriate to their network properties. For example, a device implementation, operating over a wireless network, might use a much larger timeout to account for low bandwidth or high latency.

Administrator considerations

Administrators should configure at least one of the DHCP or DNS A record lookup methods in their environments, as those are the only two that all compatible clients are required to implement. Beyond that, configuring to support mechanisms earlier in the search order will improve client startup time.

One of the major motivations for this protocol structure was to support client location of nearby proxy servers. In many environments, there are several proxy servers (workgroup, corporate gateway, ISP, backbone).

There are a number of possible points at which "nearness" decisions can be made in the WPAD framework:

- DHCP servers for different subnets can return different answers. They also can base decisions on the client cipaddr field or the client identifier option.
- DNS servers can be configured to return different SRV/A/TXT resource records (RRs) for different domain suffixes (for example, QNAMEs *wpad.marketing.bigcorp.com* and *wpad.development.bigcorp.com*).
- The web server handling the CURL request can make decisions based on the User-Agent header, Accept header, client IP address/subnet/hostname, topological distribution of nearby proxy servers, etc. This can occur inside a CGI executable created to handle the CURL. As mentioned earlier, it even can be a proxy server handling the CURL requests and making these decisions.
- The PAC file may be expressive enough to select from a set of alternatives at runtime on the client. CARP is based on this premise for an array of caches. It is not inconceivable that the PAC file could compute some network distance or fitness metrics to a set of candidate proxy servers and then select the "closest" or "most responsive" server.

Cache Redirection Methods

We've discussed techniques to redirect traffic to general servers and specialized techniques to vector traffic to proxies and gateways. This final section will explain some of the more sophisticated redirection techniques used for caching proxy servers. These techniques are more complex than the previously discussed protocols because they try to be reliable, high-performance, and content-aware—dispatching requests to locations likely to have particular pieces of content.

WCCP Redirection

Cisco Systems developed the Web Cache Coordination Protocol (WCCP) to enable routers to redirect web traffic to proxy caches. WCCP governs communication between routers and caches so that routers can verify caches (make sure they are up and running), load balance among caches, and send specific types of traffic to specific caches. WCCP Version 2 (WCCP2) is an open protocol. We'll discuss WCCP2 here.

How WCCP redirection works

Here's a brief overview of how WCCP redirection works for HTTP (WCCP redirects other protocols similarly):

- Start with a network containing WCCP-enabled routers and caches that can communicate with one another.
- A set of routers and their target caches form a WCCP service group. The configuration of the service group specifies what traffic is sent where, how traffic is sent, and how load should be balanced among the caches in the service group.
- If the service group is configured to redirect HTTP traffic, routers in the service group send HTTP requests to caches in the service group.
- When an HTTP request arrives at a router in the service group, the router chooses one of the caches in the service group to serve the request (based on either a hash on the request's IP address or a mask/value set pairing scheme).
- The router sends the request packets to the cache, either by encapsulating the packets with the cache's IP address or by IP MAC forwarding.
- If the cache cannot serve the request, the packets are returned to the router for normal forwarding.
- The members of the service group exchange heartbeat messages with one another, continually verifying one another's availability.

WCCP2 messages

There are four WCCP2 messages, described in Table 20-4.

Table 20-4. WCCP2 messages

Message name	Who sends it	Information carried
WCCP2_HERE_I_AM	Cache to router	These messages tell routers that caches are available to receive traffic. The messages contain all of the cache's service group information. As soon as a cache joins a service group, it sends these messages to all routers in the group. These messages negotiate with routers sending WCCP2_I_SEE_YOU messages.
WCCP2_I_SEE_YOU	Router to cache	These messages respond to WCCP2_HERE_I_AM messages. They are used to negotiate the packet forwarding method, assignment method (who is the designated cache), packet return method, and security.

Table 20-4. WCCP2 messages (continued)

Message name	Who sends it	Information carried
WCCP2_REDIRECT_ASSIGN	Designated cache to router	These messages make assignments for load balancing; they send bucket information for hash table load balancing or mask/value set pair information for mask/value load balancing.
WCCP2_REMOVAL_QUERY	Router to cache that has not sent WCCP2_HERE_I_AM messages for 2.5 × HERE_I_AM_T seconds	If a router does not receive WCCP2_HERE_I_AM messages regularly, the router sends this message to see if the cache should be removed from the service group. The proper response from a cache is three identical WCCP2_HERE_I_AM messages, separated by HERE_I_AM_T/10 seconds.

The WCCP2_HERE_I_AM message format is:

```
WCCP Message Header
Security Info Component
Service Info Component
Web-cache Identity Info Component
Web-cache View Info Component
Capability Info Component (optional)
Command Extension Component (optional)
```

The WCCP2_I_SEE_YOU message format is:

```
WCCP Message Header
Security Info Component
Service Info Component
Router Identity Info Component
Router View Info Component
Capability Info Component (optional)
Command Extension Component (optional)
```

The WCCP2_REDIRECT_ASSIGN message format is:

```
WCCP Message Header
Security Info Component
Service Info Component
Assignment Info Component, or Alternate Assignment Component
```

The WCCP2_REMOVAL_QUERY message format is:

```
WCCP Message Header
Security Info Component
Service Info Component
Router Query Info Component
```

Message components

Each WCCP2 message consists of a header and components. The WCCP header information contains the message type (Here I Am, I See You, Assignment, or Removal Query), WCCP version, and message length (not including the length of the header).

The components each begin with a four-octet header describing the component type and length. The component length does not include the length of the component header. The message components are described in Table 20-5.

Table 20-5. WCCP2 message components

Component	Description

Security Info	Contains the security option and security implementation. The security option can be:
	`WCCP2_NO_SECURITY (0)`
	`WCCP2_MD5_SECURITY (1)`
	If the option is no security, the security implementation field does not exist. If the option is MD5, the security implementation field is a 16-octet field containing the message checksum and Service Group password. The password can be no more than eight octets.

Service Info	Describes the service group. The service type ID can have two values:
	`WCCP2_SERVICE_STANDARD (0)`
	`WCCP2_SERVICE_DYNAMIC (1)`
	If the service type is standard, the service is a well-known service, defined entirely by service ID. HTTP is an example of a well-known service. If the service type is dynamic, the following settings define the service: priority, protocol, service flags (which determine hashing), and port.

Router Identity Info	Contains the router IP address and ID, and lists (by IP address) all of the web caches with which the router intends to communicate.
Web Cache Identity Info	Contains the web cache IP address and redirection hash table mapping.
Router View Info	Contains the router's view of the service group (identities of the routers and caches).
Web Cache View Info	Contains the web cache's view of the service group.
Assignment Info	Shows the assignment of a web cache to a particular hashing bucket.
Router Query Info	Contains the router's IP address, address of the web cache being queried, and ID of the last router in the service group that received a Here I Am message from the web cache.
Capabilities Info	Used by routers to advertise supported packet forwarding, load balancing, and packet return methods; used by web caches to let routers know what method the web cache prefers.
Alternate Assignment	Contains hash table assignment information for load balancing.
Assignment Map	Contains mask/value set elements for service group.
Command Extension	Used by web caches to tell routers they are shutting down; used by routers to acknowledge a cache shutdown.

Service groups

A *service group* consists of a set of WCCP-enabled routers and caches that exchange WCCP messages. The routers send web traffic to the caches in the service group. The configuration of the service group determines how traffic is distributed to caches in the service group. The routers and caches exchange service group configuration information in Here I Am and I See You messages.

GRE packet encapsulation

Routers that support WCCP redirect HTTP packets to a particular server by encapsulating them with the server's IP address. The packet encapsulation also contains an IP header proto field that indicates Generic Router Encapsulation (GRE). The existence of the proto field tells the receiving proxy that it has an encapsulated packet.

Because the packet is encapsulated, the client IP address is not lost. Figure 20-12 illustrates GRE packet encapsulation.

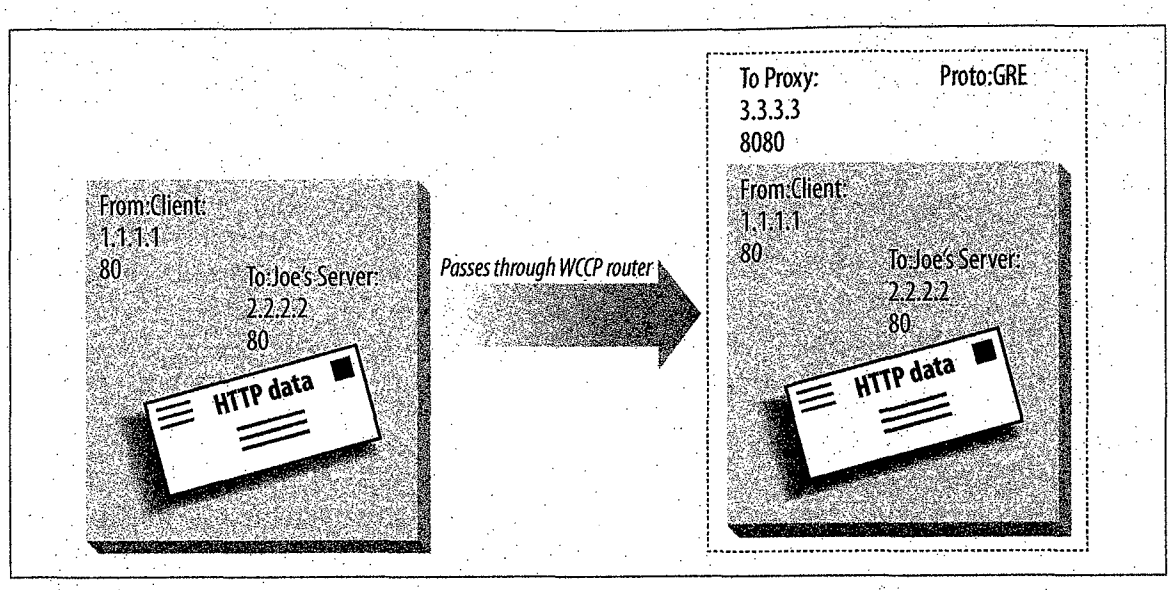


Figure 20-12. How a WCCP router changes an HTTP packet's destination IP address

WCCP load balancing

In addition to routing, WCCP routers can balance load among several receiving servers. WCCP routers and their receiving servers exchange *heartbeat messages* to let one another know they are up and running. If a particular receiving server stops sending heartbeat messages, the WCCP router sends request traffic directly to the Internet, instead of redirecting it to that node. When the node returns to service, the WCCP router begins receiving heartbeat messages again and resumes sending request traffic to the node.

Internet Cache Protocol

The Internet Cache Protocol (ICP) allows caches to look for content hits in sibling caches. If a cache does not have the content requested in an HTTP message, it can find out if the content is in a nearby sibling cache and, if so, retrieve the content from there, hopefully avoiding a more costly query to an origin server. ICP can be thought of as a cache clustering protocol. It is a redirection protocol in the sense that the final destination of an HTTP request message can be determined by a series of ICP queries.

ICP is an object discovery protocol. It asks nearby caches, all at the same time, if any of them have a particular URL in their caches. The nearby caches send back a short message saying "HIT" if they have that URL or "MISS" if they don't. The cache is then free to open an HTTP connection to a neighbor cache that has the object.

ICP is simple and lightweight. ICP messages are 32-bit packed structures in network byte order, making them easy to parse. They are carried in UDP datagrams for

efficiency. UDP is an unreliable Internet protocol, which means that the data can get destroyed in transit, so programs that speak ICP need to have timeouts to detect lost datagrams.

Here is a brief description of the parts of an ICP message:

Opcode
: The opcode is an 8-bit value that describes the meaning of the ICP message. Basic opcodes are ICP_OP_QUERY request messages and ICP_OP_HIT and ICP_OP_MISS response messages.

Version
: The 8-bit version number describes the version number of the ICP protocol. The version of ICP used by Squid, documented in Internet RFC 2186, is Version 2.

Message length
: The total size in bytes of the ICP message. Because there are only 16 bits, the ICP message size cannot be larger than 16,383 bytes. URLs usually are shorter than 16 KB; if they're longer than that, many web applications will not process them.

Request number
: ICP-enabled caches use the request number to keep track of multiple simultaneous requests and replies. An ICP reply message always must contain the same request number as the ICP request message that triggered the reply.

Options
: The 32-bit ICP options field is a bit vector containing flags that modify ICP behavior. ICPv2 defines two flags, both of which modify ICP_OP_QUERY requests. The ICP_FLAG_HIT_OBJ flag enables and disables the return of document data in ICP responses. The ICP_FLAG_SRC_RTT flag requests an estimate of the round-trip time to the origin server, as measured by a sibling cache.

Option data
: The 32-bit option data is reserved for optional features. ICPv2 uses the low 16 bits of the option data to hold an optional round-trip time estimate from the sibling to the origin server.

Sender host address
: A historic field carrying the 32-bit IP address of the message sender; not used in practice.

Payload
: The contents of the payload vary depending on the message type. For ICP_OP_QUERY, the payload is a 4-byte original requester host address followed by a NUL-terminated URL. For ICP_OP_HIT_OBJ, the payload is a NUL-terminated URL followed by a 16 bit object size, followed by the object data.

For more information about ICP, refer to informational RFCs 2186 and 2187. Excellent ICP and peering references also are available from the U.S. National Laboratory for Applied Network Research (*http://www.nlanr.net/Squid/*).

Cache Array Routing Protocol

Proxy servers greatly reduce traffic to the Internet by intercepting requests from individual users and serving cached copies of the requested web objects. However, as the number of users grows, a high volume of traffic can overload the proxy servers themselves.

One solution to this problem is to use multiple proxy servers to distribute the load to a collection of servers. The Cache Array Routing Protocol (CARP) is a standard proposed by Microsoft Corporation and Netscape Communication Corporation to administer a collection of proxy servers such that an array of proxy servers appears to clients as one logical cache.

CARP is an alternative to ICP. Both CARP and ICP allow administrators to improve performance by using multiple proxy servers. This section discusses how CARP differs from ICP, the advantages and disadvantages of using CARP over ICP, and the technical details of how the CARP protocol is implemented.

Upon a cache miss in ICP, the proxy server queries neighboring caches using an ICP message format to determine the availability of the web object. The neighboring caches respond with either a "HIT" or a "MISS," and the requesting proxy server uses these responses to select the most appropriate location from which to retrieve the object. If the ICP proxy servers were arranged in a hierarchical fashion, a miss would be elevated to the parent. Figure 20-13 diagrammatically shows how hits and misses are resolved using ICP.

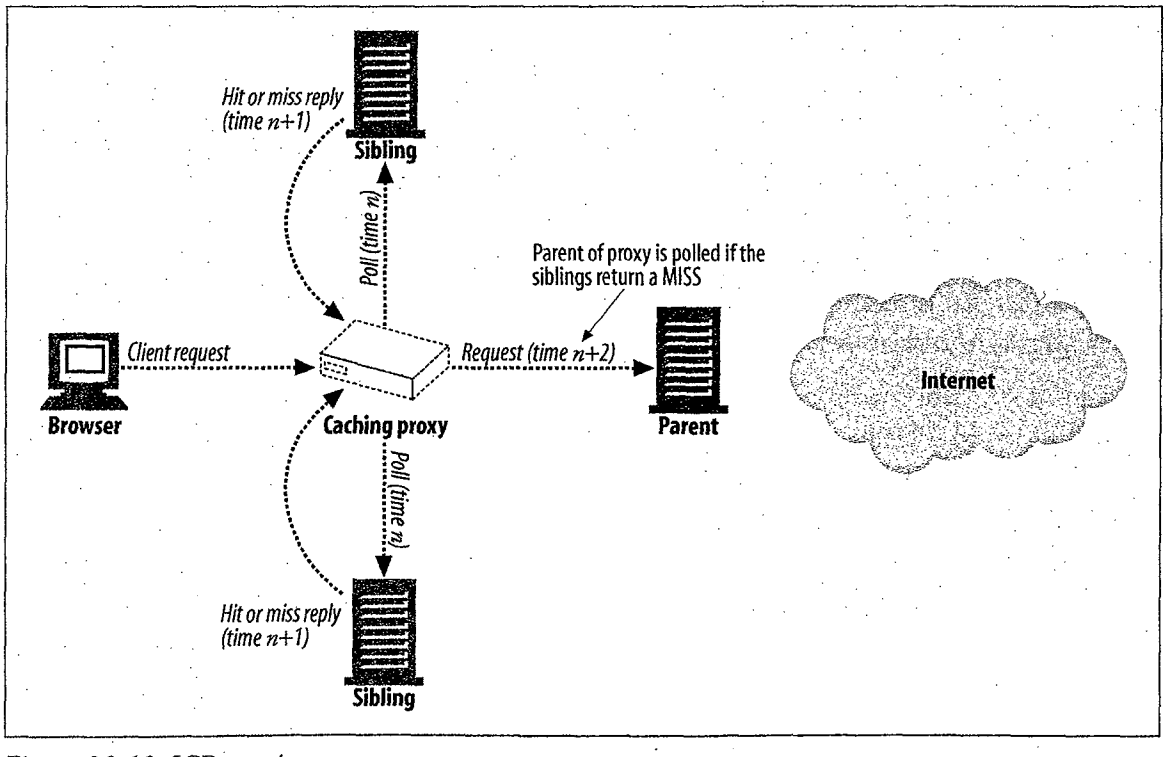


Figure 20-13. ICP queries

Note that each of the proxy servers, connected together using the ICP protocol, is a standalone cache server with redundant mirrors of content, meaning that duplicate entries of web objects across proxy servers is possible. In contrast, the collection of servers connected using CARP operates as a single, large server with each component server containing only a fraction of the total cached documents. By applying a hash function to the URL of a web object, CARP maps web objects to a specific proxy server. Because each web object has a unique home, we can determine the location of the object by a single lookup, rather than polling each of the proxy servers configured in the collection. Figure 20-14 summarizes the CARP approach.

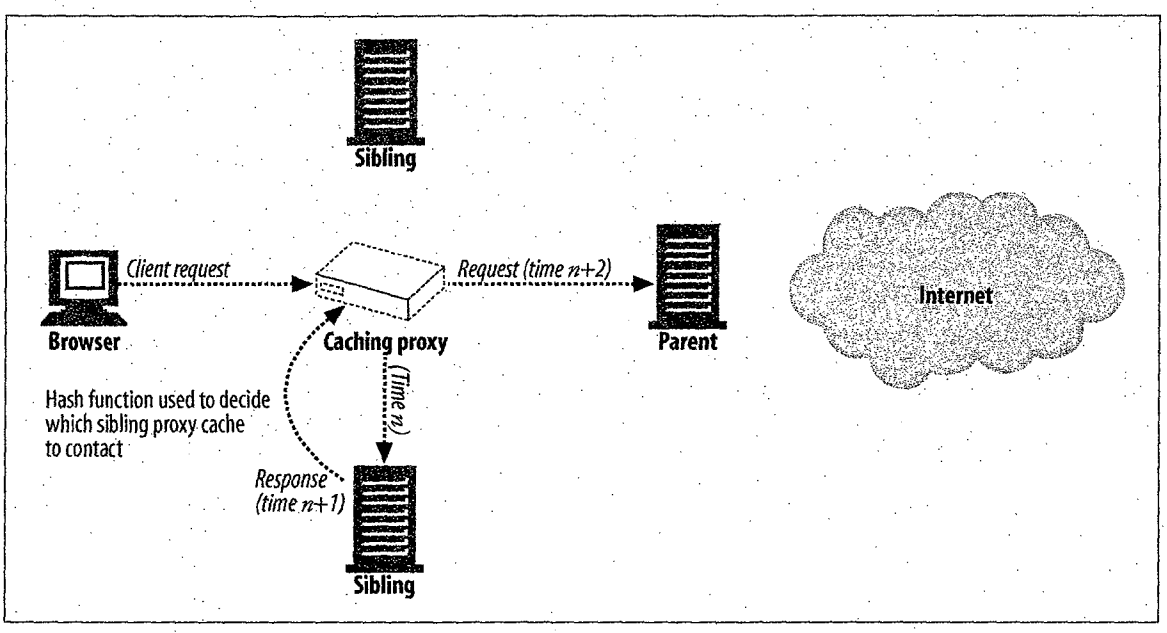


Figure 20-14. CARP redirection

Although Figure 20-14 shows the caching proxy as being the intermediary between clients and proxy servers that distributes the load to the various proxy servers, it is possible for this function to be served by the clients themselves. Commercial browsers such as Internet Explorer and Netscape Navigator can be configured to compute the hash function in the form of a plug-in that determines the proxy server to which the request should be sent.

Deterministic resolution of the proxy server in CARP means that it isn't necessary to send queries to all the neighbors, which means that this method requires fewer inter-cache messages to be sent out. As more proxy servers are added to the configuration, the collective cache system will scale fairly well. However, a disadvantage of CARP is that if one of the proxy servers becomes unavailable, the hash function needs to be modified to reflect this change, and the contents of the proxy servers must be reshuffled across the existing proxy servers. This can be expensive if the proxy server crashes often. In contrast, redundant content in ICP proxy servers

means that reshuffling is not required. Another potential problem is that, because CARP is a new protocol, existing proxy servers running only the ICP protocol may not be included readily in a CARP collection.

Having described the difference between CARP and ICP, let us now describe CARP in a little more detail. The CARP redirection method involves the following tasks:

- Keep a table of participating proxy servers. These proxy servers are polled periodically to see which ones are still active.
- For each participating proxy server, compute a hash function. The value returned by the hash function takes into account the amount of load this proxy can handle.
- Define a separate hash function that returns a number based on the URL of the requested web object.
- Take the sum of the hash function of the URL and the hash function of the proxy servers to get an array of numbers. The maximum value of these numbers determines the proxy server to use for the URL. Because the computed values are deterministic, subsequent requests for the same web object will be forwarded to the same proxy server.

These four chores can either be carried out on the browser, in a plug-in, or be computed on an intermediate server.

For each collection of proxy servers, create a table listing all of the servers in the collection. Each entry in the table should contain information about load factors, time-to-live (TTL) countdown values, and global parameters such as how often members should be polled. The load factor indicates how much load that machine can handle, which depends on the CPU speed and hard drive capacity of that machine. The table can be maintained remotely via an RPC interface. Once the fields in the tables have been updated by RPC, they can be made available or published to downstream clients and proxies. This publication is done in HTTP, allowing any client or proxy server to consume the table information without introducing another inter-proxy protocol. Clients and proxy servers simply use a well-known URL to retrieve the table.

The hash function used must ensure that the web objects are statistically distributed across the participating proxy servers. The load factor of the proxy server should be used to determine the statistic probability of a web object being assigned to that proxy.

In summary, the CARP protocol allows a group of proxy servers to be viewed as single collective cache, instead of a group of cooperating but separate caches (as in ICP). A deterministic request resolution path finds the home of a specific web object within a single hop. This eliminates the inter-proxy traffic that often is generated to

find the web object in a group of proxy servers in ICP. CARP also avoids duplicate copies of web objects being stored on different proxy servers, which has the advantage that the cache system collectively has a larger capacity for storing web objects but also has the disadvantage that a failure in any one proxy requires reshuffling some of the cache contents to existing proxies.

Hyper Text Caching Protocol

Earlier, we discussed ICP, a protocol that allows proxy caches to query siblings about the presence of documents. ICP, however, was designed with HTTP/0.9 in mind and therefore allows caches to send just the URL when querying a sibling about the presence of a resource. Versions 1.0 and 1.1 of HTTP introduced many new request headers that, along with the URL, are used to make decisions about document matching, so simply sending the URL in a request may not result in accurate responses.

The Hyper Text Caching Protocol (HTCP) reduces the probability of false hits by allowing siblings to query each other for the presence of documents using the URL and all of the request and response headers. Further, HTCP allows sibling caches to monitor and request the addition and deletion of selected documents in each other's caches and to make changes in the caching policies of each other's cached documents.

Figure 20-13, which illustrates an ICP transaction, also can be used to illustrate an HTCP transaction—HTCP is just another object discovery protocol. If a nearby cache has the document, the requesting cache can open an HTTP connection to the cache to get a copy of the document. The difference between an ICP and an HTCP transaction is in the level of detail in the requests and responses.

The structure of HTCP messages is illustrated in Figure 20-15. The Header portion includes the message length and message versions. The Data portion starts with the data length and includes opcodes, response codes, and some flags and IDs, and it terminates with the actual data. An optional Authentication section may follow the Data section.

Details of the message fields are as follows:

Header

The Header section consists of a 32-bit message length, an 8-bit major protocol version, and an 8-bit minor protocol version. The message length includes all of the header, data, and authentication sizes.

Data

The Data section contains the HTCP message and has the structure illustrated in Figure 20-15. The data components are described in Table 20-6.

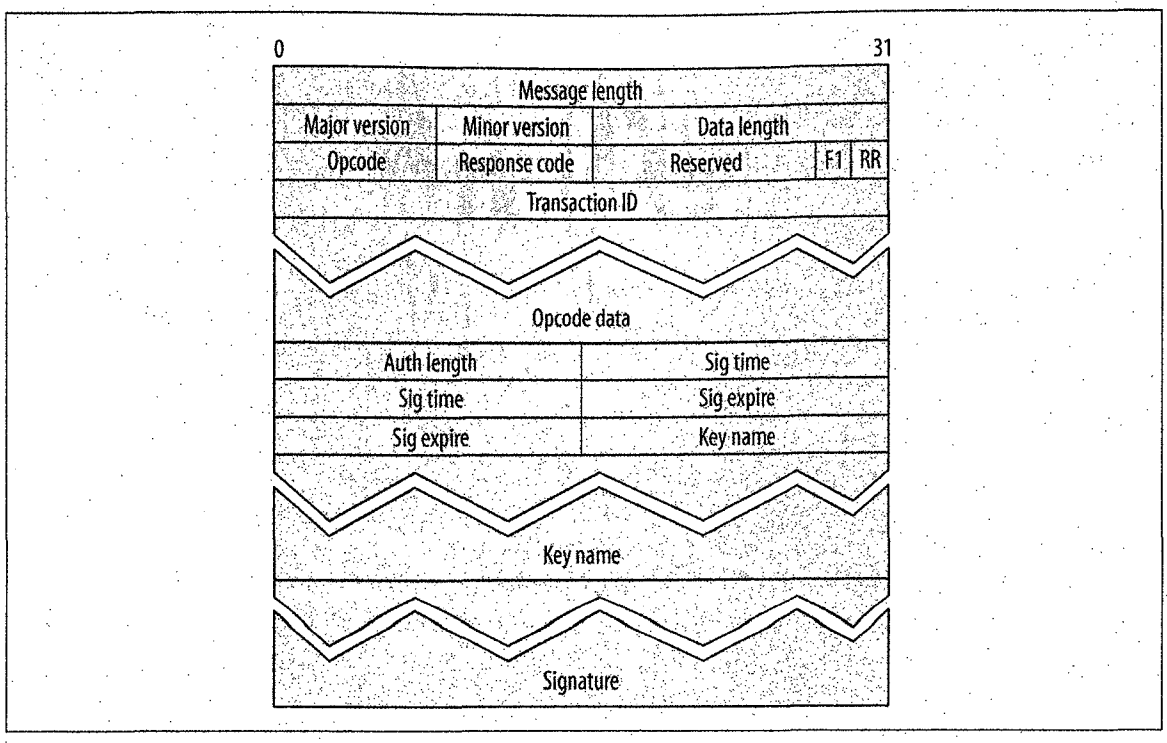


Figure 20-15. HTCP message format

Table 20-6. HTCP data components

Component	Description
Data length	A 16-bit value of the number of bytes in the Data section including the length of the Length field itself.
Opcode	The 4-bit operation code for the HTCP transaction. The full list of opcodes is provided in Table 20-7.
Response code	A 4-bit key indicating the success or failure of the transaction. The possible values are: • 0—Authentication was not used, but is needed • 1—Authentication was used, but is not satisfactory • 2—Unimplemented opcode • 3—Major version not supported • 4—Minor version not supported • 5—Inappropriate, disallowed, or undesirable opcode
F1	F1 is overloaded—if the message is a request, F1is a 1-bit flag set by the requestor indicating that it needs a response (F1=1); if the message is a response, F1 is a 1-bit flag indicating whether the response is to be interpreted as a response to the overall message (F1=1) or just as a response to the Opcode data fields (F1=0).
RR	A 1-bit flag indicating that the message is a request (RR=0) or a response (RR=1).
Transaction ID	A 32-bit value that, combined with the requestor's network address, uniquely identifies the HTCP transaction.
Opcode data	Opcode data is opcode-dependent. See Table 20-7.

Table 20-7 lists the HTCP opcodes and their corresponding data types.

Table 20-7. HTCP opcodes

Opcode	Value	Description	Response codes	Opcode data
NOP	0	Essentially a "ping" operation.	Always 0	None
TST	1		0 if entity is present, 1 if entity is not present	Contains the URL and request headers in the request and just response headers in the response
MON	2		0 if accepted, 1 if refused	
SET	3	The SET message allows caches to request changes in caching policies. See Table 20-9 for a list of the headers that can be used in SET messages.	0 if accepted, 1 if ignored	
CLR	4		0 if I had it, but it's now gone; 1 if I had it, but I am keeping it; and 2 if I didn't have it	

HTCP Authentication

The authentication portion of the HTCP message is optional. Its structure is illustrated in Figure 20-15, and its components are described inTable 20-8.

Table 20-8. HTCP authentication components

Component	Description
Auth length	The 16-bit number of bytes in the Authentication section of the message, including the length of the Length field itself.
Sig time	A 32-bit number representing the number of seconds since 00:00:00 Jan 1, 1970 GMT at the time that the signature is generated.
Sig expire	A 32-bit number representing the number of seconds since 00:00:00 Jan 1, 1970 GMT when the signature will expire.
Key name	A string that specifies the name of the shared secret. The Key section has two parts: the 16-bit length in bytes of the string that follows, followed by the stream of uninterrupted bytes of the string.
Signature	The HMAC-MD5 digest with a B value of 64 (representing the source and destination IP addresses and ports), the major and minor HTCP versions of the message, the Sig time and Sig expires values, the full HTCP data, and the key. The Signature also has two parts: the 16-bit length in bytes of the string, followed by the string.

Setting Caching Policies

The SET message allows caches to request changes in the caching policies of cached documents. The headers that can be used in SET messages are described in Table 20-9.

Table 20-9. List of Cache headers for modifying caching policies

Header	Description
Cache-Vary	The requestor has learned that the content varies on a set of headers different from the set in the response Vary header. This header overrides the response Vary header.
Cache-Location	The list of proxy caches that also may have copies of this object.
Cache-Policy	The requestor has learned the caching policies for this object in more detail than is specified in the response headers. Possible values are: "no-cache," meaning that the response is not cacheable but may be shareable among simultaneous requestors; "no-share," meaning that the object is not shareable; and "no-cache-cookie," meaning that the content may change as a result of cookies and caching therefore is not advised.
Cache-Flags	The requestor has modified the object's caching policies and the object may have to be treated specially and not necessarily in accordance with the object's actual policies.
Cache-Expiry	The actual expiration time for the document as learned by the requestor.
Cache-MD5	The requestor-computed MD5 checksum of the object, which may be different from the value in the Content-MD5 header, or may be supplied because the object does not have a Content-MD5 header.
Cache-to-Origin	The requestor-measured round-trip time to an origin server. The format of the values in this header is *<origin server name or ip> <average round-trip time in seconds> <number of samples> <number of router hops between requestor and origin server>*.

By allowing request and response headers to be sent in query messages to sibling caches, HTCP can decrease the false-hit rate in cache queries. By further allowing sibling caches to exchange policy information with each other, HTCP can improve sibling caches' ability to cooperate with each other.

For More Information

For more information, consult the following references:

DNS and Bind
: Cricket Liu, Paul Albitz, and Mike Loukides, O'Reilly & Associates, Inc.

http://www.wrec.org/Drafts/draft-cooper-webi-wpad-00.txt
: "Web Proxy Auto-Discovery Protocol."

http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html
: "Navigator Proxy Auto-Config File Format."

http://www.ietf.org/rfc/rfc2186.txt
: IETF RFC 2186, "Intercache Communication Protocol (ICP) Version 2," by D. Wessels and K. Claffy.

http://icp.ircache.net/carp.txt
: "Cache Array Routing Protocol v1.0."

http://www.ietf.org/rfc/rfc2756.txt
: IETF RFC 2756, "Hyper Text Caching Protocol (HTCP/0.0)," by P. Vixie and D. Wessels.

http://www.ietf.org/internet-drafts/draft-wilson-wrec-wccp-v2-00.txt
draft-wilson-wrec-wccp-v2-01.txt, "Web Cache Communication Protocol V2.0," by M. Cieslak, D. Forster, G. Tiwana, and R. Wilson.

http://www.ietf.org/rfc/rfc2131.txt?number=2131
"Dynamic Host Configuration Protocol."

http://www.ietf.org/rfc/rfc2132.txt?number=2132
"DHCP Options and BOOTP Vendor Extensions."

http://www.ietf.org/rfc/rfc2608.txt?number=2608
"Service Location Protocol, Version 2."

http://www.ietf.org/rfc/rfc2219.txt?number=2219
"Use of DNS Aliases for Network Services."

CHAPTER 21

Logging and Usage Tracking

Almost all servers and proxies log summaries of the HTTP transactions they process. This is done for a variety of reasons: usage tracking, security, billing, error detection, and so on. In this chapter, we take a brief tour of logging, examining what information about HTTP transactions typically is logged and what some of the common log formats contain.

What to Log?

For the most part, logging is done for two reasons: to look for problems on the server or proxy (e.g., which requests are failing), and to generate statistics about how web sites are accessed. Statistics are useful for marketing, billing, and capacity planning (for instance, determining the need for additional servers or bandwidth).

You could log all of the headers in an HTTP transaction, but for servers and proxies that process millions of transactions per day, the sheer bulk of all of that data quickly would get out of hand. You also would end up logging a lot of information that you don't really care about and may never even look at.

Typically, just the basics of a transaction are logged. A few examples of commonly logged fields are:

- HTTP method
- HTTP version of client and server
- URL of the requested resource
- HTTP status code of the response
- Size of the request and response messages (including any entity bodies)
- Timestamp of when the transaction occurred
- Referer and User-Agent header values

The HTTP method and URL tell what the request was trying to do—for example, GETting a resource or POSTing an order form. The URL can be used to track popularity of pages on the web site.

The version strings give hints about the client and server, which are useful in debugging strange or unexpected interactions between clients and servers. For example, if requests are failing at a higher-than-expected rate, the version information may point to a new release of a browser that is unable to interact with the server.

The HTTP status code tells what happened to the request: whether it was successful, the authorization attempt failed, the resource was found, etc. (See "Status Codes" in Chapter 3 for a list of HTTP status codes.)

The size of the request/response and the timestamp are used mainly for accounting purposes; i.e., to track how many bytes flowed into, out of, or through the application. The timestamp also can be used to correlate observed problems with the requests that were being made at the time.

Log Formats

Several log formats have become standard, and we'll discuss some of the most common formats in this section. Most commercial and open source HTTP applications support logging in one or more of these common formats. Many of these applications also support the ability of administrators to configure log formats and create their own custom formats.

One of the main benefits of supporting (for applications) and using (for administrators) these more standard formats rests in the ability to leverage the tools that have been built to process and generate basic statistics from these logs. Many open source and commercial packages exist to crunch logs for reporting purposes, and by utilizing standard formats, applications and their administrators can plug into these resources.

Common Log Format

One of the most common log formats in use today is called, appropriately, the Common Log Format. Originally defined by NCSA, many servers use this log format as a default. Most commercial and open source servers can be configured to use this format, and many commercial and freeware tools exist to help parse common log files. Table 21-1 lists, in order, the fields of the Common Log Format.

Table 21-1. Common Log Format fields

Field	Description
remotehost	The hostname or IP address of the requestor's machine (IP if the server was not configured to perform reverse DNS or cannot look up the requestor's hostname)
username	If an *ident* lookup was performed, the requestor's authenticated username[a]

Table 21-1. Common Log Format fields (continued)

Field	Description
auth-username	If authentication was performed, the username with which the requestor authenticated
timestamp	The date and time of the request
request-line	The exact text of the HTTP request line, "GET /index.html HTTP/1.1"
response-code	The HTTP status code that was returned in the response
response-size	The Content-Length of the response entity—if no entity was returned in the response, a zero is logged

[a] RFC 931 describes the *ident* lookup used in this authentication. The *ident* protocol was discussed in Chapter 5.

Example 21-1 lists a few examples of Common Log Format entries.

Example 21-1. Common Log Format

```
209.1.32.44 - - [03/Oct/1999:14:16:00 -0400] "GET / HTTP/1.0" 200 1024
http-guide.com - dg [03/Oct/1999:14:16:32 -0400] "GET / HTTP/1.0" 200 477
http-guide.com - dg [03/Oct/1999:14:16:32 -0400] "GET /foo HTTP/1.0" 404 0
```

In these examples, the fields are assigned as follows:

Field	Entry 1	Entry 2	Entry 2
remotehost	209.1.32.44	http-guide.com	http-guide.com
username	<empty>	<empty>	<empty>
auth-username	<empty>	dg	dg
timestamp	03/Oct/1999:14:16:00 -0400	03/Oct/1999:14:16:32 -0400	03/Oct/1999:14:16:32 -0400
request-line	GET / HTTP/1.0	GET / HTTP/1.0	GET /foo HTTP/1.0
response-code	200	200	404
response-size	1024	477	0

Note that the *remotehost* field can be either a hostname, as in *http-guide.com*, or an IP address, such as 209.1.32.44.

The dashes in the second (username) and third (auth-username) fields indicate that the fields are empty. This indicates that either an *ident* lookup did not occur (second field empty) or authentication was not performed (third field empty).

Combined Log Format

Another commonly used log format is the Combined Log Format. This format is supported by servers such as Apache. The Combined Log Format is very similar to the Common Log Format; in fact, it mirrors it exactly, with the addition of two fields (listed in Table 21-2). The User-Agent field is useful in noting which HTTP client applications are making the logged requests, while the Referer field provides more detail about where the requestor found this URL.

Table 21-2. Additional Combined Log Format fields

Field	Description
Referer	The contents of the Referer HTTP header
User-Agent	The contents of the User-Agent HTTP header

Example 21-2 gives an example of a Combined Log Format entry.

Example 21-2. Combined Log Format

```
209.1.32.44 - - [03/Oct/1999:14:16:00 -0400] "GET / HTTP/1.0" 200 1024 "http://www.joes-
hardware.com/"  "5.0: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)"
```

In Example 21-2, the Referer and User-Agent fields are assigned as follows:

Field	Value
Referer	http://www.joes-hardware.com/
User-Agent	5.0: Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)

The first seven fields of the example Combined Log Format entry in Example 21-2 are exactly as they would be in the Common Log Format (see the first entry in Example 21-1). The two new fields, Referer and User-Agent, are tacked onto the end of the log entry.

Netscape Extended Log Format

When Netscape entered into the commercial HTTP application space, it defined for its servers many log formats that have been adopted by other HTTP application developers. Netscape's formats derive from the NCSA Common Log Format, but they extend that format to incorporate fields relevant to HTTP applications such as proxies and web caches.

The first seven fields in the Netscape Extended Log Format are identical to those in the Common Log Format (see Table 21-1). Table 21-3 lists, in order, the new fields that the Netscape Extended Log Format introduces.

Table 21-3. Additional Netscape Extended Log Format fields

Field	Description
proxy-response-code	If the transaction went through a proxy, the HTTP response code from the server to the proxy
proxy-response-size	If the transaction went through a proxy, the Content-Length of the server's response entity sent to the proxy
client-request-size	The Content-Length of any body or entity in the client's request to the proxy
proxy-request-size	If the transaction went through a proxy, the Content-Length of any body or entity in the proxy's request to the server
client-request-hdr-size	The length, in bytes, of the client's request headers

Table 21-3. Additional Netscape Extended Log Format fields (continued)

Field	Description
proxy-response-hdr-size	If the transaction went through a proxy, the length, in bytes, of the proxy's response headers that were sent to the requestor
proxy-request-hdr-size	If the transaction went through a proxy, the length, in bytes, of the proxy's request headers that were sent to the server
server-response-hdr-size	The length, in bytes, of the server's response headers
proxy-timestamp	If the transaction went through a proxy, the elapsed time for the request and response to travel through the proxy, in seconds

Example 21-3 gives an example of a Netscape Extended Log Format entry.

Example 21-3. Netscape Extended Log Format

```
209.1.32.44 - - [03/Oct/1999:14:16:00-0400] "GET / HTTP/1.0" 200 1024 200 1024 0 0 215 260
279 254 3
```

In this example, the extended fields are assigned as follows:

Field	Value
proxy-response-code	200
proxy-response-size	1024
client-request-size	0
proxy-request-size	0
client-request-hdr-size	215
proxy-response-hdr-size	260
proxy-request-hdr-size	279
server-response-hdr-size	254
proxy-timestamp	3

The first seven fields of the example Netscape Extended Log Format entry in Example 21-3 mirror the entries in the Common Log Format example (see the first entry in Example 21-1).

Netscape Extended 2 Log Format

Another Netscape log format, the Netscape Extended 2 Log Format, takes the Extended Log Format and adds further information relevant to HTTP proxy and web caching applications. These extra fields help paint a better picture of the interactions between an HTTP client and an HTTP proxy application.

The Netscape Extended 2 Log Format derives from the Netscape Extended Log Format, and its initial fields are identical to those listed in Table 21-3 (it also extends the Common Log Format fields listed in Table 21-1).

Table 21-4 lists, in order, the additional fields of the Netscape Extended 2 Log Format.

Table 21-4. Additional Netscape Extended 2 Log Format fields

Field	Description
route	The route that the proxy used to make the request for the client (see Table 21-5)
client-finish-status-code	The client finish status code; specifies whether the client request to the proxy completed successfully (FIN) or was interrupted (INTR)
proxy-finish-status-code	The proxy finish status code; specifies whether the proxy request to the server completed successfully (FIN) or was interrupted (INTR)
cache-result-code	The cache result code; tells how the cache responded to the request[a]

[a] Table 21-7 lists the Netscape cache result codes.

Example 21-4 gives an example of a Netscape Extended 2 Log Format entry.

Example 21-4. Netscape Extended 2 Log Format

```
209.1.32.44 - - [03/Oct/1999:14:16:00-0400] "GET / HTTP/1.0" 200 1024 200 1024 0 0 215 260
279 254 3 DIRECT FIN FIN WRITTEN
```

The extended fields in this example are assigned as follows:

Field	Value
route	DIRECT
client-finish-status-code	FIN
proxy-finish-status-code	FIN
cache-result-code	WRITTEN

The first 16 fields in the Netscape Extended 2 Log Format entry in Example 21-4 mirror the entries in the Netscape Extended Log Format example (see Example 21-3).

Table 21-5 lists the valid Netscape route codes.

Table 21-5. Netscape route codes

Value	Description
DIRECT	The resource was fetched directly from the server.
PROXY(host:port)	The resource was fetched through the proxy "host."
SOCKS(socks:port)	The resource was fetched through the SOCKS server "host."

Table 21-6 lists the valid Netscape finish codes.

Table 21-6. Netscape finish status codes

Value	Description
-	The request never even started.
FIN	The request was completed successfully.

Table 21-6. Netscape finish status codes (continued)

Value	Description
INTR	The request was interrupted by the client or ended by a proxy/server.
TIMEOUT	The request was timed out by the proxy/server.

Table 21-7 lists the valid Netscape cache codes.*

Table 21-7. Netscape cache codes

Code	Description
-	The resource was uncacheable.
WRITTEN	The resource was written into the cache.
REFRESHED	The resource was cached and it was refreshed.
NO-CHECK	The cached resource was returned; no freshness check was done.
UP-TO-DATE	The cached resource was returned; a freshness check was done.
HOST-NOT-AVAILABLE	The cached resource was returned; no freshness check was done because the remote server was not available.
CL-MISMATCH	The resource was not written to the cache; the write was aborted because the Content-Length did not match the resource size.
ERROR	The resource was not written to the cache due to some error; for example, a timeout occurred or the client aborted the transaction.

Netscape applications, like many other HTTP applications, have other log formats too, including a Flexible Log Format and a means for administrators to output custom log fields. These formats allow administrators greater control and the ability to customize their logs by choosing which parts of the HTTP transaction (headers, status, sizes, etc.) to report in their logs.

The ability for administrators to configure custom formats was added because it is difficult to predict what information administrators will be interested in getting from their logs. Many other proxies and servers also have the ability to emit custom logs.

Squid Proxy Log Format

The Squid proxy cache (*http://www.squid-cache.org*) is a venerable part of the Web. Its roots trace back to one of the early web proxy cache projects (*ftp://ftp.cs.colorado.edu/pub/techreports/schwartz/Harvest.Conf.ps.Z*). Squid is an open source project that has been extended and enhanced by the open source community over the years. Many tools have been written to help administer the Squid application, including tools to help process, audit, and mine its logs. Many subsequent proxy caches adopted the Squid format for their own logs so that they could leverage these tools.

* Chapter 7 discusses HTTP caching in detail.

The format of a Squid log entry is fairly simple. Its fields are summarized in Table 21-8.

Table 21-8. Squid Log Format fields

Field	Description
timestamp	The timestamp when the request arrived, in seconds since January 1, 1970 GMT.
time-elapsed	The elapsed time for request and response to travel through the proxy, in milliseconds.
host-ip	The IP address of the client's (requestor's) host machine.
result-code/status	The result field is a Squid-ism that tells what action the proxy took during this request[a]; the code field is the HTTP response code that the proxy sent to the client.
size	The length of the proxy's response to the client, including HTTP response headers and body, in bytes.
method	The HTTP method of the client's request.
url	The URL in the client's request.[b]
rfc931-ident[c]	The client's authenticated username.[d]
hierarchy/from	Like the route field in Netscape formats, the hierarchy field tells what route the proxy used to make the request for the client.[e] The from field tells the name of the server that the proxy used to make the request.
content-type	The Content-Type of the proxy response entity.

[a] Table 21-9 lists the various result codes and their meanings.
[b] Recall from Chapter 2 that proxies often log the entire requested URL, so if a username and password component are in the URL, a proxy can inadvertently record this information.
[c] The rfc931-ident, hierarchy/from, and content-type fields were added in Squid 1.1. Previous versions did not have these fields.
[d] RFC 931 describes the *ident* lookup used in this authentication.
[e] *http://squid.nlanr.net/Doc/FAQ/FAQ-6.html#ss6.6* lists all of the valid Squid hierarchy codes.

Example 21-5 gives an example of a Squid Log Format entry.

Example 21-5. Squid Log Format

```
99823414 3001 209.1.32.44 TCP_MISS/200 4087 GET http://www.joes-hardware.com - DIRECT/
proxy.com text/html
```

The fields are assigned as follows:

Field	Value
timestamp	99823414
time-elapsed	3001
host-ip	209.1.32.44
action-code	TCP_MISS
status	200
size	4087
method	GET
URL	http://www.joes-hardware.com

Field	Value
RFC 931 ident	-
hierarchy	DIRECT[a]
from	proxy.com
content-type	text/html

[a] The DIRECT Squid hierarchy value is the same as the DIRECT route value in Netscape log formats.

Table 21-9 lists the various Squid result codes.*

Table 21-9. Squid result codes

Action	Description
TCP_HIT	A valid copy of the resource was served out of the cache.
TCP_MISS	The resource was not in the cache.
TCP_REFRESH_HIT	The resource was in the cache but needed to be checked for freshness. The proxy revalidated the resource with the server and found that the in-cache copy was indeed still fresh.
TCP_REF_FAIL_HIT	The resource was in the cache but needed to be checked for freshness. However, the revalidation failed (perhaps the proxy could not connect to the server), so the "stale" resource was returned.
TCP_REFRESH_MISS	The resource was in the cache but needed to be checked for freshness. Upon checking with the server, the proxy learned that the resource in the cache was out of date and received a new version.
TCP_CLIENT_REFRESH_MISS	The requestor sent a Pragma: no-cache or similar Cache-Control directive, so the proxy was forced to fetch the resource.
TCP_IMS_HIT	The requestor issued a conditional request, which was validated against the cached copy of the resource.
TCP_SWAPFAIL_MISS	The proxy thought the resource was in the cache but for some reason could not access it.
TCP_NEGATIVE_HIT	A cached response was returned, but the response was a negatively cached response. Squid supports the notion of caching errors for resources—for example, caching a 404 Not Found response—so if multiple requests go through the proxy-cache for an invalid resource, the error is served from the proxy cache.
TCP_MEM_HIT	A valid copy of the resource was served out of the cache, and the resource was in the proxy cache's memory (as opposed to having to access the disk to retrieve the cached resource).
TCP_DENIED	The request for this resource was denied, probably because the requestor does not have permission to make requests for this resource.
TCP_OFFLINE_HIT	The requested resource was retrieved from the cache during its *offline* mode. Resources are not validated when Squid (or another proxy using this format) is in offline mode.
UDP_*	The UDP_* codes indicate that requests were received through the UDP interface to the proxy. HTTP normally uses the TCP transport protocol, so these requests are not using the HTTP protocol.[a]

* Several of these action codes deal more with the internals of the Squid proxy cache, so not all of them are used by other proxies that implement the Squid Log Format.

Table 21-9. Squid result codes (continued)

Action	Description
UDP_HIT	A valid copy of the resource was served out of the cache.
UDP_MISS	The resource was not in the cache.
UDP_DENIED	The request for this resource was denied, probably because the requestor does not have permission to make requests for this resource.
UDP_INVALID	The request that the proxy received was invalid.
UDP_MISS_NOFETCH	Used by Squid during specific operation modes or in the cache of frequent failures. A cache miss was returned and the resource was not fetched.
NONE	Logged sometimes with errors.
TCP_CLIENT_REFRESH	See TCP_CLIENT_REFRESH_MISS.
TCP_SWAPFAIL	See TCP_SWAPFAIL_MISS.
UDP_RELOADING	See UDP_MISS_NOFETCH.

[a] Squid has its own protocol for making these requests: ICP. This protocol is used for cache-to-cache requests. See *http://www.squid-cache.org* for more information.

Hit Metering

Origin servers often keep detailed logs for billing purposes. Content providers need to know how often URLs are accessed, advertisers want to know how often their ads are shown, and web authors want to know how popular their content is. Logging works well for tracking these things when clients visit web servers directly.

However, caches stand between clients and servers and prevent many accesses from reaching servers (the very purpose of caches).* Because caches handle many HTTP requests and satisfy them without visiting the origin server, the server has no record that a client accessed its content, creating omissions in log files.

Missing log data makes content providers resort to *cache busting* for their most important pages. Cache busting refers to a content producer intentionally making certain content uncacheable, so all requests for this content must go to the origin server.† This allows the origin server to log the access. Defeating caching might yield better logs, but it slows down requests and increases load on the origin server and network.

Because proxy caches (and some clients) keep their own logs, if servers could get access to these logs—or at least have a crude way to determine how often their content is served by a proxy cache—cache busting could be avoided. The proposed Hit Metering protocol, an extension to HTTP, suggests a solution to this problem. The Hit Metering protocol requires caches to periodically report cache access statistics to origin servers.

* Recall that virtually every browser has a cache.

† Chapter 7 describes how HTTP responses can be marked as uncacheable.

RFC 2227 defines the Hit Metering protocol in detail. This section provides a brief tour of the proposal.

Overview

The Hit Metering protocol defines an extension to HTTP that provides a few basic facilities that caches and servers can implement to share access information and to regulate how many times cached resources can be used.

Hit Metering is, by design, not a complete solution to the problem caches pose for logging access, but it does provide a basic means for obtaining metrics that servers want to track. The Hit Metering protocol has not been widely implemented or deployed (and may never be). That said, a cooperative scheme like Hit Metering holds some promise of providing accurate access statistics while retaining caching performance gains. Hopefully, that will be motivation to implement the Hit Metering protocol instead of marking content uncacheable.

The Meter Header

The Hit Metering extension proposes the addition of a new header, Meter, that caches and servers can use to pass to each other directives about usage and reporting, much like the Cache-Control header allows caching directives to be exchanged.

Table 21-10 defines the various directives and who can pass them in the Meter header.

Table 21-10. Hit Metering directives

Directive	Abbreviation	Who	Description
will-report-and-limit	w	Cache	The cache is capable of reporting usage and obeying any usage limits the server specifies.
wont-report	x	Cache	The cache is able to obey usage limits but won't report usage.
wont-limit	y	Cache	The cache is able to report usage but won't limit usage.
count	c	Cache	The reporting directive, specified as "uses/reuses" integers—for example, ":count=2/4".[a]
max-uses	u	Server	Allows the server to specify the maximum number of times a response can be used by a cache—for example, "max-uses=100".
max-reuses	r	Server	Allows the server to specify the maximum number of times a response can be reused by a cache—for example, "max-reuses=100".
do-report	d	Server	The server requires proxies to send usage reports.
dont-report	e	Server	The server does not want usage reports.
timeout	t	Server	Allows the server to specify a timeout on the metering of a resource. The cache should send a report at or before the specified timeout, plus or minus 1 minute. The timeout is specified in minutes—for example, "timeout=60".
wont-ask	n	Server	The server does not want any metering information.

[a] Hit Metering defines a *use* as satisfying a request with the response, whereas a *reuse* is revalidating a client request.

Figure 21-1 shows an example of Hit Metering in action. The first part of the transaction is just a normal HTTP transaction between a client and proxy cache, but in the proxy request, note the insertion of the Meter header and the response from the server. Here, the proxy is informing the server that it is capable of doing Hit Metering, and the server in turn is asking the proxy to report its hit counts.

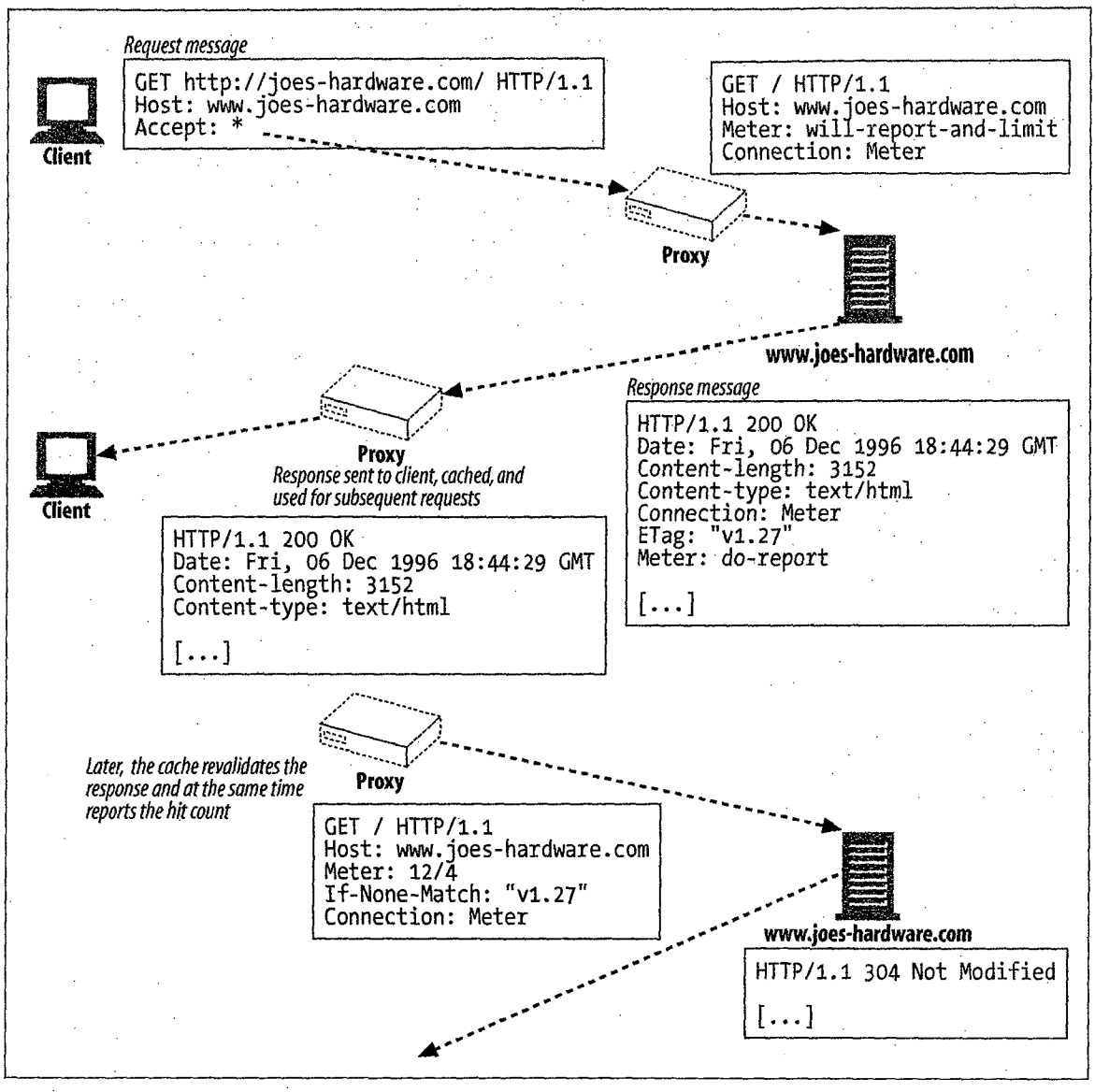


Figure 21-1. Hit Metering example

The request completes as it normally would, from the client's perspective, and the proxy begins tracking hits to that resource on behalf of the server. Later, the proxy tries to revalidate the resource with the server. The proxy embeds the metered information it has been tracking in the conditional request to the server.

A Word on Privacy

Because logging really is an administrative function that servers and proxies perform, the whole operation is transparent to users. Often, they may not even be aware that their HTTP transactions are being logged—in fact, many users probably do not even know that they are using the HTTP protocol when accessing content on the Web.

Web application developers and administrators need to be aware of the implications of tracking a user's HTTP transactions. Much can be gleaned about a user based on the information he retrieves. This information obviously can be put to bad use—discrimination, harassment, blackmail, etc. Web servers and proxies that log must be vigilant in protecting the privacy of their end users.

Sometimes, such as in work environments, tracking a user's usage to make sure he is not goofing off may be appropriate, but administrators also should make public the fact that people's transactions are being monitored.

In short, logging is a very useful tool for the administrator and developer—just be aware of the privacy infringements that logs can have without the permission or knowledge of the users whose actions are being logged.

For More Information

For more information on logging, refer to:

http://httpd.apache.org/docs/logs.html
: "Apache HTTP Server: Log Files." Apache HTTP Server Project web site.

http://www.squid-cache.org/Doc/FAQ/FAQ-6.html
: "Squid Log Files." Squid Proxy Cache web site.

http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format
: "Logging Control in W3C httpd."

http://www.w3.org/TR/WD-logfile.html
: "Extended Log File Format."

http://www.ietf.org/rfc/rfc2227.txt
: RFC 2227, "Simple Hit-Metering and Usage-Limiting for HTTP," by J. Mogul and P. Leach.

PART VI
Appendixes

This collection of appendixes contains useful reference tables, background information, and tutorials on a variety of topics relevant to HTTP architecture and implementation:

- Appendix A, *URI Schemes*
- Appendix B, *HTTP Status Codes*
- Appendix C, *HTTP Header Reference*
- Appendix D, *MIME Types*
- Appendix E, *Base-64 Encoding*
- Appendix F, *Digest Authentication*
- Appendix G, *Language Tags*
- Appendix H, *MIME Charset Registry*

APPENDIX A
URI Schemes

Many URI schemes have been defined, but few are in common use. Generally speaking, those URI schemes with associated RFCs are in more common use, though there are a few schemes that have been developed by leading software corporations (notably Netscape and Microsoft), but not formalized, that also are in wide use.

The W3C maintains a list of URI schemes, which you can view at:

http://www.w3.org/Addressing/schemes.html

The IANA also maintains a list of URL schemes, at:

http://www.iana.org/assignments/uri-schemes

Table A-1 informally describes some of the schemes that have been proposed and those that are in active use. Note that many of the approximately 90 schemes in the table are not widely used, and many are extinct.

Table A-1. URI schemes from the W3C registry

Scheme	Description	RFCs
about	Netscape scheme to explore aspects of the browser. For example: about by itself is the same as choosing "About Communicator" from the Navigator Help menu, about:cache displays disk-cache statistics, and about:plugins displays information about configured plug-ins. Other browsers, such as Microsoft Internet Explorer, also use this scheme.	
acap	Application Configuration Access Protocol.	2244
afp	For file-sharing services using the Apple Filing Protocol (AFP) protocol, defined as part of the expired IETF *draft-ietf-svrloc-afp-service-01.txt*.	
afs	Reserved for future use by the Andrew File System.	
callto	Initiates a Microsoft NetMeeting conference session, such as: *callto: ws3.joes-hardware.com/joe@joes-hardware.com*	
chttp	The CHTTP caching protocol defined by Real Networks. RealPlayer does not cache all items streamed by HTTP. Instead, you designate files to cache by using *chttp://* instead of *http://* in the file's URL. When RealPlayer reads a CHTTP URL in a SMIL file, it first checks its disk cache for the file. If the file isn't present, it requests the file through HTTP, storing the file in its cache.	

Table A-1. URI schemes from the W3C registry (continued)

Scheme	Description	RFCs
cid	The use of [MIME] within email to convey web pages and their associated images requires a URL scheme to permit the HTML to refer to the images or other data included in the message. The Content-ID URL, "cid:", serves that purpose.	2392 2111
clsid	Allows Microsoft OLE/COM (Component Object Model) classes to be referenced. Used to insert active objects into web pages.	
data	Allows inclusion of small, constant data items as "immediate" data. This URL encodes the text/plain string "A brief note": *data:A%20brief%20note*	2397
date	Proposal for scheme to support dates, as in *date:1999-03-04T20:42:08*.	
dav	To ensure correct interoperation based on this specification, the IANA must reserve the URI namespaces starting with "DAV:" and with "opaquelocktoken:" for use by this specification, its revisions, and related WebDAV specifications.	2518
dns	Used by REBOL software. See *http://www.rebol.com/users/valurl.html*.	
eid	The external ID (eid) scheme provides a mechanism by which the local application can reference data that has been obtained by other, non-URL scheme means. The scheme is intended to provide a general escape mechanism to allow access to information for applications that are too specialized to justify their own schemes. There is some controversy about this URI. See *http://www.ics.uci.edu/pub/ietf/uri/draft-finseth-url-00.txt*.	
fax	The "fax" scheme describes a connection to a terminal that can handle telefaxes (facsimile machines).	2806
file	Designates files accessible on a particular host computer. A hostname can be included, but the scheme is unusual in that it does not specify an Internet protocol or access method for such files; as such, its utility in network protocols between hosts is limited.	1738
finger	The finger URL has the form: *finger://host[:port][/<request>]* The <request> must conform with the RFC 1288 request format. See *http://www.ics.uci.edu/pub/ietf/uri/draft-ietf-uri-url-finger-03.txt*.	
freenet	URIs for information in the Freenet distributed information system. See *http://freenet.sourceforge.net*.	
ftp	File Transfer Protocol scheme.	1738
gopher	The archaic gopher protocol.	1738
gsm-sms	URIs for the GSM mobile phone short message service.	
h323, h324	Multimedia conferencing URI schemes. See *http://www.ics.uci.edu/pub/ietf/uri/draft-cordell-sg16-conv-url-00.txt*.	
hdl	The Handle System is a comprehensive system for assigning, managing, and resolving persistent identifiers, known as "handles," for digital objects and other resources on the Internet. Handles can be used as URNs. See *http://www.handle.net*.	
hnews	HNEWS is an HTTP-tunneling variant of the NNTP news protocol. The syntax of hnews URLs is designed to be compatible with the current common usage of the news URL scheme. See *http://www.ics.uci.edu/pub/ietf/uri/draft-stockwell-hnews-url-00.txt*.	

Table A-1. URI schemes from the W3C registry (continued)

Scheme	Description	RFCs
http	The HTTP protocol. Read this book for more information.	2616
https	HTTP over SSL. See *http://sitesearch.netscape.com/eng/ssl3/draft302.txt*.	
iioploc	CORBA extensions. The Interoperable Name Service defines one URL-format object reference, iioploc, that can be typed into a program to reach defined services at remote locations, including the Naming Service. For example, this iioploc identifier: *iioploc://www.omg.org/NameService* would resolve to the CORBA Naming Service running on the machine whose IP address corresponded to the domain name *www.omg.org*. See *http://www.omg.org*.	
ilu	The Inter-Language Unification (ILU) system is a multilingual object interface system. The object interfaces provided by ILU hide implementation distinctions between different languages, different address spaces, and different operating system types. ILU can be used to build multilingual object-oriented libraries ("class libraries") with well-specified, language-independent interfaces. It also can be used to implement distributed systems. See *ftp://parcftp.parc.xerox.com/pub/ilu/ilu.html*.	
imap	The IMAP URL scheme is used to designate IMAP servers, mailboxes, messages, MIME bodies [MIME], and search programs on Internet hosts accessible using the IMAP protocol.	2192
IOR	CORBA interoperable object reference. See *http://www.omg.org*.	
irc	The irc URL scheme is used to refer to either Internet Relay Chat (IRC) servers or individual entities (channels or people) on IRC servers. See *http://www.w3.org/Addressing/draft-mirashi-url-irc-01.txt*.	
isbn	Proposed scheme for ISBN book references. See *http://lists.w3.org/Archives/Public/www-talk/1991NovDec/0008.html*.	
java	Identifies Java classes.	
javascript	The Netscape browser processes javascript URLs, evaluates the expression after the colon (:), if there is one, and loads a page containing the string value of the expression, unless it is undefined.	
jdbc	Used in the Java SQL API.	
ldap	Allows Internet clients direct access to the LDAP protocol.	2255
lid	The Local Identifier (lid:) scheme. See *draft-blackketter-lid-00*.	
lifn	A Location-Independent File Name (LIFN) for the Bulk File Distribution distributed storage system developed at UTK.	
livescript	Old name for JavaScript.	
lrq	See h323.	
mailto	The mailto URL scheme is used to designate the Internet mailing address of an individual or service.	2368
mailserver	Old proposal from 1994–1995 to let an entire message be encoded in a URL, so that (for example) the URL can automatically send email to a mail server for subscribing to a mailing list.	

Table A-1. URI schemes from the W3C registry (continued)

Scheme	Description	RFCs
md5	MD5 is a cryptographic checksum.	
mid	The mid scheme uses (a part of) the message-id of an email message to refer to a specific message.	2392 2111
mocha	See javascript.	
modem	The modem scheme describes a connection to a terminal that can handle incoming data calls.	2806
mms, mmst, mmsu	Scheme for Microsoft Media Server (MMS) to stream Active Streaming Format (ASF) files. To force UDP transport, use the mmsu scheme. To force TCP transport, use mmst.	
news	The news URL scheme is used to refer to either news groups or individual articles of USENET news. A news URL takes one of two forms: *news:<newsgroup-name>* or *news:<message-id>*.	1738 1036
nfs	Used to refer to files and directories on NFS servers.	2224
nntp	An alternative method of referencing news articles, useful for specifying news articles from NNTP servers. An nntp URL looks like: *nntp://<host>:<port>/<newsgroup-name>/<article-num>* Note that while nntp URLs specify a unique location for the article resource, most NNTP servers currently on the Internet are configured to allow access only from local clients, and thus nntp URLs do not designate globally accessible resources. Hence, the news form of URL is preferred as a way of identifying news articles.	1738 977
opaquelocktoken	A WebDAV lock token, represented as a URI, that identifies a particular lock. A lock token is returned by every successful LOCK operation in the lockdiscovery property in the response body and also can be found through lock discovery on a resource. See RFC 2518.	
path	The path scheme defines a uniformly hierarchical namespace where a path URN is a sequence of components and an optional opaque string. See *http://www.hypernews.org/~liberte/www/path.html*.	
phone	Used in "URLs for Telephony"; replaced with tel: in RFC 2806.	
pop	The POP URL designates a POP email server, and optionally a port number, authentication mechanism, authentication ID, and/or authorization ID.	2384
pnm	Real Networks's streaming protocol.	
pop3	The POP3 URL scheme allows a URL to specify a POP3 server, allowing other protocols to use a general "URL to be used for mail access" in place of an explicit reference to POP3. Defined in expired *draft-earhart-url-pop3-00.txt*.	
printer	Abstract URLs for use with the Service Location standard. See *draft-ietf-srvloc-printer-scheme-02.txt*.	
prospero	Names resources to be accessed via the Prospero Directory Service.	1738
res	Microsoft scheme that specifies a resource to be obtained from a module. Consists of a string or numerical resource type, and a string or numerical ID.	
rtsp	Real-time streaming protocol that is the basis for Real Networks's modern streaming control protocols.	2326
rvp	URLs for the RVP rendezvous protocol, used to notify the arrival of users on a computer network. See *draft-calsyn-rvp-01*.	

Table A-1. URI schemes from the W3C registry (continued)

Scheme	Description	RFCs
rwhois	RWhois is an Internet directory access protocol, defined in RFC 1714 and RFC 2167. The RWhois URL gives clients direct access to rwhois. See *http://www.rwhois.net/rwhois/docs/*.	
rx	An architecture to allow remote graphical applications to display data inside web pages. See *http://www.w3.org/People/danield/papers/mobgui/*.	
sdp	Session Description Protocol (SDP) URLs. See RFC 2327.	
service	The service scheme is used to provide access information for arbitrary network services. These URLs provide an extensible framework for client-based network software to obtain configuration information required to make use of network services.	2609
sip	The sip* family of schemes are used to establish multimedia conferences using the Session Initiation Protocol (SIP).	2543
shttp	S-HTTP is a superset of HTTP designed to secure HTTP connections and provide a wide variety of mechanisms to provide for confidentiality, authentication, and integrity. It has not been widely deployed, and it has mostly been supplanted with HTTPS SSL-encrypted HTTP. See *http://www.homeport.org/~adam/shttp.html*.	
snews	SSL-encrypted news.	
STANF	Old proposal for stable network filenames. Related to URNs. See *http://web3.w3.org/Addressing/#STANF*.	
t120	See h323.	
tel	URL to place a call using the telephone network.	2806
telephone	Used in previous drafts of tel.	
telnet	Designates interactive services that may be accessed by the Telnet protocol. A telnet URL takes the form: *telnet://<user>:<password>@<host>:<port>/*	1738
tip	Supports TIP atomic Internet transactions.	2371 2372
tn3270	Reserved, as per *ftp://ftp.isi.edu/in-notes/iana/assignments/url-schemes*.	
tv	The TV URL names a particular television broadcast channel.	2838
uuid	Universally unique identifiers (UUIDs) contain no information about location. They also are known as globally unique identifiers (GUIDs). They are persistent over time, like URNs, and consist of a 128-bit unique ID. UUID URIs are useful in situations where a unique identifier is required that cannot or should not be tied to a particular physical root namespace (such as a DNS name). See *draft-kindel-uuid-uri-00.txt*.	
urn	Persistent, location-independent, URNs.	2141
vemmi	Allows versatile multimedia interface (VEMMI) client software and VEMMI terminals to connect to VEMMI-compliant services. VEMMI is an international standard for online multimedia services.	2122
videotex	Allows videotex client software or terminals to connect to videotex services compliant with the ITU-T and ETSI videotex standards. See *http://www.ics.uci.edu/pub/ietf/uri/draft-mavrakis-videotex-url-spec-01.txt*.	

Table A-1. URI schemes from the W3C registry (continued)

Scheme	Description	RFCs
view-source	Netscape Navigator source viewers. These view-source URLs display HTML that was generated with JavaScript.	
wais	The wide area information service—an early form of search engine.	1738
whois++	URLs for the WHOIS++ simple Internet directory protocol. See *http://martinh.net/wip/whois-url.txt*.	1835
whodp	The Widely Hosted Object Data Protocol (WhoDP) exists to communicate the current location and state of large numbers of dynamic, relocatable objects. A WhoDP program "subscribes" to locate and receive information about an object and "publishes" to control the location and visible state of an object. See *draft-mohr-whodp-00.txt*.	
z39.50r, z39.50s	Z39.50 session and retrieval URLs. Z39.50 is an information retrieval protocol that does not fit neatly into a retrieval model designed primarily around the stateless fetch of data. Instead, it models a general user inquiry as a session-oriented, multi-step task, any step of which may be suspended temporarily while the server requests additional parameters from the client before continuing.	2056

APPENDIX B
HTTP Status Codes

This appendix is a quick reference of HTTP status codes and their meanings.

Status Code Classifications

HTTP status codes are segmented into five classes, shown in Table B-1.

Table B-1. Status code classifications

Overall range	Defined range	Category
100–199	100–101	Informational
200–299	200–206	Successful
300–399	300–305	Redirection
400–499	400–415	Client error
500–599	500–505	Server error

Status Codes

Table B-2 is a quick reference for all the status codes defined in the HTTP/1.1 specification, providing a brief summary of each. "Status Codes" in Chapter 3 goes into more detailed descriptions of these status codes and their uses.

Table B-2. Status codes

Status code	Reason phrase	Meaning
100	Continue	An initial part of the request was received, and the client should continue.
101	Switching Protocols	The server is changing protocols, as specified by the client, to one listed in the Upgrade header.
200	OK	The request is okay.
201	Created	The resource was created (for requests that create server objects).

Table B-2. Status codes (continued)

Status code	Reason phrase	Meaning
202	Accepted	The request was accepted, but the server has not yet performed any action with it.
203	Non-Authoritative Information	The transaction was okay, except the information contained in the entity headers was not from the origin server, but from a copy of the resource.
204	No Content	The response message contains headers and a status line, but no entity body.
205	Reset Content	Another code primarily for browsers; basically means that the browser should clear any HTML form elements on the current page.
206	Partial Content	A partial request was successful.
300	Multiple Choices	A client has requested a URL that actually refers to multiple resources. This code is returned along with a list of options; the user can then select which one he wants.
301	Moved Permanently	The requested URL has been moved. The response should contain a Location URL indicating where the resource now resides.
302	Found	Like the 301 status code, but the move is temporary. The client should use the URL given in the Location header to locate the resource temporarily.
303	See Other	Tells the client that the resource should be fetched using a different URL. This new URL is in the Location header of the response message.
304	Not Modified	Clients can make their requests conditional by the request headers they include. This code indicates that the resource has not changed.
305	Use Proxy	The resource must be accessed through a proxy, the location of the proxy is given in the Location header.
306	(Unused)	This status code currently is not used.
307	Temporary Redirect	Like the 301 status code; however, the client should use the URL given in the Location header to locate the resource temporarily.
400	Bad Request	Tells the client that it sent a malformed request.
401	Unauthorized	Returned along with appropriate headers that ask the client to authenticate itself before it can gain access to the resource.
402	Payment Required	Currently this status code is not used, but it has been set aside for future use.
403	Forbidden	The request was refused by the server.
404	Not Found	The server cannot find the requested URL.
405	Method Not Allowed	A request was made with a method that is not supported for the requested URL. The Allow header should be included in the response to tell the client what methods are allowed on the requested resource.
406	Not Acceptable	Clients can specify parameters about what types of entities they are willing to accept. This code is used when the server has no resource matching the URL that is acceptable for the client.
407	Proxy Authentication Required	Like the 401 status code, but used for proxy servers that require authentication for a resource.

Table B-2. Status codes (continued)

Status code	Reason phrase	Meaning
408	Request Timeout	If a client takes too long to complete its request, a server can send back this status code and close down the connection.
409	Conflict	The request is causing some conflict on a resource.
410	Gone	Like the 404 status code, except that the server once held the resource.
411	Length Required	Servers use this code when they require a Content-Length header in the request message. The server will not accept requests for the resource without the Content-Length header.
412	Precondition Failed	If a client makes a conditional request and one of the conditions fails, this response code is returned.
413	Request Entity Too Large	The client sent an entity body that is larger than the server can or wants to process.
414	Request URI Too Long	The client sent a request with a request URL that is larger than what the server can or wants to process.
415	Unsupported Media Type	The client sent an entity of a content type that the server does not understand or support.
416	Requested Range Not Satisfiable	The request message requested a range of a given resource, and that range either was invalid or could not be met.
417	Expectation Failed	The request contained an expectation in the Expect request header that could not be satisfied by the server.
500	Internal Server Error	The server encountered an error that prevented it from servicing the request.
501	Not Implemented	The client made a request that is beyond the server's capabilities.
502	Bad Gateway	A server acting as a proxy or gateway encountered a bogus response from the next link in the request response chain.
503	Service Unavailable	The server cannot currently service the request but will be able to in the future.
504	Gateway Timeout	Similar to the 408 status code, except that the response is coming from a gateway or proxy that has timed out waiting for a response to its request from another server.
505	HTTP Version Not Supported	The server received a request in a version of the protocol that it can't or won't support.

APPENDIX C
HTTP Header Reference

It's almost amusing to remember that the first version of HTTP, 0.9, had no headers. While this certainly had its down sides, its fun to marvel in its simplistic elegance.

Well, back to reality. Today there are a horde of HTTP headers, many part of the specification and still others that are extensions to it. This appendix provides some background on these official and extension headers. It also acts as an index for the various headers in this book, pointing out where their concepts and features are discussed in the running text. Most of these headers are simple up-front; it's the interactions with each other and other features of HTTP where things get hairy. This appendix provides a bit of background for the headers listed and directs you to the sections of the book where they are discussed at length.

The headers listed in this appendix are drawn from the HTTP specifications, related documents, and our own experience poking around with HTTP messages and the various servers and clients on the Internet.

This list is far from exhaustive. There are many other extension headers floating around on the Web, not to mention those potentially used in private intranets. Nonetheless, we have attempted to make this list as complete as possible. See RFC 2616 for the current version of the HTTP/1.1 specification and a list of official headers and their specification descriptions.

Accept

The Accept header is used by clients to let servers know what media types are *acceptable*. The value of the Accept header field is a list of media types that the client can use. For instance, your web browser cannot display every type of multimedia object on the Web. By including an Accept header in your requests, your browser can save you from downloading a video or other type of object that you can't use.

The Accept header field also may include a list of quality values (q values) that tell the server which media type is preferred, in case the server has multiple versions of the media type. See Chapter 17 for a complete discussion of content negotiation and q values.

Type Request header

Notes "*" is a special value that is used to wildcard media types. For example, "*/*" represents all types, and "image/*" represents all image types.

Examples Accept: text/*, image/*
Accept: text/*, image/gif, image/jpeg;q=1

Accept-Charset

The Accept-Charset header is used by clients to tell servers what character sets are acceptable or preferred. The value of this request header is a list of character sets and possibly quality values for the listed character sets. The quality values let the server know which character set is preferred, in case the server has the document in multiple acceptable character sets. See Chapter 17 for a complete discussion of content negotiation and q values.

Type Request header

Notes As with the Accept header, "*" is a special character. If present, it represents all character sets, except those that also are mentioned explicitly in the value. If it's not present, any charset not in the value field has a default q value of zero, with the exception of the iso-latin-1 charset, which gets a default of 1.

Basic Syntax Accept-Charset: 1# ((charset | "*") [";" "q" "=" qvalue])

Example Accept-Charset: iso-latin-1

Accept-Encoding

The Accept-Encoding header is used by clients to tell servers what encodings are acceptable. If the content the server is holding is encoded (perhaps compressed), this request header lets the server know whether the client will accept it. Chapter 17 contains a complete description of the Accept-Encoding header.

Type Request header

Basic Syntax Accept-Encoding: 1# ((content-coding | "*") [";" "q" "=" qvalue])

Examples* Accept-Encoding:
Accept-Encoding: gzip
Accept-Encoding: compress;q=0.5, gzip;q=1

* The empty Accept-Encoding example is not a typo. It refers to the identity encoding—that is, the unencoded content. If the Accept-Encoding header is present and empty, only the unencoded content is acceptable.

Accept-Language

The Accept-Language request header functions like the other Accept headers, allowing clients to inform the server about what languages (e.g., the natural language for content) are acceptable or preferred. Chapter 17 contains a complete description of the Accept-Language header.

Type Request header

Basic Syntax Accept-Language: 1# (language-range [";" "q" "=" qvalue])
language-range = ((1*8ALPHA * ("-" 1*8ALPHA)) | "*")

Examples Accept-Language: en
Accept-Language: en;q=0.7, en-gb;q=0.5

Accept-Ranges

The Accept-Ranges header differs from the other Accept headers—it is a response header used by servers to tell clients whether they accept requests for ranges of a resource. The value of this header tells what type of ranges, if any, the server accepts for a given resource.

A client can attempt to make a range request on a resource without having received this header. If the server does not support range requests for that resource, it can respond with an appropriate status code* and the Accept-Ranges value "none". Servers might want to send the "none" value for normal requests to discourage clients from making range requests in the future.

Chapter 17 contains a complete description of the Accept-Ranges header.

Type Response header

Basic Syntax Accept-Ranges: 1# range-unit | none

Examples Accept-Ranges: none
Accept-Ranges: bytes

Age

The Age header tells the receiver how old a response is. It is the sender's best guess as to how long ago the response was generated by or revalidated with the origin server. The value of the header is the sender's guess, a delta in seconds. See Chapter 7 for more on the Age header.

Type Response header

Notes HTTP/1.1 caches must include an Age header in every response they send.

* For example, status code 416 (see "400–499: Client Error Status Codes" in Chapter 3).

Basic Syntax Age: delta-seconds

Example Age: 60

Allow

The Allow header is used to inform clients what HTTP methods are supported on a particular resource.

Type Response header

Notes An HTTP/1.1 server sending a 405 Method Not Allowed response must include an Allow header.*

Basic Syntax Allow: #Method

Example Allow: GET, HEAD

Authorization

The Authorization header is sent by a client to authenticate itself with a server. A client will include this header in its request after receiving a 401 Authentication Required response from a server. The value of this header depends on the authentication scheme in use. See Chapter 14 for a detailed discussion of the Authorization header.

Type Response header

Basic Syntax Authorization: authentication-scheme #authentication-param

Example Authorization: Basic YnJpYW4tdG90dHk6T3ch

Cache-Control

The Cache-Control header is used to pass information about how an object can be cached. This header is one of the more complex headers introduced in HTTP/1.1. Its value is a caching directive, giving caches special instructions about an object's cacheability.

In Chapter 7, we discuss caching in general as well as the specific details about this header.

Type General header

Example Cache-Control: no-cache

* See "Status Codes" in Chapter 3 for more on the 405 status code.

Client-ip

The Client-ip header is an extension header used by some older clients and some proxies to transmit the IP address of the machine on which the client is running.

Type Extension request header

Notes Implementors should be aware that the information provided in the value of this header is not secure.

Basic Syntax Client-ip: ip-address

Example Client-ip: 209.1.33.49

Connection

The Connection header is a somewhat overloaded header that can lead to a bit of confusion. This header was used in HTTP/1.0 clients that were extended with keep-alive connections for control information.* In HTTP/1.1, the older semantics are mostly recognized, but the header has taken on a new function.

In HTTP/1.1, the Connection header's value is a list of tokens that correspond to header names. Applications receiving an HTTP/1.1 message with a Connection header are supposed to parse the list and remove any of the headers in the message that are in the Connection header list. This is mainly for proxies, allowing a server or other proxy to specify hop-by-hop headers that should not be passed along.

One special token value is "close". This token means that the connection is going to be closed after the response is completed. HTTP/1.1 applications that do not support persistent connections need to insert the Connection header with the "close" token in all requests and responses.

Type General header

Notes While RFC 2616 does not specifically mention keep-alive as a connection token, some browsers (including those sending HTTP/1.1 as their versions) use it in making requests.

Basic Syntax Connection: 1# (connection-token)

Examples Connection: close

* See Chapter 4 for more on keep-alive and persistent connections.

Content-Base

The Content-Base header provides a way for a server to specify a base URL for resolving URLs found in the entity body of a response.* The value of the Content-Base header is an absolute URL that can be used to resolve relative URLs found inside the entity.

Type Entity header

Notes This header is not defined in RFC 2616; it was previously defined in RFC 2068, an earlier draft of the HTTP/1.1 specification, and has since been removed from the official specification.

Basic Syntax Content-Base: absoluteURL

Example Content-Base: http://www.joes-hardware.com/

Content-Encoding

The Content-Encoding header is used to specify whether any encodings have been performed on the object. By encoding the content, a server can compress it before sending the response. The value of the Content-Encoding header tells the client what type or types of encoding have been performed on the object. With that information, the client can then decode the message.

Sometimes more than one encoding is applied to an entity, in which case the encodings must be listed in the order in which they were performed.

Type Entity header

Basic Syntax Content-Encoding: 1# content-coding

Examples Content-Encoding: gzip
Content-Encoding: compress, gzip

Content-Language

The Content-Language header tells the client the natural language that should be understood in order to understand the object. For instance, a document written in French would have a Content-Language value indicating French. If this header is not present in the response, the object is intended for all audiences. Multiple languages in the header's value indicate that the object is suitable for audiences of each language listed.

One caveat about this header is that the header's value may just represent the natural language of the intended audience of this object, not all or any of the languages contained

* See Chapter 2 for more on base URLs.

in the object. Also, this header is not limited to text or written data objects; images, video, and other media types can be tagged with their intended audiences' natural languages.

Type Entity header

Basic Syntax Content-Language: 1# language-tag

Examples Content-Language: en
Content-Language: en, fr

Content-Length

The Content-Length header gives the length or size of the entity body. If the header is in a response message to a HEAD HTTP request, the value of the header indicates the size that the entity body would have been had it been sent.

Type Entity header

Basic Syntax Content-Length: 1*DIGIT

Example Content-Length: 2417

Content-Location

The Content-Location header is included in an HTTP message to give the URL corresponding to the entity in the message. For objects that may have multiple URLs, a response message can include a Content-Location header indicating the URL of the object used to generate the response. The Content-Location can be different from the requested URL. This generally is used by servers that are directing or redirecting a client to a new URL.

If the URL is relative, it should be interpreted relative to the Content-Base header. If the Content-Base header is not present, the URL used in the request should be used.

Type Entity header

Basic Syntax Content-Location: (absoluteURL | relativeURL)

Example Content-Location: http://www.joes-hardware.com/index.html

Content-MD5

The Content-MD5 header is used by servers to provide a message-integrity check for the message body. Only an origin server or requesting client should insert a Content-MD5

header in the message. The value of the header is an MD5 digest* of the (potentially encoded) message body.

The value of this header allows for an end-to-end check on the data, useful for detecting unintentional modifications to the data in transit. It is not intended to be used for security purposes.

RFC 1864 defines this header in more detail.

Type Entity header

Notes The MD5 digest value is a base-64 (see Appendix E) or 128-bit MD5 digest, as defined in RFC 1864.

Basic Syntax Content-MD5: md5-digest

Example Content-MD5: Q2h1Y2sgSW51ZwDIAXR5IQ==

Content-Range

The Content-Range header is sent as the result of a request that transmitted a range of a document. It provides the location (range) within the original entity that this entity represents. It also gives the length of the entire entity.

If an "*" is present in the value instead of the length of the entire entity, this means that the length was not known when the response was sent.

See Chapter 15 for more on the Content-Range header.

Type Entity header

Notes Servers responding with the 206 Partial Content response code must not include a Content-Range header with an "*" as the length.

Example Content-Range: bytes 500-999 / 5400

Content-Type

The Content-Type header tells the media type of the object in the message.

Type Entity header

Basic Syntax Content-Type: media-type

Example Content-Type: text/html; charset=iso-latin-1

* The MD5 digest is defined in RFC 1864.

Cookie

The Cookie header is an extension header used for client identification and tracking. Chapter 11 talks about the Cookie header and its use in detail (also see "Set-Cookie").

Type Extension request header

Example Cookie: ink=IUOK164y59BC708378908CFF89OE5573998A115

Cookie2

The Cookie2 header is an extension header used for client identification and tracking. Cookie2 is used to identify what version of cookies a requestor understands. It is defined in greater detail in RFC 2965.

Chapter 11 talks about the Cookie2 header and its use in detail.

Type Extension request header

Example Cookie2: $version="1"

Date

The Date header gives the date and time at which the message was created. This header is required in servers' responses because the time and date at which the server believes the message was created can be used by caches in evaluating the freshness of a response. For clients, this header is completely optional, although it's good form to include it.

Type General header

Basic Syntax Date: HTTP-date

Examples Date: Tue, 3 Oct 1997 02:15:31 GMT

HTTP has a few specific date formats. This one is defined in RFC 822 and is the preferred format for HTTP/1.1 messages. However, in earlier specifications of HTTP, the date format was not spelled out as well, so server and client implementors have used other formats, which need to be supported for the sake of legacy. You will run into date formats like the one specified in RFC 850, as well as dates in the format produced by the *asctime()* system call. Here they are for the date represented above:

```
Date: Tuesday, 03-Oct-97 02:15:31 GMT    RFC 850 format
Date: Tue Oct 3 02:15:31 1997            asctime() format
```

The *asctime()* format is looked down on because it is in local time and it does not specify its time zone (e.g., GMT). In general, the date header should be in GMT; however, robust applications should handle dates that either do not specify the time zone or include Date values in non-GMT time.

ETag

The ETag header provides the *entity tag* for the entity contained in the message. An entity tag is basically a way of identifying a resource.

Entity tags and their relationship to resources are discussed in detail in Chapter 15.

Type Entity header

Basic Syntax ETag: entity-tag

Examples ETag: "11e92a-457b-31345aa"
ETag: W/"11e92a-457b-3134b5aa"

Expect

The Expect header is used by clients to let servers know that they expect certain behavior. This header currently is closely tied to the response code 100 Continue (see "100–199: Informational Status Codes" in Chapter 3).

If a server does not understand the Expect header's value, it should respond with a status code of 417 Expectation Failed.

Type Request header

Basic Syntax Expect: 1# ("100-continue" | expectation-extension)

Example Expect: 100-continue

Expires

The Expires header gives a date and time at which the response is no longer valid. This allows clients such as your browser to cache a copy and not have to ask the server if it is still valid until after this time has expired.

Chapter 7 discusses how the Expires header is used—in particular, how it relates to caches and having to revalidate responses with the origin server.

Type Entity header

Basic Syntax Expires: HTTP-date

Example Expires: Thu, 03 Oct 1997 17:15:00 GMT

From

The From header says who the request is coming from. The format is just a valid Internet email address (specified in RFC 1123) for the user of the client.

There are potential privacy issues with using/populating this header. Client implementors should be careful to inform their users and give them a choice before including this header in a request message. Given the potential for abuse by people collecting email addresses for unsolicited mail messages, woe to the implementor who broadcasts this header unannounced and has to answer to angry users.

Type Request header

Basic Syntax From: mailbox

Example From: slurp@inktomi.com

Host

The Host header is used by clients to provide the server with the Internet hostname and port number of the machine from which the client wants to make a request. The hostname and port are those from the URL the client was requesting.

The Host header allows servers to differentiate different relative URLs based on the hostname, giving the server the ability to host several different hostnames on the same machine (i.e., the same IP address).

Type Request header

Notes HTTP/1.1 clients must include a Host header in all requests. All HTTP/1.1 servers must respond with the 400 Bad Request status code to HTTP/1.1 clients that do not provide a Host header.

Basic Syntax Host: host [":" port]

Example Host: www.hotbot.com:80

If-Modified-Since

The If-Modified-Since request header is used to make conditional requests. A client can use the GET method to request a resource from a server, having the response hinge on whether the resource has been modified since the client last requested it.

If the object has not been modified, the server will respond with a 304 Not Modified response, instead of with the resource. If the object has been modified, the server will respond as if it was a non-conditional GET request. Chapter 7 discusses conditional requests in detail.

Type Request header

Basic Syntax If-Modified-Since: HTTP-date

Example If-Modified-Since: Thu, 03 Oct 1997 17:15:00 GMT

If-Match

Like the If-Modified-Since header, the If-Match header can be used to make a request conditional. Instead of a date, the If-Match request uses an entity tag. The server compares the entity tag in the If-Match header with the current entity tag of the resource and returns the object if the tags match.

The server should use the If-Match value of "*" to match any entity tag it has for a resource; "*" will always match, unless the server no longer has the resource.

This header is useful for updating resources that a client or cache already has. The resource is returned only if it has changed—that is, if the previously requested object's entity tag does not match the entity tag of the current version on the server. Chapter 7 discusses conditional requests in detail.

Type Request header

Basic Syntax If-Match: ("*" | 1# entity-tag)

Example If-Match: "11e92a-457b-31345aa"

If-None-Match

The If-None-Match header, like all the If headers, can be used to make a request conditional. The client supplies the server with a list of entity tags, and the server compares those tags against the entity tags it has for the resource, returning the resource only if none match.

This allows a cache to update resources only if they have changed. Using the If-None-Match header, a cache can use a single request to both invalidate the entities it has and receive the new entity in the response. Chapter 7 discusses conditional requests in detail.

Type Request header

Basic Syntax If-None-Match: ("*" | 1# entity-tag)

Example If-None-Match: "11e92a-457b-31345aa"

If-Range

The If-Range header, like all the If headers, can be used to make a request conditional. It is used when an application has a copy of a range of a resource, to revalidate the range or get the complete resource if the range is no longer valid. Chapter 7 discusses conditional requests in detail.

Type Request header

Basic Syntax If-Range: (HTTP-date | entity-tag)

Examples If-Range: Tue, 3 Oct 1997 02:15:31 GMT
If-Range: "11e92a-457b-3134b5aa"

If-Unmodified-Since

The If-Unmodified-Since header is the twin of the If-Modified-Since header. Including it in a request makes the request conditional. The server should look at the date value of the header and return the object only if it has not been modified since the date provided. Chapter 7 discusses conditional requests in detail.

Type Request header

Basic Syntax If-Unmodified-Since: HTTP-date

Example If-Unmodified-Since: Thu, 03 Oct 1997 17:15:00 GMT

Last-Modified

The Last-Modified header tries to provide information about the last time this entity was changed. This could mean a lot of things. For example, resources typically are files on a server, so the Last-Modified value could be the last-modified time provided by the server's filesystem. On the other hand, for dynamically created resources such as those created by scripts, the Last-Modified value could be the time the response was created.

Servers need to be careful that the Last-Modified time is not in the future. HTTP/1.1 servers should reset the Last-Modified time if it is later than the value that would be sent in the Date header.

Type Entity header

Basic Syntax Last-Modified: HTTP-date

Example Last-Modified: Thu, 03 Oct 1997 17:15:00 GMT

Location

The Location header is used by servers to direct clients to the location of a resource that either was moved since the client last requested it or was created in response to the request.

Type Response header

Basic Syntax Location: absoluteURL

Example Location: http://www.hotbot.com

Max-Forwards

This header is used only with the TRACE method, to limit the number of proxies or other intermediaries that a request goes through. Its value is an integer. Each application that receives a TRACE request with this header should decrement the value before it forwards the request along.

If the value is zero when the application receives the request, it should send back a 200 OK response to the request, with an entity body containing the original request. If the Max-Forwards header is missing from a TRACE request, assume that there is no maximum number of forwards.

For other HTTP methods, this header should be ignored. See "Methods" in Chapter 3 for more on the TRACE method.

Type Request header

Basic Syntax Max-Forwards: 1*DIGIT

Example Max-Forwards: 5

MIME-Version

MIME is HTTP's cousin. While they are radically different, some HTTP servers do construct messages that are valid under the MIME specification. When this is the case, the MIME-Version header can be supplied by the server.

This header has never been part of the official specification, although it is mentioned in the HTTP/1.0 specification. Many older servers send messages with this header, however, those messages often are not valid MIME messages, making this header both confusing and impossible to trust.

Type Extension general header

Basic Syntax MIME-Version: DIGIT "." DIGIT

Example MIME-Version: 1.0

Pragma

The Pragma header is used to pass directions along with the message. These directions could be almost anything, but often they are used to control caching behavior. Proxies and gateways must not remove the Pragma header, because it could be intended for all applications that receive the message.

The most common form of Pragma, Pragma: no-cache, is a request header that forces caches to request or revalidate the document from the origin server even when a fresh copy is available in the cache. It is sent by browsers when users click on the Reload/Refresh button. Many servers send Pragma: no-cache as a response header (as an equivalent to

Cache-Control: no-cache), but despite its common use, this behavior is technically undefinded. Not all applications support Pragma response headers.

Chapter 7 discusses the Pragma header and how it is used by HTTP/1.0 applications to control caches.

Type Request header

Basic Syntax Pragma: 1# pragma-directive*

Example Pragma: no-cache

Proxy-Authenticate

The Proxy-Authenticate header functions like the WWW-Authenticate header. It is used by proxies to challenge an application sending a request to authenticate itself. The full details of this challenge/response, and other security mechanisms of HTTP, are discussed in detail in Chapter 14.

If an HTTP/1.1 proxy server is sending a 407 Proxy Authentication Required response, it must include the Proxy-Authenticate header.

Proxies and gateways must be careful in interpreting all the Proxy headers. They generally are hop-by-hop headers, applying only to the current connection. For instance, the Proxy-Authenticate header requests authentication for the current connection.

Type Response header

Basic Syntax Proxy-Authenticate: challenge

Example Proxy-Authenticate: Basic realm="Super Secret Corporate Financial Documents"

Proxy-Authorization

The Proxy-Authorization header functions like the Authorization header. It is used by client applications to respond to Proxy-Authenticate challenges. See Chapter 14 for more on how the challenge/response security mechanism works.

Type Request header

Basic Syntax Proxy-Authorization: credentials

Example Proxy-Authorization: Basic YnJpYW4tdG90dHk6T3ch

* The only specification-defined Pragma directive is "no-cache"; however, you may run into other Pragma headers that have been defined as extensions to the specification.

Proxy-Connection

The Proxy-Connection header was meant to have similar semantics to the HTTP/1.0 Connection header. It was to be used between clients and proxies to specify options about the connections (chiefly keep-alive connections).* It is not a standard header and is viewed as an ad hoc header by the standards committee. However, it is widely used by browsers and proxies.

Browser implementors created the Proxy-Connection header to solve the problem of a client sending an HTTP/1.0 Connection header that gets blindly forwarded by a dumb proxy. A server receiving the blindly forwarded Connection header could confuse the capabilities of the client connection with those of the proxy connection.

The Proxy-Connection header is sent instead of the Connection header when the client knows that it is going through a proxy. Because servers don't recognize the Proxy-Connection header, they ignore it, allowing dumb proxies that blindly forward the header to do so without causing harm.

The problem with this solution occurs if there is more than one proxy in the path of the client to the server. If the first one blindly forwards the header to the second, which understands it, the second proxy can suffer from the same confusion the server did with the Connection header.

This is the problem that the HTTP working group had with this solution—they saw it as a hack that solved the case of a single proxy, but not the bigger problem. Nonetheless, it does handle some of the more common cases, and because older versions of both Netscape Navigator and Microsoft Internet Explorer implement it, proxy implementors need to deal with it. See Chapter 4 for more information.

Type General header

Basic Syntax Proxy-Connection: 1# (connection-token)

Example Proxy-Connection: close

Public

The Public header allows a server to tell a client what methods it supports. These methods can be used in future requests by the client. Proxies need to be careful when they receive a response from a server with the Public header. The header indicates the capabilities of the server, not the proxy, so the proxy needs to either edit the list of methods in the header or remove the header before it sends the response to the client.

Type Response header

* See Chapter 4 for more on keep-alive and persistent connections.

Notes This header is not defined in RFC 2616. It was previously defined in RFC 2068, an earlier draft of the HTTP/1.1 specification, but it has since been removed from the official specification.

Basic Syntax Public: 1# HTTP-method

Example Public: OPTIONS, GET, HEAD, TRACE, POST

Range

The Range header is used in requests for parts or ranges of an entity. Its value indicates the range of the entity that is included in the message.

Requests for ranges of a document allow for more efficient requests of large objects (by requesting them in segments) or for recovery from failed transfers (allowing a client to request the range of the resource that did not make it). Range requests and the headers that make the requests possible are discussed in detail in Chapter 15.

Type Entity header

Example Range: bytes=500-1500

Referer

The Referer header is inserted into client requests to let the server know where the client got the URL from. This is a voluntary effort, for the server's benefit; it allows the server to better log the requests or perform other tasks. The misspelling of "Referer" hearkens back to the early days of HTTP, to the frustration of English-speaking copyeditors throughout the world.

What your browser does is fairly simple. If you get home page A and click on a link to go to home page B, your browser will insert a Referer header in the request with value A. Referer headers are inserted by your browser only when you click on links; requests for URLs you type in yourself will not contain a Referer header.

Because some pages are private, there are some privacy concerns with this header. While some of this is unwarranted paranoia, this header does allow web servers and their administrators to see where you came from, potentially allowing them to better track your surfing. As a result, the HTTP/1.1 specification recommends that application writers allow the user to decide whether this header is transmitted.

Type Request header

Basic Syntax Referer: (absoluteURL | relativeURL)

Example Referer: http://www.inktomi.com/index.html

Retry-After

Servers can use the Retry-After header to tell a client when to retry its request for a resource. It is used with the 503 Service Unavailable status code to give the client a specific date and time (or number of seconds) at which it should retry its request.

A server can also use this header when it is redirecting clients to resources, giving the client a time to wait before making a request on the resource to which it is redirected.* This can be very useful to servers that are creating dynamic resources, allowing the server to redirect the client to the newly created resource but giving time for the resource to be created.

Type Response header

Basic Syntax Retry-After: (HTTP-date | delta-seconds)

Examples
Retry-After: Tue, 3 Oct 1997 02:15:31 GMT
Retry-After: 120

Server

The Server header is akin to the User-Agent header; it provides a way for servers to identify themselves to clients. Its value is the server name and an optional comment about the server.

Because the Server header identifies the server product and can contain additional comments about the product, its format is somewhat free-form. If you are writing software that depends on how a server identifies itself, you should experiment with the server software to see what it sends back, because these tokens vary from product to product and release to release.

As with the User-Agent header, don't be surprised if an older proxy or gateway inserts what amounts to a Via header in the Server header itself.

Type Response header

Basic Syntax Server: 1* (product | comment)

Examples
Server: Microsoft-Internet-Information-Server/1.0
Server: websitepro/1.1f (s/n wpo-07d0)
Server: apache/1.2b6 via proxy gateway CERN-HTTPD/3.0 libwww/2.13

Set-Cookie

The Set-Cookie header is the partner to the Cookie header; in Chapter 11, we discuss the use of this header in detail.

* See "Redirection status codes and reason phrases" in Chapter 3 for more on server redirect responses.

Type Extension response header

Basic Syntax Set-Cookie: command

Examples Set-Cookie: lastorder=00183; path=/orders
Set-Cookie: private_id=519; secure

Set-Cookie2

The Set-Cookie2 header is an extension of the Set-Cookie header; in Chapter 11, we discuss the use of this header in detail.

Type Extension response header

Basic Syntax Set-Cookie2: command

Examples Set-Cookie2: ID="29046"; Domain=".joes-hardware.com"
Set-Cookie2: color=blue

TE

The poorly named TE header functions like the Accept-Encoding header, but for transfer encodings (it could have been named Accept-Transfer-Encoding, but it wasn't). The TE header also can be used to indicate whether a client can handle headers in the trailer of a response that has been through the chunked encoding. See Chapter 15 for more on the TE header, chunked encoding, and trailers.

Type Request header

Notes If the value is empty, only the chunked transfer encoding is acceptable. The special token "trailers" indicates that trailer headers are acceptable in a chunked response.

Basic Syntax TE: # (transfer-codings)
transfer-codings= "trailers" | (transfer-extension [accept-params])

Examples TE:
TE: chunked

Trailer

The Trailer header is used to indicate which headers are present in the trailer of a message. Chapter 15 discusses chunked encodings and trailers in detail.

Type General header

Basic Syntax Trailer: 1#field-name

Example Trailer: Content-Length

Title

The Title header is a non-specification header that is supposed to give the title of the entity. This header was part of an early HTTP/1.0 extension and was used primarily for HTML pages, which have clear title markers that servers can use. Because many, if not most, media types on the Web do not have such an easy way to extract a title, this header has limited usefulness. As a result, it never made it into the official specification, though some older servers on the Net still send it faithfully.

Type Response header

Notes The Title header is not defined in RFC 2616. It was originally defined in the HTTP/1.0 draft definition (*http://www.w3.org/Protocols/HTTP/HTTP2.html*) but has since been removed from the official specification.

Basic Syntax Title: document-title

Example Title: CNN Interactive

Transfer-Encoding

If some encoding had to be performed to transfer the HTTP message body safely, the message will contain the Transfer-Encoding header. Its value is a list of the encodings that were performed on the message body. If multiple encodings were performed, they are listed in order.

The Transfer-Encoding header differs from the Content-Encoding header because the transfer encoding is an encoding that was performed by a server or other intermediary application to transfer the message.

Transfer encodings are discussed in Chapter 15.

Type General header

Basic Syntax Transfer-Encoding: 1# transfer-coding

Example Transfer-Encoding: chunked

UA-(CPU, Disp, OS, Color, Pixels)

These User-Agent headers are nonstandard and no longer common. They provide information about the client machine that could allow for better content selection by a server. For

instance, if a server knew that a user's machine had only an 8-bit color display, the server could select images that were optimized for that type of display.

With any header that gives information about the client that otherwise would be unavailable, there are some security concerns (see Chapter 14 for more information).

Type Extension request headers

Notes These headers are not defined in RFC 2616, and their use is frowned upon.

Basic Syntax "UA" "-" ("CPU" | "Disp" | "OS" | "Color" | "Pixels") ":" machine-value
machine-value = (cpu | screensize | os-name | display-color-depth)

Examples

UA-CPU: x86	*CPU of client's machine*
UA-Disp: 640, 480, 8	*Size and color depth of client's display*
UA-OS: Windows 95	*Operating system of client machine*
UA-Color: color8	*Color depth of client's display*
UA-Pixels: 640x480	*Size of client's display*

Upgrade

The Upgrade header provides the sender of a message with a means of broadcasting the desire to use another, perhaps completely different, protocol. For instance, an HTTP/1.1 client could send an HTTP/1.0 request to a server and include an Upgrade header with the value "HTTP/1.1", allowing the client to test the waters and see whether the server speaks HTTP/1.1.

If the server is capable, it can send an appropriate response letting the client know that it is okay to use the new protocol. This provides an efficient way to move to other protocols. Most servers currently are only HTTP/1.0-compliant, and this strategy allows a client to avoid confusing a server with too many HTTP/1.1 headers until it determines whether the server is indeed capable of speaking HTTP/1.1.

When a server sends a 101 Switching Protocols response, it must include this header.

Type General header

Basic Syntax Upgrade: 1# protocol

Example Upgrade: HTTP/2.0

User-Agent

The User-Agent header is used by client applications to identify themselves, much like the Server header for servers. Its value is the product name and possibly a comment describing the client application.

This header's format is somewhat free-form. Its value varies from client product to product and release to release. This header sometimes even contains information about the machine on which the client is running.

As with the Server header, don't be surprised if older proxy or gateway applications insert what amounts to a Via header in the User-Agent header itself.

Type; Request header

Basic Syntax User-Agent: 1* (product | comment)

Example User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)

Vary

The Vary header is used by servers to inform clients what headers from a client's request will be used in server-side negotiation.* Its value is a list of headers that the server looks at to determine what to send the client as a response.

An example of this would be a server that sends special HTML pages based on your web browser's features. A server sending these special pages for a URL would include a Vary header that indicated that it looked at the User-Agent header of the request to determine what to send as a response.

The Vary header also is used by caching proxies; see Chapter 7 for more on how the Vary header relates to cached HTTP responses.

Type Response header

Basic Syntax Vary: ("*" | 1# field-name)

Example Vary: User-Agent

Via

The Via header is used to trace messages as they pass through proxies and gateways. It is an informational header that can be used to see what applications are handling requests and responses.

When a message passes through an HTTP application on its way to a client or a server, that application can use the Via header to tag the message as having gone *via* it. This is an HTTP/1.1 header; many older applications insert a Via-like string in the User-Agent or Server headers of requests and responses.

If the message passes through multiple in-between applications, each one should tack on its Via string. The Via header must be inserted by HTTP/1.1 proxies and gateways.

* See Chapter 17 for more on content negotiation.

Type General header

Basic Syntax* Via: 1# (received-protocol received-by [comment])

Example Via: 1.1 joes-hardware.com (Joes-Server/1.0)

The above says that the message passed through the Joes Server Version 1.0 software running on the machine *joes-hardware.com*. Joe's Server was speaking HTTP 1.1. The Via header should be formatted like this:

```
HTTP-Version machine-hostname (Application-Name-Version)
```

Warning

The Warning header is used to give a little more information about what happened during a request. It provides the server with a way to send additional information that is not in the status code or reason phrase. Several warning codes are defined in the HTTP/1.1 specification:

101 Response Is Stale
: When a response message is known to be stale—for instance, if the origin server is unavailable for revalidation—this warning must be included.

111 Revalidation Failed
: If a cache attempts to revalidate a response with an origin server and the revalidation fails because the cache cannot reach the origin server, this warning must be included in the response to the client.

112 Disconnected Operation
: An informative warning; should be used if a cache's connectivity to the network is removed.

113 Heuristic Expiration
: Caches must include this warning if their freshness heuristic is greater than 24 hours and they are returning a response with an age greater than 24 hours.

199 Miscellaneous Warning
: Systems receiving this warning must not take any automated response; the message may and probably should contain a body with additional information for the user.

214 Transformation Applied
: Must be added by any intermediate application, such as a proxy, if the application performs any transformation that changes the content encoding of the response.

299 Miscellaneous Persistent Warning
: Systems receiving this warning must not take any automated reaction; the error may contain a body with more information for the user.

* See the HTTP/1.1 specification for the complete Via header syntax.

Type Response header

Basic Syntax Warning: 1# warning-value

Example Warning: 113

WWW-Authenticate

The WWW-Authenticate header is used in 401 Unauthorized responses to issue a challenge authentication scheme to the client. Chapter 14 discusses the WWW-Authenticate header and its use in HTTP's basic challenge/response authentication system.

Type Response header

Basic Syntax WWW-Authenticate: 1# challenge

Example WWW-Authenticate: Basic realm="Your Private Travel Profile"

X-Cache

The X headers are all extension headers. The X-Cache header is used by Squid to inform a client whether a resource is available.

Type Extension response header

Example X-Cache: HIT

X-Forwarded-For

This header is used by many proxy servers (e.g., Squid) to note whom a request has been forwarded for. Like the Client-ip header mentioned earlier, this request header notes the address from which the request originates.

Type Extension request header

Basic Syntax X-Forwarded-For: addr

Example X-Forwarded-For: 64.95.76.161

X-Pad

This header is used to overcome a bug related to response header length in some browsers; it pads the response message headers with extra bytes to work around the bug.

Type Extension general header

Basic Syntax X-Pad: pad-text

Example X-Pad: bogosity

X-Serial-Number

The X-Serial-Number header is an extension header. It was used by some older HTTP applications to insert the serial number of the licensed software in the HTTP message.

Its use has pretty much died out, but it is listed here as an example of the X headers that are out there.

Type Extension general header

Basic Syntax X-Serial-Number: serialno

Example X-Serial-Number: 010014056

APPENDIX D
MIME Types

MIME media types (MIME types, for short) are standardized names that describe the contents of a message entity body (e.g., text/html, image/jpeg). This appendix explains how MIME types work, how to register new ones, and where to go for more information.

In addition, this appendix contains 10 convenient tables, detailing hundreds of MIME types, gathered from many sources around the globe. This may be the most detailed tabular listing of MIME types ever compiled. We hope these tables are useful to you.

In this appendix, we will:

- Outline the primary reference material, in "Background."
- Explain the structure of MIME types, in "MIME Type Structure."
- Show you how to register MIME types, in "MIME Type IANA Registration."
- Make it easier for you to look up MIME types.

The following MIME type tables are included in this appendix:

- application/*—Table D-3
- audio/*—Table D-4
- chemical/*—Table D-5
- image/*—Table D-6
- message/*—Table D-7
- model/*—Table D-8
- multipart/*—Table D-9
- text/*—Table D-10
- video/*—Table D-11
- Other—Table D-12

Background

MIME types originally were developed for multimedia email (MIME stands for Multipurpose Internet Mail Extensions), but they have been reused for HTTP and several other protocols that need to describe the format and purpose of data objects.

MIME is defined by five primary documents:

RFC 2045, "MIME: Format of Internet Message Bodies"
: Describes the overall MIME message structure, and introduces the Content-Type header, borrowed by HTTP

RFC 2046, "MIME: Media Types"
: Introduces MIME types and their structure

RFC 2047, "MIME: Message Header Extensions for Non-ASCII Text"
: Defines ways to include non-ASCII characters in headers

RFC 2048, "MIME: Registration Procedures"
: Defines how to register MIME values with the Internet Assigned Numbers Authority (IANA)

RFC 2049, "MIME: Conformance Criteria and Examples"
: Details rules for compliance, and provides examples

For the purposes of HTTP, we are most interested in RFC 2046 (Media Types) and RFC 2048 (Registration Procedures).

MIME Type Structure

Each MIME media type consists of a type, a subtype, and a list of optional parameters. The type and subtype are separated by a slash, and the optional parameters begin with a semicolon, if they are present. In HTTP, MIME media types are widely used in Content-Type and Accept headers. Here are a few examples:

```
Content-Type: video/quicktime
Content-Type: text/html; charset="iso-8859-6"
Content-Type: multipart/mixed; boundary=gc0p4Jq0M2Yt08j34c0p
Accept: image/gif
```

Discrete Types

MIME types can directly describe the object type, or they can describe collections or packages of other object types. If a MIME type describes an object type directly, it is a *discrete type*. These include text files, videos, and application-specific file formats.

Composite Types

If a MIME type describes a collection or encapsulation of other content, the MIME type is called a *composite type*. A composite type describes the format of the enclosing

package. When the enclosing package is opened, each enclosed object will have its own type.

Multipart Types

Multipart media types are composite types. A multipart object consists of multiple component types. Here's an example of multipart/mixed content, where each component has its own MIME type:

```
Content-Type: multipart/mixed; boundary=unique-boundary-1

--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

Hi there, I'm some boring ASCII text...

--unique-boundary-1
Content-Type: multipart/parallel; boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic

    ... 8000 Hz single-channel mu-law-format
        audio data goes here ...

--unique-boundary-2
Content-Type: image/jpeg

    ... image data goes here ...

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>enriched.</italic></bold>
<smaller>as defined in RFC 1896</smaller>

Isn't it <bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

    ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

Syntax

As we stated earlier, MIME types consist of a primary type, a subtype, and an optional list of parameters.

The primary type can be a predefined type, an IETF-defined extension token, or an experimental token (beginning with "x-"). Some common primary types are described in Table D-1.

Table D-1. Common primary MIME types

Type	Description
application	Application-specific content format (discrete type)
audio	Audio format (discrete type)
chemical	Chemical data set (discrete IETF extension type)
image	Image format (discrete type)
message	Message format (composite type)
model	3-D model format (discrete IETF extension type)
multipart	Collection of multiple objects (composite type)
text	Text format (discrete type)
video	Video movie format (discrete type)

Subtypes can be primary types (as in "text/text"), IANA-registered subtypes, or experimental extension tokens (beginning with "x-").

Types and subtypes are made up of a subset of US-ASCII characters. Spaces and certain reserved grouping and punctuation characters, called "tspecials," are control characters and are forbidden from type and subtype names.

The grammar from RFC 2046 is shown below:

```
TYPE := "application" | "audio" | "image" | "message" | "multipart" |
        "text" | "video" | IETF-TOKEN | X-TOKEN
SUBTYPE := IANA-SUBTOKEN | IETF-TOKEN | X-TOKEN

IETF-TOKEN := <extension token with RFC and registered with IANA>
IANA-SUBTOKEN := <extension token registered with IANA>
X-TOKEN := <"X-" or "x-" prefix, followed by any token>

PARAMETER := TOKEN "=" VALUE
VALUE := TOKEN / QUOTED-STRING
TOKEN := 1*<any (US-ASCII) CHAR except SPACE, CTLs, or TSPECIALS>
TSPECIALS :=  "(" | ")" | "<" | ">" | "@" |
              "," | ";" | ":" | "\" | <"> |
              "/" | "[" | "]" | "?" | "="
```

MIME Type IANA Registration

The MIME media type registration process is described in RFC 2048. The goal of the registration process is to make it easy to register new media types but also to provide some sanity checking to make sure the new types are well thought out.

Registration Trees

MIME type tokens are split into four classes, called "registration trees," each with its own registration rules. The four trees—IETF, vendor, personal, and experimental—are described in Table D-2.

Table D-2. Four MIME media type registration trees

Registration tree	Example	Description
IETF	text/html (HTML text)	The IETF tree is intended for types that are of general significance to the Internet community. New IETF tree media types require approval by the Internet Engineering Steering Group (IESG) and an accompanying standards-track RFC. IETF tree types have no periods (.) in tokens.
Vendor (vnd.)	image/vnd.fpx (Kodak FlashPix image)	The vendor tree is intended for media types used by commercially available products. Public review of new vendor types is encouraged but not required. Vendor tree types begin with "vnd.".
Personal/Vanity (prs.)	image/prs.btif (internal check-management format used by Nations Bank)	Private, personal, or vanity media types can be registered in the personal tree. These media types will not be distributed commercially. Personal tree types begin with "prs.".
Experimental (x- or x.)	application/x-tar (Unix tar archive)	The experimental tree is for unregistered or experimental media types. Because it's relatively simple to register a new vendor or personal media type, software should not be distributed widely using x- types. Experimental tree types begin with "x." or "x-".

Registration Process

Read RFC 2048 carefully for the details of MIME media type registration.

The basic registration process is not a formal standards process; it's just an administrative procedure intended to sanity check new types with the community, and record them in a registry, without much delay. The process follows the following steps:

1. Present the media type to the community for review.

   Send a proposed media type registration to the *ietf-types@iana.org* mailing list for a two-week review period. The public posting solicits feedback about the choice of name, interoperability, and security implications. The "x-" prefix specified in RFC 2045 can be used until registration is complete.

2. IESG approval (for IETF tree only).

   If the media type is being registered in the IETF tree, it must be submitted to the IESG for approval and must have an accompanying standards-track RFC.

3. IANA registration.

   As soon as the media type meets the approval requirements, the author can submit the registration request to the IANA, using the email template in Example D-1 and mailing the information to *ietf-types@iana.org*. The IANA will register the media type and make the media type application available to the community at *http://www.isi.edu/in-notes/iana/assignments/media-types/*.

Registration Rules

The IANA will register media types in the IETF tree only in response to a communication from the IESG stating that a given registration has been approved.

Vendor and personal types will be registered by the IANA automatically and without any formal review as long as the following minimal conditions are met:

1. Media types must function as actual media formats. Types that act like transfer encodings or character sets may not be registered as media types.
2. All media types must have proper type and subtype names. All type names must be defined by standards-track RFCs. All subtype names must be unique, must conform to the MIME grammar for such names, and must contain the proper tree prefixes.
3. Personal tree types must provide a format specification or a pointer to one.
4. Any security considerations given must not be obviously bogus. Everyone who is developing Internet software needs to do his part to prevent security holes.

Registration Template

The actual IANA registration is done via email. You complete a registration form using the template shown in Example D-1, and mail it to *ietf-types@iana.org*.*

Example D-1. IANA MIME registration email template

```
To: ietf-types@iana.org
Subject: Registration of MIME media type XXX/YYY

MIME media type name:
```

* The lightly structured nature of the form makes the submitted information fine for human consumption but difficult for machine processing. This is one reason why it is difficult to find a readable, well-organized summary of MIME types, and the reason we created the tables that end this appendix.

Example D-1. IANA MIME registration email template (continued)

```
MIME subtype name:

Required parameters:

Optional parameters:

Encoding considerations:

Security considerations:

Interoperability considerations:

Published specification:

Applications which use this media type:

Additional information:

    Magic number(s):
    File extension(s):
    Macintosh File Type Code(s):

Person & email address to contact for further information:

Intended usage:

(One of COMMON, LIMITED USE or OBSOLETE)

Author/Change controller:

(Any other information that the author deems interesting may be added below this line.)
```

MIME Media Type Registry

The submitted forms are accessible from the IANA web site (*http://www.iana.org*). At the time of writing, the actual database of MIME media types is stored on an ISI web server, at *http://www.isi.edu/in-notes/iana/assignments/media-types/*.

The media types are stored in a directory tree, structured by primary type and subtype, with one leaf file per media type. Each file contains the email submission. Unfortunately, each person completes the registration template slightly differently, so the quality and format of information varies across submissions. (In the tables in this appendix, we tried to fill in the holes omitted by registrants.)

MIME Type Tables

This section summarizes hundreds of MIME types in 10 tables. Each table lists the MIME media types within a particular primary type (image, text, etc.).

The information is gathered from many sources, including the IANA media type registry, the Apache *mime.types* file, and assorted Internet web pages. We spent several days refining the data, plugging holes, and including descriptive summaries from cross-references to make the data more useful.

This may well be the most detailed tabular listing of MIME types ever compiled. We hope you find it handy!

application/*

Table D-3 describes many of the application-specific MIME media types.

Table D-3. "Application" MIME types

MIME type	Description	Extension	Contact and reference
application/activemessage	Supports the Active Mail groupware system.		"Active Mail: A Framework for Integrated Groupware Applications" in *Readings in Groupware and Computer-Supported Cooperative Work*, Ronald M. Baecker, ed., Morgan Kaufmann, ISBN 1558602410
application/andrew-inset	Supports the creation of multimedia content with the Andrew toolkit.	ez	*Multimedia Applications Development with the Andrew Toolkit*, Nathaniel S. Borenstein, Prentice Hall, ASIN 0130366331 *nsb@bellcore.com*
application/applefile	Permits MIME-based transmission of data with Apple/Macintosh-specific information, while allowing general access to nonspecific user data.		RFC 1740
application/atomicmail	ATOMICMAIL was an experimental research project at Bellcore, designed for including programs in electronic mail messages that are executed when mail is read. ATOMICMAIL is rapidly becoming obsolete in favor of safe-tcl.		"ATOMICMAIL Language Reference Manual," Nathaniel S. Borenstein, Bellcore Technical Memorandum TM ARH-018429
application/batch-SMTP	Defines a MIME content type suitable for tunneling an ESMTP mail transaction through any MIME-capable transport.		RFC 2442
application/beep+xml	Supports the interaction protocol called BEEP. BEEP permits simultaneous and independent exchanges of MIME messages between peers, where the messages usually are XML-structured text.		RFC 3080

Table D-3: "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/cals-1840	Supports MIME email exchanges of U.S. Department of Defense digital data that was previously exchanged by tapem, as defined by MIL-STD-1840.		RFC 1895
application/commonground	Common Ground is an electronic document exchange and distribution program that lets users create documents that anyone can view, search, and print, without requiring that they have the creating applications or fonts on their systems.		Nick Gault No Hands Software *ngault@nohands.com*
application/cybercash	Supports credit card payment through the CyberCash protocol. When a user starts payment, a message is sent by the merchant to the customer as the body of a message of MIME type application/cybercash.		RFC 1898
application/dca-rft	IBM Document Content Architecture.		"IBM Document Content Architecture/Revisable Form Text Reference," document number SC23-0758-1, International Business Machines
application/dec-dx	DEC Document Transfer Format.		"Digital Document Transmission (DX) Technical Notebook," document number EJ29141-86, Digital Equipment Corporation
application/dvcs	Supports the protocols used by a Data Validation and Certification Server (DVCS), which acts as a trusted third party in a public-key security infrastructure.		RFC 3029
application/EDI-Consent	Supports bilateral trading via electronic data interchange (EDI), using nonstandard specifications.		*http://www.isi.edu/in-notes/iana/assignments/media-types/application/EDI-Consent*
application/EDI-X12	Supports bilateral trading via electronic data interchange (EDI), using the ASC X12 EDI specifications.		*http://www.isi.edu/in-notes/iana/assignments/media-types/application/EDI-X12*
application/EDIFACT	Supports bilateral trading via electronic data interchange (EDI), using the EDIFACT specifications.		*http://www.isi.edu/in-notes/iana/assignments/media-types/application/EDIFACT*
application/eshop	Unknown.		Steve Katz System Architecture Shop *steve_katz@eshop.com*
application/font-tdpfr	Defines a Portable Font Resource (PFR) that contains a set of glyph shapes, each associated with a character code.		RFC 3073

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/http	Used to enclose a pipeline of one or more HTTP request or response messages (not intermixed).		RFC 2616
application/hyperstudio	Supports transfer of HyperStudio educational hypermedia files.	stk	*http://www.hyperstudio.com*
application/iges	A commonly used format for CAD model interchange.		"ANS/US PRO/IPO-100" U.S. Product Data Association 2722 Merrilee Drive, Suite 200 Fairfax, VA 22031-4499
application/index application/index.cmd application/index.obj application/index.response application/index.vnd	Support the Common Indexing Protocol (CIP). CIP is an evolution of the Whois++ directory service, used to pass indexing information from server to server in order to redirect and replicate queries through a distributed database system.		RFC 2652, and RFCs 2651, 1913, and 1914
application/iotp	Supports Internet Open Trading Protocol (IOTP) messages over HTTP.		RFC 2935
application/ipp	Supports Internet Printing Protocol (IPP) over HTTP.		RFC 2910
application/mac-binhex40	Encodes a string of 8-bit bytes into a string of 7-bit bytes, which is safer for some applications (though not quite as safe as the 6-bit base-64 encoding).	hqx	RFC 1341
application/mac-compactpro	From Apache *mime.types*.	cpt	
application/macwriteii	Claris MacWrite II.		
application/marc	MARC objects are Machine-Readable Cataloging records—standards for the representation and communication of bibliographic and related information.	mrc	RFC 2220
application/mathematica application/mathematica-old	Supports Mathematica and MathReader numerical analysis software.	nb, ma, mb	*The Mathematica Book*, Stephen Wolfram, Cambridge University Press, ISBN 0521643147
application/msword	Microsoft Word MIME type.	doc	
application/news-message-id			RFCs 822 (message IDs), 1036 (application to news), and 977 (NNTP)
application/news-transmission	Allows transmission of news articles by email or other transport.		RFC 1036
application/ocsp-request	Supports the Online Certificate Status Protocol (OCSP), which provides a way to check on the validity of a digital certificate without requiring local certificate revocation lists.	orq	RFC 2560

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/ocsp-response	Same as above.	ors	RFC 2560
application/octet-stream	Unclassified binary data.	bin, dms, lha, lzh, exe, class	RFC 1341
application/oda	Used for information encoded according to the Office Document Architecture (ODA) standards, using the Office Document Interchange Format (ODIF) representation format. The Content-Type line also should specify an attribute/value pair that indicates the document application profile (DAP), as in: Content-Type: application/oda; profile=Q112	oda	RFC 1341 ISO 8613; "Information Processing: Text and Office System; Office Document Architecture (ODA) and Interchange Format (ODIF)," Part 1-8, 1989
application/parityfec	Forward error correction parity encoding for RTP data streams.		RFC 3009
application/pdf	Adode PDF files.	pdf	See *Portable Document Format Reference Manual*, Adobe Systems, Inc., Addison Wesley, ISBN 0201626284
application/pgp-encrypted	PGP encrypted data.		RFC 2015
application/pgp-keys	PGP public-key blocks.		RFC 2015
application/pgp-signature	PGP cryptographic signature.		RFC 2015
application/pkcs10	Public Key Crypto System #10—the application/pkcs10 body type *must* be used to transfer a PKCS #10 certification request.	p10	RFC 2311
application/pkcs7-mime	Public Key Crypto System #7—this type is used to carry PKCS #7 objects of several types including envelopedData and signedData.	p7m	RFC 2311
application/pkcs7-signature	Public Key Crypto System #7—this type always contains a single PKCS #7 object of type signedData.	p7s	RFC 2311
application/pkix-cert	Transports X.509 certificates.	cer	RFC 2585
application/pkix-crl	Transports X.509 certificate revocation lists.	crl	RFC 2585
application/pkixcmp	Message format used by X.509 Public Key Infrastructure Certificate Management Protocols.	pki	RFC 2510
application/postscript	An Adobe PostScript graphics file (program).	ai, ps, eps	RFC 2046
application/prs.alvestrand.titrax-sheet	"TimeTracker" program by Harald T. Alvestrand.		*http://domen.uninett.no/~hta/titrax/*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/prs.cww	CU-Writer for Windows.	cw, cww	Dr. Somchai Prasitjutrakul *somchaip@chulkn.car.chula.ac.th*
application/prs.nprend	Unknown.	rnd, rct	John M. Doggett *jdoggett@tiac.net*
application/remote-printing	Contains meta information used when remote printing, for the printer cover sheet.		RFC 1486 Marshall T. Rose *mrose@dbc.mtview.ca.us*
application/riscos	Acorn RISC OS binaries.		*RISC OS Programmer's Reference Manuals*, Acorn Computers, Ltd., ISBN1852501103
application/sdp	SDP is intended for describing live multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation.		RFC 2327 Henning Schulzrinne *hgs@cs.columbia.edu*
application/set-payment application/set-payment-initiation application/set-registration application/set-registration-initiation	Supports the SET secure electronic transaction payment protocol.		*http://www.visa.com* *http://www.mastercard.com*
application/sgml-open-catalog	Intended for use with systems that support the SGML Open TR9401:1995 "Entity Management" specification.		SGML Open 910 Beaver Grade Road, #3008 Coraopolis, PA 15109 *info@sgmlopen.org*
application/sieve	Sieve mail filtering script.		RFC 3028
application/slate	The BBN/Slate document format is published as part of the standard documentation set distributed with the BBN/Slate product.		BBN/Slate Product Mgr BBN Systems and Technologies 10 Moulton Street Cambridge, MA 02138
application/smil	The Synchronized Multimedia Integration Language (SMIL) integrates a set of independent multimedia objects into a synchronized multimedia presentation.	smi, smil	*http://www.w3.org/AudioVideo/*
application/tve-trigger	Supports embedded URLs in enhanced television receivers.		"SMPTE: Declarative Data Essence, Content Level 1," produced by the Society of Motion Picture and Television Engineers *http://www.smpte.org*
application/vemmi	Enhanced videotex standard.		RFC 2122
application/vnd.3M.Post-it-Notes	Used by the "Post-it® Notes for Internet Designers" Internet control/plug-in.	pwn	*http://www.3M.com/psnotes/*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.accpac.simply.aso	Simply Accounting v7.0 and higher. Files of this type conform to Open Financial Exchange v1.02 specifications.	aso	*http://www.ofx.net*
application/vnd.accpac.simply.imp	Used by Simply Accounting v7.0 and higher, to import its own data.	imp	*http://www.ofx.net*
application/vnd.acucobol	ACUCOBOL-GT Runtime.		Dovid Lubin *dovid@acucobol.com*
application/vnd.aether.imp	Supports airtime-efficient Instant Message communications between an Instant Messaging service, such as AOL Instant Messenger, Yahoo! Messenger, or MSN Messenger, and a special set of Instant Messaging client software on a wireless device.		Wireless Instant Messaging Protocol (IMP) specification available from Aether Systems by license
application/vnd.anser-web-certificate-issue-initiation	Trigger for web browsers to launch the ANSER-WEB Terminal Client.	cii	Hiroyoshi Mori *mori@mm.rd.nttdata.co.jp*
application/vnd.anser-web-funds-transfer-initiation	Same as above.	fti	Same as above
application/vnd.audiograph	AudioGraph.	aep	Horia Cristian *H.C.Slusanschi@massey.ac.nz*
application/vnd.bmi	BMI graphics format by CADAM Systems.	bmi	Tadashi Gotoh *tgotoh@cadamsystems.co.jp*
application/vnd.businessobjects	BusinessObjects 4.0 and higher.	rep	
application/vnd.canon-cpdl application/vnd.canon-lips	Supports Canon, Inc. office imaging products.		Shin Muto *shinmuto@pure.cpdc.canon.co.jp*
application/vnd.claymore	Claymore.exe.	cla	Ray Simpson *ray@cnation.com*
application/vnd.commerce-battelle	Supports a generic mechanism for delimiting smart card–based information, for digital commerce, identification, authentication, and exchange of smart card–based card holder information.	ica, icf, icd, icc, ic0, ic1, ic2, ic3, ic4, ic5, ic6, ic7, ic8	David C. Applebaum *applebau@131.167.52.15*
application/vnd.commonspace	Allows for proper transmission of CommonSpace™ documents via MIME-based processes. CommonSpace is published by Sixth Floor Media, part of the Houghton-Mifflin Company.	csp, cst	Ravinder Chandhok *chandhok@within.com*
application/vnd.contact.cmsg	Used for CONTACT software's CIM DATABASE.	cdbcmsg	Frank Patz *fp@contact.de* *http://www.contact.de*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.cosmocaller	Allows for files containing connection parameters to be downloaded from web sites, invokes the CosmoCaller application to interpret the parameters, and initiates connections with the CosmoCallACD server.	cmc	Steve Dellutri *sdellutri@cosmocom.com*
application/vnd.ctc-posml	Continuum Technology's PosML.	pml	Bayard Kohlhepp *bayardk@ctcexchange.com*
application/vnd.cups-postscript application/vnd.cups-raster application/vnd.cups-raw	Supports Common UNIX Printing System (CUPS) servers and clients.		*http://www.cups.org*
application/vnd.cybank	Proprietary data type for Cybank data.		Nor Helmee B. Abd. Halim *helmee@cybank.net* *http://www.cybank.net*
application/vnd.dna	DNA is intended to easily Web-enable any 32-bit Windows application.	dna	Meredith Searcy *msearcy@newmoon.com*
application/vnd.dpgraph	Used by DPGraph 2000 and MathWare Cyclone.	dpg, mwc, dpgraph	David Parker *http://www.davidparker.com*
application/vnd.dxr	Digital Xpress Reports by PSI Technologies.	dxr	Michael Duffy *miked@psiaustin.com*
application/vnd.ecdis-update	Supports ECDIS applications.		*http://www.sevencs.com*
application/vnd.ecowin.chart application/vnd.ecowin.filerequest application/vnd.ecowin.fileupdate application/vnd.ecowin.series application/vnd.ecowin.seriesrequest application/vnd.ecowin.seriesupdate	EcoWin.	mag	Thomas Olsson *thomas@vinga.se*
application/vnd.enliven	Supports delivery of Enliven interactive multimedia.	nml	Paul Santinelli *psantinelli@narrative.com*
application/vnd.epson.esf	Proprietary content for Seiko Epson QUASS Stream Player.	esf	Shoji Hoshina *Hoshina.Shoji@exc.epson.co.jp*
application/vnd.epson.msf	Proprietary content for Seiko Epson QUASS Stream Player.	msf	Same as above
application/vnd.epson.quickanime	Proprietary content for Seiko Epson QuickAnime Player.	qam	Yu Gu *guyu@rd.oda.epson.co.jp*
application/vnd.epson.salt	Proprietary content for Seiko Epson SimpleAnimeLite Player.	slt	Yasuhito Nagatomo *naga@rd.oda.epson.co.jp*
application/vnd.epson.ssf	Proprietary content for Seiko Epson QUASS Stream Player.	ssf	Shoji Hoshina *Hoshina.Shoji@exc.epson.co.jp*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.ericsson.quickcall	Phone Doubler Quick Call.	qcall, qca	Paul Tidwell *paul.tidwell@ericsson.com* *http://www.ericsson.com*
application/vnd.eudora.data	Eudora Version 4.3 and later.		Pete Resnick *presnick@qualcomm.com*
application/vnd.fdf	Adobe Forms Data Format.		"Forms Data Format," Technical Note 5173, Adobe Systems
application/vnd.ffsns	Used for application communication with FirstFloor's Smart Delivery.		Mary Holstege *holstege@firstfloor.com*
application/vnd.FloGraphIt	NpGraphIt.	gph	
application/vnd.framemaker	Adobe FrameMaker files.	fm, mif, book	*http://www.adobe.com*
application/vnd.fsc.weblaunch	Supports Friendly Software Corporation's golf simulation software.	fsc	Derek Smith *derek@friendlysoftware.com*
application/vnd.fujitsu.oasys application/vnd.fujitsu.oasys2	Supports Fujitsu's OASYS software.	oas	Nobukazu Togashi *togashi@ai.cs.fujitsu.co.jp*
application/vnd.fujitsu.oasys2	Supports Fujitsu's OASYS V2 software.	oa2	Same as above
application/vnd.fujitsu.oasys3	Support's Fujitsu's OASYS V5 software.	oa3	Seiji Okudaira *okudaira@candy.paso.fujitsu.co.jp*
application/vnd.fujitsu.oasysgp	Supports Fujitsu's OASYS GraphPro software.	fg5	Masahiko Sugimoto *sugimoto@sz.sel.fujitsu.co.jp*
application/vnd.fujitsu.oasysprs	Support's Fujitsu's OASYS Presentation software.	bh2	Masumi Ogita *ogita@oa.tfl.fujitsu.co.jp*
application/vnd.fujixerox.ddd	Supports Fuji Xerox's EDMICS 2000 and DocuFile.	ddd	Masanori Onda *Masanori.Onda@fujixerox.co.jp*
application/vnd.fujixerox.docuworks	Supports Fuji Xerox's DocuWorks Desk and DocuWorks Viewer software.	xdw	Yasuo Taguchi *yasuo.taguchi@fujixerox.co.jp*
application/vnd.fujixerox.docuworks.binder	Supports Fuji Xerox's DocuWorks Desk and DocuWorks Viewer software.	xbd	Same as above.
application/vnd.fut-misnet	Unknown.		Jaan Pruulmann *jaan@fut.ee*
application/vnd.grafeq	Lets users of GrafEq exchange GrafEq documents through the Web and email.	gqf, gqs	*http://www.peda.com*
application/vnd.groove-account	Groove is a peer-to-peer communication system implementing a virtual space for small group interaction.	gac	Todd Joseph *todd_joseph@groove.net*
application/vnd.groove-identity-message	Same as above.	gim	Same as above
application/vnd.groove-injector	Same as above.	grv	Same as above
application/vnd.groove-tool-message	Same as above.	gtm	Same as above

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.groove-tool-template	Same as above.	tpl	Same as above
application/vnd.groove-vcard	Same as above.	vcg	Same as above
application/vnd.hhe.lesson-player	Supports the LessonPlayer and PresentationEditor software.	les	Randy Jones Harcourt E-Learning *randy_jones@archipelago.com*
application/vnd.hp-HPGL	HPGL files.		*The HP-GL/2 and HP RTL Reference Guide*, Addison Wesley, ISBN 0201310147
application/vnd.hp-hpid	Supports Hewlett-Packard's Instant Delivery Software.	hpi, hpid	*http://www.instant-delivery.com*
application/vnd.hp-hps	Supports Hewlett-Packard's Web-PrintSmart software.	hps	*http://www.hp.com/go/webprintsmart_mimetype_specs/*
application/vnd.hp-PCL application/vnd.hp-PCLXL	PCL printer files.	pcl	"PCL-PJL Technical Reference Manual Documentation Package," HP Part No. 5012-0330
application/vnd.httphone	HTTPhone asynchronous voice over IP system.		Franck LeFevre *franck@k1info.com*
application/vnd.hzn-3d-crossword	Used to encode crossword puzzles by Horizon, A Glimpse of Tomorrow.	x3d	James Minnis *james_minnis@glimpse-of-tomorrow.com*
application/vnd.ibm.afplinedata	Print Services Facility (PSF), AFP Conversion and Indexing Facility (ACIF).		Roger Buis *buis@us.ibm.com*
application/vnd.ibm.MiniPay	MiniPay authentication and payment software.	mpy	Amir Herzberg *amirh@vnet.ibm.com*
application/vnd.ibm.modcap	Mixed Object Document Content.	list3820, listafp, afp, pseg3820	Reinhard Hohensee *rhohensee@vnet.ibm.com* "Mixed Object Document Content Architecture Reference," IBM publication SC31-6802
application/vnd.informix-visionary	Informix Visionary.	vis	Christopher Gales *christopher.gales@informix.com*
application/vnd.intercon.formnet	Supports Intercon Associates FormNet software.	xpw, xpx	Thomas A. Gurak *assoc@intercon.roc.servtech.com*
application/vnd.intertrust.digibox application/vnd.intertrust.nncp	Supports InterTrust architecture for secure electronic commerce and digital rights management.		InterTrust Technologies 460 Oakmead Parkway Sunnyvale, CA 94086 USA *info@intertrust.com* *http://www.intertrust.com*
application/vnd.intu.qbo	Intended for use only with QuickBooks 6.0 (Canada).	qbo	Greg Scratchley *greg_scratchley@intuit.com* Format of these files discussed in the Open Financial Exchange specs, available from *http://www.ofx.net*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.intu.qfx	Intended for use only with Quicken 99 and following versions.	qfx	Same as above
application/vnd.is-xpr	Express by Infoseek.	xpr	Satish Natarajan *satish@infoseek.com*
application/vnd.japannet-directory-service application/vnd.japannet-jpnstore-wakeup application/vnd.japannet-payment-wakeup application/vnd.japannet-registration application/vnd.japannet-registration-wakeup application/vnd.japannet-setstore-wakeup application/vnd.japannet-verification application/vnd.japannet-verification-wakeup	Supports Mitsubishi Electric's Japan-Net security, authentication, and payment sofwtare.		Jun Yoshitake *yositake@iss.isl.melco.co.jp*
application/vnd.koan	Supports the automatic playback of Koan music files over the Internet, by helper applications such as SSEYO Koan Netscape Plugin.	skp, skd, skm, skt	Peter Cole *pcole@sseyod.demon.co.uk*
application/vnd.lotus-1-2-3	Lotus 1-2-3 and Lotus approach.	123, wk1, wk3, wk4	Paul Wattenberger *Paul_Wattenberger@lotus.com*
application/vnd.lotus-approach	Lotus Approach.	apr, vew	Same as above
application/vnd.lotus-freelance	Lotus Freelance.	prz, pre	Same as above
application/vnd.lotus-notes	Lotus Notes.	nsf, ntf, ndl, ns4, ns3, ns2, nsh, nsg	Michael Laramie *laramiem@btv.ibm.com*
application/vnd.lotus-organizer	Lotus Organizer.	or3, or2, org	Paul Wattenberger *Paul_Wattenberger@lotus.com*
application/vnd.lotus-screencam	Lotus ScreenCam.	scm	Same as above
application/vnd.lotus-wordpro	Lotus Word Pro.	lwp, sam	Same as above
application/vnd.mcd	Micro CADAM CAD software.	mcd	Tadashi Gotoh *tgotoh@cadamsystems.co.jp* *http://www.cadamsystems.co.jp*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.mediastation.cdkey	Supports Media Station's CDKey remote CDROM communications protocol.	cdkey	Henry Flurry *henryf@mediastation.com*
application/vnd.meridian-slingshot	Slingshot by Meridian Data.		Eric Wedel Meridian Data, Inc. 5615 Scotts Valley Drive Scotts Valley, CA 95066 *ewedel@meridian-data.com*
application/vnd.mif	FrameMaker interchange format.	mif	*ftp://ftp.frame.com/pub/techsup/techinfo/dos/mif4.zip* Mike Wexler Adobe Systems, Inc 333 W. San Carlos St. San Jose, CA 95110 USA *mwexler@adobe.com*
application/vnd.minisoft-hp3000-save	NetMail 3000 save format.		Minisoft, Inc. *support@minisoft.com* *ftp://ftp.3k.com/DOC/ms92-save-format.txt*
application/vnd.mitsubishi.misty-guard.trustweb	Supports Mitsubishi Electric's Trustweb software.		Manabu Tanaka *mtana@iss.isl.melco.co.jp*
application/vnd.Mobius.DAF	Supports Mobius Management Systems software.	daf	Celso Rodriguez *crodrigu@mobius.com* Greg Chrzczon *gchrzczo@mobius.com*
application/vnd.Mobius.DIS	Same as above.	dis	Same as above
application/vnd.Mobius.MBK	Same as above.	mbk	Same as above
application/vnd.Mobius.MQY	Same as above.	mqy	Same as above
application/vnd.Mobius.MSL	Same as above.	msl	Same as above
application/vnd.Mobius.PLC	Same as above.	plc	Same as above
application/vnd.Mobius.TXF	Same as above.	txf	Same as above
application/vnd.motorola.flexsuite	FLEXsuite™ is a collection of wireless messaging protocols. This type is used by the network gateways of wireless messaging service providers as well as wireless OSs and applications.		Mark Patton Motorola Personal Networks Group *fmp014@email.mot.com* FLEXsuite™ specification available from Motorola under appropriate licensing agreement
application/vnd.motorola.flexsuite.adsi	FLEXsuite™ is a collection of wireless messaging protocols. This type provides a wireless-friendly format for enabling various data-encryption solutions.		Same as above

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.motorola.flexsuite.fis	FLEXsuite™ is a collection of wireless messaging protocols. This type is a wireless-friendly format for the efficient delivery of structured information (e.g., news, stocks, weather) to a wireless device.		Same as above
application/vnd.motorola.flexsuite.gotap	FLEXsuite™ is a collection of wireless messaging protocols. This type provides a common wireless-friendly format for the programming of wireless device attributes via over-the-air messages.		Same as above
application/vnd.motorola.flexsuite.kmr	FLEXsuite™ is a collection of wireless messaging protocols. This type provides a wireless-friendly format for encryption key management.		Same as above
application/vnd.motorola.flexsuite.ttc	FLEXsuite™ is a collection of wireless messaging protocols. This type supports a wireless-friendly format for the efficient delivery of text using token text compression.		Same as above
application/vnd.motorola.flexsuite.wem	FLEXsuite™ is a collection of wireless messaging protocols. This type provides a wireless-friendly format for the communication of Internet email to wireless devices.		Same as above
application/vnd.mozilla.xul+xml	Supports the Mozilla Internet application suite.	xul	Dan Rosen2 *dr@netscape.com*
application/vnd.ms-artgalry	Supports Microsoft's Art Gallery.	cil	*deansl@microsoft.com*
application/vnd.ms-asf	ASF is a multimedia file format whose contents are designed to be streamed across a network to support distributed multimedia applications. ASF content may include any combination of any media type (e.g., audio, video, images, URLs, HTML content, MIDI, 2-D and 3-D modeling, scripts, and objects of various types).	asf	Eric Fleischman *ericf@microsoft.com* *http://www.microsoft.com/mind/0997/netshow/netshow.asp*
application/vnd.ms-excel	Microsoft Excel spreadsheet.	xls	Sukvinder S. Gill *sukvg@microsoft.com*
application/vnd.ms-lrm	Microsoft proprietary.	lrm	Eric Ledoux *ericle@microsoft.com*
application/vnd.ms-powerpoint	Microsoft PowerPoint presentation.	ppt	Sukvinder S. Gill *sukvg@microsoft.com*
application/vnd.ms-project	Microsoft Project file.	mpp	Same as above

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.ms-tnef	Identifies an attachment that in general would be processable only by a MAPI-aware application. This type is an encapsulated format of rich MAPI properties, such as Rich Text and Icon information, that may otherwise be degraded by the messaging transport.		Same as above
application/vnd.ms-works	Microsoft Works software.		Same as above
application/vnd.mseq	MSEQ is a compact multimedia format suitable for wireless devices.	mseq	Gwenael Le Bodic *Gwenael.le_bodic@alcatel.fr* *http://www.3gpp.org*
application/vnd.msign	Used by applications implementing the msign protocol, which requests signatures from mobile devices.		Malte Borcherding *Malte.Borcherding@brokat.com*
application/vnd.music-niff	NIFF music files.		Cindy Grande *72723.1272@compuserve.com* *ftp://blackbox.cartah.washington.edu/pub/NIFF/NIFF6A.TXT*
application/vnd.musician	MUSICIAN scoring language/encoding conceived and developed by Renai-Science Corporation.	mus	Robert G. Adams *gadams@renaiscience.com*
application/vnd.netfpx	Intended for dynamic retrieval of multiresolution image information, as used by Hewlett-Packard Company Imaging for Internet.	fpx	Andy Mutz *andy_mutz@hp.com*
application/vnd.noblenet-directory	Supports the NobleNet Directory software, purchased by RogueWave.	nnd	*http://www.noblenet.com*
application/vnd.noblenet-sealer	Supports the NobleNet Sealer software, purchased by RogueWave.	nns	*http://www.noblenet.com*
application/vnd.noblenet-web	Supports the NobleNet Web software, purchased by RogueWave.	nnw	*http://www.noblenet.com*
application/vnd.novadigm.EDM	Supports Novadigm's RADIA and EDM products.	edm	Phil Burgard *pburgard@novadigm.com*
application/vnd.novadigm.EDX	Same as above.	edx	Same as above
application/vnd.novadigm.EXT	Same as above.	ext	Same as above
application/vnd.osa.netdeploy	Supports the Open Software Associates netDeploy application deployment software.	ndc	Steve Klos *stevek@osa.com* *http://www.osa.com*
application/vnd.palm	Used by PalmOS system software and applications—this new type, "application/vnd.palm," replaces the old type "application/x-pilot."	prc, pdb, pqa, oprc	Gavin Peacock *gpeacock@palm.com*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.pg.format	Proprietary Proctor & Gamble Standard Reporting System.	str	April Gandert TN152 Procter & Gamble Way Cincinnati, Ohio 45202 (513) 983-4249
application/vnd.pg.osasli	Proprietary Proctor & Gamble Standard Reporting System.	ei6	Same as above
application/vnd.powerbuilder6 application/vnd.powerbuilder6-s application/vnd.powerbuilder7 application/vnd.powerbuilder7-s application/vnd.powerbuilder75 application/vnd.powerbuilder75-s	Used only by Sybase PowerBuilder release 6, 7, and 7.5 runtime environments, nonsecure and secure.	pbd	Reed Shilts *reed.shilts@sybase.com*
application/vnd.previewsystems.box	Preview Systems ZipLock/VBox product.	box, vbox	Roman Smolgovsky *romans@previewsystems.com* *http://www.previewsystems.com*
application/vnd.publishare-delta-tree	Used by Capella Computers' PubliShare runtime environment.	qps	Oren Ben-Kiki *publishare-delta-tree@capella.co.il*
application/vnd.rapid	Emultek's rapid packaged applications.	zrp	Itay Szekely *etay@emultek.co.il*
application/vnd.s3sms	Integrates the transfer mechanisms of the Sonera SmartTrust products into the Internet infrastructure.		Lauri Tarkkala *Lauri.Tarkkala@sonera.com* *http://www.smarttrust.com*
application/vnd.seemail	Supports the transmission of SeeMail files. SeeMail is an application that captures video and sound and uses bitwise compression to compress and archive the two pieces into one file.	see	Steven Webb *steve@wynde.com* *http://www.realmediainc.com*
application/vnd.shana.informed.formdata	Shana e-forms data formats.	ifm	Guy Selzler Shana Corporation *gselzler@shana.com*
application/vnd.shana.informed.formtemp	Shana e-forms data formats.	itp	Same as above
application/vnd.shana.informed.interchange	Shana e-forms data formats.	iif, iif1	Same as above
application/vnd.shana.informed.package	Shana e-forms data formats.	ipk, ipkg	Same as above
application/vnd.street-stream	Proprietary to Street Technologies.		Glenn Levitt Street Technologies *streetd1@ix.netcom.com*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.svd	Dateware Electronics SVD files.		Scott Becker *dataware@compumedia.com*
application/vnd.swiftview-ics	Supports SwiftView®.		Randy Prakken *tech@ndg.com* *http://www.ndg.com/svm.htm*
application/vnd.triscape.mxs	Supports Triscape Map Explorer.	mxs	Steven Simonoff *scs@triscape.com*
application/vnd.trueapp	True BASIC files.	tra	J. Scott Hepler *scott@truebasic.com*
application/vnd.truedoc	Proprietary to Bitstream, Inc.		Brad Chase *brad_chase@bitstream.com*
application/vnd.ufdl	UWI's UFDL files.	ufdl, ufd, frm	Dave Manning *dmanning@uwi.com* *http://www.uwi.com/*
application/vnd.uplanet.alert application/vnd.uplanet.alert-wbxml application/vnd.uplanet.bearer-choi-wbxml application/vnd.uplanet.bearer-choice application/vnd.uplanet.cacheop application/vnd.uplanet.cacheop-wbxml application/vnd.uplanet.channel application/vnd.uplanet.channel-wbxml application/vnd.uplanet.list application/vnd.uplanet.list-wbxml application/vnd.uplanet.listcmd application/vnd.uplanet.listcmd-wbxml application/vnd.uplanet.signal	Formats used by Unwired Planet (now Openwave) UP browser micro-browser for mobile devices.		*iana-registrar@uplanet.com* *http://www.openwave.com*
application/vnd.vcx	VirtualCatalog.	vcx	Taisuke Sugimoto *sugimototi@noanet.nttdata.co.jp*
application/vnd.vectorworks	VectorWorks graphics files.	mcd	Paul C. Pharr *pharr@diehlgraphsoft.com*
application/vnd.vidsoft.vidconference	VidConference format.	vsc	Robert Hess *hess@vidsoft.de*
application/vnd.visio	Visio files.	vsd, vst, vsw, vss	Troy Sandal *troys@visio.com*

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/vnd.vividence.scriptfile	Vividence files.	vsf, vtd, vd	Mark Risher *markr@vividence.com*
application/vnd.wap.sic	WAP Service Indication format.	sic, wbxml	WAP Forum Ltd *http://www.wapforum.org*
application/vnd.wap.slc	WAP Service Loading format. Anything that conforms to the Service Loading specification, available at *http://www.wapforum.org*.	slc, wbxml	Same as above
application/vnd.wap.wbxml	WAP WBXML binary XML format for wireless devices.	wbxml	Same as above "WAP Binary XML Content Format—WBXML version 1.1"
application/vnd.wap.wmlc	WAP WML format for wireless devices.	wmlc, wbxml	Same as above
application/vnd.wap.wmlscriptc	WAP WMLScript format.	wmlsc	Same as above
application/vnd.webturbo	WebTurbo format.	wtb	Yaser Rehem Sapient Corporation *yrehem@sapient.com*
application/vnd.wrq-hp3000-labelled	Supports HP3000 formats.		*support@wrq.com* *support@3k.com*
application/vnd.wt.stf	Supports Worldtalk software.	stf	Bill Wohler *wohler@worldtalk.com*
application/vnd.xara	Xara files are saved by CorelXARA, an object-oriented vector graphics package written by Xara Limited (and marketed by Corel).	xar	David Matthewman *david@xara.com* *http://www.xara.com*
application/vnd.xfdl	UWI's XFDL files.	xfdl, xfd, frm	Dave Manning *dmanning@uwi.com* *http://www.uwi.com*
application/vnd.yellowriver-custom-menu	Supports the Yellow River Custom-Menu plug-in, which provides customized browser drop-down menus.	cmp	*yellowriversw@yahoo.com*
application/whoispp-query	Defines Whois++ protocol queries within MIME.		RFC 2957
application/whoispp-response	Defines Whois++ protocol responses within MIME.		RFC 2958
application/wita	Wang Information Transfer Architecture.		Document number 715-0050A, Wang Laboratories *campbell@redsox.bsw.com*
application/wordperfect5.1	WordPerfect documents.		
application/x400-bp	Carries any X.400 body part for which there is no registered IANA mapping.		RFC 1494

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/x-bcpio	Old-style binary CPIO archives.	bcpio	
application/x-cdlink	Allows integration of CD-ROM media within web pages.	vcd	*http://www.cdlink.com*
application/x-chess-pgm	From Apache *mime.types*.	pgn	
application/x-compress	Binary data from Unix compress.	z	
application/x-cpio	CPIO archive file.	cpio	
application/x-csh	CSH scripts.	csh	
application/x-director	Macromedia director files.	dcr, dir, dxr	
application/x-dvi	TeX DVI files.	dvi	
application/x-futuresplash	From Apache *mime.types*.	spl	
application/x-gtar	GNU tar archives.	gtar	
application/x-gzip	GZIP compressed data.	gz	
application/x-hdf	From Apache *mime.types*.	hdf	
application/x-javascript	JavaScript files.	js	
application/x-koan	Supports the automatic playback of Koan music files over the Internet, by helper applications such as SSEYO Koan Netscape Plugin.	skp, skd, skt, skm	
application/x-latex	LaTeX files.	latex	
application/x-netcdf	NETCDF files.	nc, cdf	
application/x-sh	SH scripts.	sh	
application/x-shar	SHAR archives.	shar	
application/x-shockwave-flash	Macromedia Flash files.	swf	
application/x-stuffit	StuffIt archives.	sit	
application/x-sv4cpio	Unix SysV R4 CPIO archives.	sv4cpio	
application/x-sv4crc	Unix SysV R4 CPIO w/CRC archives.	sv4crc	
application/x-tar	TAR archives.	tar	
application/x-tcl	TCL scripts.	tcl	
application/x-tex	TeX files.	tex	
application/x-texinfo	TeX info files.	texinfo, texi	
application/x-troff	TROFF files.	t, tr, roff	
application/x-troff-man	TROFF Unix manpages.	man	
application/x-troff-me	TROFF+me files.	me	
application/x-troff-ms	TROFF+ms files	ms	

Table D-3. "Application" MIME types (continued)

MIME type	Description	Extension	Contact and reference
application/x-ustar	The extended tar interchange format.	ustar	See the IEEE 1003.1(1990) specifications
application/x-wais-source	WAIS source structure.	src	
application/xml	Extensible Markup Language format file (use text/xml if you want the file treated as plain text by browsers, etc.).	xml, dtd	RFC 2376
application/zip	PKWARE zip archives.	zip	

audio/*

Table D-4 summarizes audio content types.

Table D-4. "Audio" MIME types

MIME type	Description	Extension	Contact and reference
audio/32kadpcm	8 kHz ADPCM audio encoding.		RFC 2421
audio/basic	Audio encoded with 8-kHz monaural 8-bit ISDN u-law PCM.	au, snd	RFC 1341
audio/G.772.1	G.722.1 compresses 50Hz–7kHz audio signals into 24 kbit/s or 32 kbit/s. It may be used for speech, music, and other types of audio.		RFC 3047
audio/L16	Audio/L16 is based on L16, described in RFC 1890. L16 denotes uncompressed audio data, using 16-bit signed representation.		RFC 2586
audio/MP4A-LATM	MPEG-4 audio.		RFC 3016
audio/midi	MIDI music files.	mid, midi, kar	
audio/mpeg	MPEG encoded audio files.	mpga, mp2, mp3	RFC 3003
audio/parityfec	Parity-based forward error correction for RTP audio.		RFC 3009
audio/prs.sid	Commodore 64 SID audio files.	sid, psid	*http://www.geocities.com/SiliconValley/Lakes/5147/sidplay/docs.html#fileformats*
audio/telephone-event	Logical telephone event.		RFC 2833
audio/tone	Telephonic sound pattern.		RFC 2833
audio/vnd.cns.anp1	Supports voice and unified messaging application features available on the Access NP network services platform from Comverse Network Systems.		Ann McLaughlin Comverse Network Systems *amclaughlin@comversens.com*

Table D-4. "Audio" MIME types (continued)

MIME type	Description	Extension	Contact and reference
audio/vnd.cns.inf1	Supports voice and unified messaging application features available on the TRILOGUE Infinity network services platform from Comverse Network Systems.		Same as above
audio/vnd.digital-winds	Digital Winds music is never-ending, reproducible, and interactive MIDI music in very small packages (<3K).	eol	Armands Strazds *armands.strazds@medienhaus-bremen.de*
audio/vnd.everad.plj	Proprietary EverAD audio encoding.	plj	Tomer Weisberg *tomer@everad.com*
audio/vnd.lucent.voice	Voice messaging including Lucent Technologies' Intuity™ AUDIX® Multimedia Messaging System and the Lucent Voice Player.	lvp	Frederick Block *rickblock@lucent.com* *http://www.lucent.com/lvp/*
audio/vnd.nortel.vbk	Proprietary Nortel Networks Voice Block audio encoding.	vbk	Glenn Parsons *Glenn.Parsons@NortelNetworks.com*
audio/vnd.nuera.ecelp4800	Proprietary Nuera Communications audio and speech encoding, available in Nuera voice-over-IP gateways, terminals, application servers, and as a media service for various host platforms and OSs.	ecelp4800	Michael Fox *mfox@nuera.com*
audio/vnd.nuera.ecelp7470	Same as above.	ecelp7470	Same as above
audio/vnd.nuera.ecelp9600	Same as above.	ecelp9600	Same as above
audio/vnd.octel.sbc	Variable-rate encoding averaging 18 kbps used for voice messaging in Lucent Technologies' Sierra™, Overture™, and IMA™ platforms.		Jeff Bouis *jbouis@lucent.com*
audio/vnd.qcelp	Qualcomm audio encoding.	qcp	Andy Dejaco *adejaco@qualcomm.com*
audio/vnd.rhetorex.32kadpcm	32-kbps Rhetorex™ ADPCM audio encoding used in voice messaging products such as Lucent Technologies's CallPerformer™, Unified Messenger™, and other products.		Jeff Bouis *jbouis@lucent.com*
audio/vnd.vmx.cvsd	Audio encoding used in voice messaging products including Lucent Technologies' Overture200™, Overture 300™, and VMX 300™ product lines.		Same as above
audio/x-aiff	AIFF audio file format.	aif, aiff, aifc	
audio/x-pn-realaudio	RealAudio metafile format by Real Networks (formerly Progressive Networks).	ram, rm	
audio/x-pn-realaudio-plugin	From Apache *mime.types*.	rpm	

Table D-4. "Audio" MIME types (continued)

MIME type	Description	Extension	Contact and reference
audio/x-realaudio	RealAudio audio format by Real Networks (formerly Progressive Networks).	ra	
audio/x-wav	WAV audio files.	wav	

chemical/*

Much of the information in Table D-5 was obtained courtesy of the "Chemical MIME Home Page" (*http://www.ch.ic.ac.uk/chemime/*).

Table D-5. "Chemical" MIME types

MIME type	Description	Extension	Contact and reference
chemical/x-alchemy	Alchemy format	alc	*http://www.camsoft.com*
chemical/x-cache-csf		csf	
chemical/x-cactvs-binary	CACTVS binary format	cbin	*http://cactvs.cit.nih.gov*
chemical/x-cactvs-ascii	CACTVS ASCII format	cascii	*http://cactvs.cit.nih.gov*
chemical/x-cactvs-table	CACTVS table format	ctab	*http://cactvs.cit.nih.gov*
chemical/x-cdx	ChemDraw eXchange file	cdx	*http://www.camsoft.com*
chemical/x-cerius	MSI Cerius II format	cer	*http://www.msi.com*
chemical/x-chemdraw	ChemDraw file	chm	*http://www.camsoft.com*
chemical/x-cif	Crystallographic Interchange Format	cif	*http://www.bernstein-plus-sons.com/software/rasmol/* *http://ndbserver.rutgers.edu/NDB/mmcif/examples/index.html*
chemical/x-mmcif	MacroMolecular CIF	mcif	Same as above
chemical/x-chem3d	Chem3D format	c3d	*http://www.camsoft.com*
chemical/x-cmdf	CrystalMaker Data Format	cmdf	*http://www.crystalmaker.co.uk*
chemical/x-compass	Compass program of the Takahashi	cpa	
chemical/x-crossfire	Crossfire file	bsd	
chemical/x-cml	Chemical Markup Language	cml	*http://www.xml-cml.org*
chemical/x-csml	Chemical Style Markup Language	csml, csm	*http://www.mdli.com*
chemical/x-ctx	Gasteiger group CTX file format	ctx	
chemical/x-cxf		cxf	
chemical/x-daylight-smiles	Smiles format	smi	*http://www.daylight.com/dayhtml/smiles/index.html*
chemical/x-embl-dl-nucleotide	EMBL nucleotide format	emb	*http://mercury.ebi.ac.uk*
chemical/x-galactic-spc	SPC format for spectral and chromatographic data	spc	*http://www.galactic.com/galactic/Data/spcvue.htm*

Table D-5. "Chemical" MIME types (continued)

MIME type	Description	Extension	Contact and reference
chemical/x-gamess-input	GAMESS Input format	inp, gam	*http://www.msg.ameslab.gov/GAMESS/Graphics/MacMolPlt.shtml*
chemical/x-gaussian-input	Gaussian Input format	gau	*http://www.mdli.com*
chemical/x-gaussian-checkpoint	Gaussian Checkpoint format	fch, fchk	*http://products.camsoft.com*
chemical/x-gaussian-cube	Gaussian Cube (Wavefunction) format	cub	*http://www.mdli.com*
chemical/x-gcg8-sequence		gcg	
chemical/x-genbank	ToGenBank format	gen	
chemical/x-isostar	IsoStar Library of intermolecular interactions	istr, ist	*http://www.ccdc.cam.ac.uk*
chemical/x-jcamp-dx	JCAMP Spectroscopic Data Exchange format	jdx, dx	*http://www.mdli.com*
chemical/x-jjc-review-surface	Re_View3 Orbital Contour files	rv3	*http://www.brunel.ac.uk/depts/chem/ch241s/re_view/rv3.htm*
chemical/x-jjc-review-xyz	Re_View3 Animation files	xyb	*http://www.brunel.ac.uk/depts/chem/ch241s/re_view/rv3.htm*
chemical/x-jjc-review-vib	Re_View3 Vibration files	rv2, vib	*http://www.brunel.ac.uk/depts/chem/ch241s/re_view/rv3.htm*
chemical/x-kinemage	Kinetic (Protein Structure) Images	kin	*http://www.faseb.org/protein/kinemages/MageSoftware.html*
chemical/x-macmolecule	MacMolecule file format	mcm	
chemical/x-macromodel-input	MacroModel Molecular Mechanics	mmd, mmod	*http://www.columbia.edu/cu/chemistry/*
chemical/x-mdl-molfile	MDL Molfile	mol	*http://www.mdli.com*
chemical/x-mdl-rdfile	Reaction data file	rd	*http://www.mdli.com*
chemical/x-mdl-rxnfile	MDL Reaction format	rxn	*http://www.mdli.com*
chemical/x-mdl-sdfile	MDL Structure data file	sd	*http://www.mdli.com*
chemical/x-mdl-tgf	MDL Transportable Graphics Format	tgf	*http://www.mdli.com*
chemical/x-mif		mif	
chemical/x-mol2	Portable representation of a SYBYL molecule	mol2	*http://www.tripos.com*
chemical/x-molconn-Z	Molconn-Z format	b	*http://www.eslc.vabiotech.com/molconn/molconnz.html*
chemical/x-mopac-input	MOPAC Input format	mop	*http://www.mdli.com*
chemical/x-mopac-graph	MOPAC Graph format	gpt	*http://products.camsoft.com*
chemical/x-ncbi-asn1		asn (old form)	
chemical/x-ncbi-asn1-binary		val	
chemical/x-pdb	Protein DataBank pdb	pdb	*http://www.mdli.com*

Table D-5. "Chemical" MIME types (continued)

MIME type	Description	Extension	Contact and reference
chemical/x-swissprot	SWISS-PROT protein sequence database	sw	*http://www.expasy.ch/spdbv/text/download.htm*
chemical/x-vamas-iso14976	Versailles Agreement on Materials and Standards	vms	*http://www.acolyte.co.uk/JISO/*
chemical/x-vmd	Visual Molecular Dynamics	vmd	*http://www.ks.uiuc.edu/Research/vmd/*
chemical/x-xtel	Xtelplot file format	xtel	*http://www.recipnet.indiana.edu/graphics/xtelplot/xtelplot.htm*
chemical/x-xyz	Co-ordinate Animation format	xyz	*http://www.mdli.com*

image/*

Table D-6 summarizes some of the image types commonly exchanged by email and HTTP.

Table D-6. "Image" MIME types

MIME type	Description	Extension	Contact and reference
image/bmp	Windows BMP image format.	bmp	
image/cgm	Computer Graphics Metafile (CGM) is an International Standard for the portable storage and transfer of 2-D illustrations.		Alan Francis *A.H.Francis@open.ac.uk* See ISO 8632:1992, IS 8632:1992 Amendment 1 (1994), and IS 8632: 1992 Amendment 2 (1995)
image/g3fax	G3 Facsimile byte streams.		RFC 1494
image/gif	Compuserve GIF images.	gif	RFC 1341
image/ief		ief	RFC 1314
image/jpeg	JPEG images.	jpeg, jpg, jpe, jfif	JPEG Draft Standard ISO 10918-1 CD
image/naplps	North American Presentation Layer Protocol Syntax (NAPLPS) images.		ANSI X3.110-1983 CSA T500-1983
image/png	Portable Network Graphics (PNG) images.	png	Internet draft *draft-boutell-png-spec-04.txt*, "Png (Portable Network Graphics) Specification Version 1.0"
image/prs.btif	Format used by Nations Bank for BTIF image viewing of checks and other applications.	btif, btf	Arthur Rubin *arthurr@crt.com*
image/prs.pti	PTI encoded images.	pti	Juern Laun *juern.laun@gmx.de* *http://server.hvzgymn.wn.schule-bw.de/pti/*
image/tiff	TIFF images.	tiff, tif	RFC 2302

Table D-6. "Image" MIME types (continued)

MIME type	Description	Extension	Contact and reference
image/vnd.cns.inf2	Supports application features available on the TRILOGUE Infinity network services platform from Comverse Network Systems.		Ann McLaughlin Comverse Network Systems *amclaughlin@comversens.com*
image/vnd.dxf	DXF vector CAD files.	dxf	
image/vnd.fastbidsheet	A FastBid Sheet contains a raster or vector image that represents an engineering or architectural drawing.	fbs	Scott Becker *scottb@bxwa.com*
image/vnd.fpx	Kodak FlashPix images.	fpx	Chris Wing *format_change_request@kodak.com* *http://www.kodak.com*
image/vnd.fst	Image format from FAST Search and Transfer.	fst	Arild Fuldseth *Arild.Fuldseth@fast.no*
image/vnd.fujixerox.edmics-mmr	Fuji Xerox EDMICS MMR image format.	mmr	Masanori Onda *Masanori.Onda@fujixerox.co.jp*
image/vnd.fujixerox.edmics-rlc	Fuji Xerox EDMICS RLC image format.	rlc	Same as above
image/vnd.mix	MIX files contain binary data in streams that are used to represent images and related information. They are used by Microsoft PhotDraw and PictureIt software.		Saveen Reddy2 *saveenr@microsoft.com*
image/vnd.net-fpx	Kodak FlashPix images.		Chris Wing *format_change_request@kodak.com* *http://www.kodak.com*
image/vnd.wap.wbmp	From Apache *mime.types*.	wbmp	
image/vnd.xiff	Extended Image Format used by Pagis software.	xif	Steve Martin *smartin@xis.xerox.com*
image/x-cmu-raster	From Apache *mime.types*.	ras	
image/x-portable-anymap	PBM generic images.	pnm	Jeff Poskanzer *http://www.acme.com/software/pbmplus/*
image/x-portable-bitmap	PBM bitmap images.	pbm	Same as above
image/x-portable-graymap	PBM grayscale images.	pgm	Same as above
image/x-portable-pixmap	PBM color images.	ppm	Same as above
image/x-rgb	Silicon Graphics's RGB images.	rgb	
image/x-xbitmap	X-Window System bitmap images.	xbm	
image/x-xpixmap	X-Window System color images.	xpm	
image/x-xwindowdump	X-Window System screen capture images.	xwd	

message/*

Messages are composite types used to communicate data objects (through email, HTTP, or other transport protocols). Table D-7 describes the common MIME message types.

Table D-7. "Message" MIME types

MIME type	Description	Extension	Contact and reference
message/delivery-status			
message/disposition-notification			RFC 2298
message/external-body			RFC 1341
message/http			RFC 2616
message/news	Defines a way to transmit news articles via email for human reading—message/rfc822 is not sufficient because news headers have semantics beyond those defined by RFC 822.		RFC 1036
message/partial	Permits the fragmented transmission of bodies that are thought to be too large to be sent directly by email.		RFC 1341
message/rfc822	A complete email message.		RFC 1341
message/s-http	Secure HTTP messages, an alternative to HTTP over SSL.		RFC 2660

model/*

The model MIME type is an IETF-registered extension type. It represents mathematical models of physical worlds, for computer-aided design, and 3-D graphics. Table D-8 describes some of the model formats.

Table D-8. "Model" MIME types

MIME type	Description	Extension	Contact and reference
model/iges	The Initial Graphics Exchange Specification (IGES) defines a neutral data format that allows for the digital exchange of information between computer-aided design (CAD) systems.	igs, iges	RFC 2077
model/mesh		msh, mesh, silo	RFC 2077
model/vnd.dwf	DWF CAD files.	dwf	Jason Pratt *jason.pratt@autodesk.com*
model/vnd.flatland.3dml	Supports 3DML models supported by Flatland products.	3dml, 3dm	Michael Powers *pow@flatland.com* *http://www.flatland.com*

Table D-8. "Model" MIME types (continued)

MIME type	Description	Extension	Contact and reference
model/vnd.gdl model/vnd.gs-gdl	The Geometric Description Language (GDL) is a parametric object definition language for ArchiCAD by Graphisoft.	gdl, gsm, win, dor, lmp, rsm, msm, ism	Attila Babits *ababits@graphisoft.hu* *http://www.graphisoft.com*
model/vnd.gtw	Gen-Trix models.	gtw	Yutaka Ozaki *yutaka_ozaki@gen.co.jp*
model/vnd.mts	MTS model format by Virtue.	mts	Boris Rabinovitch *boris@virtue3d.com*
model/vnd.parasolid.transmit.binary	Binary Parasolid modeling file.	x_b	*http://www.ugsolutions.com/products/parasolid/*
model/vnd.parasolid.transmit.text	Text Parasolid modeling file.	x_t	*http://www.ugsolutions.com/products/parasolid/*
model/vnd.vtu	VTU model format by Virtue.	vtu	Boris Rabinovitch *boris@virtue3d.com*
model/vrml	Virtual Reality Markup Language format files.	wrl, vrml	RFC 2077

multipart/*

Multipart MIME types are composite objects that contain other objects. The subtype describes the implementation of the multipart packaging and how to process the components. Multipart media types are summarized in Table D-9.

Table D-9. "Multipart" MIME types

MIME type	Description	Extension	Contact and reference
multipart/alternative	The content consists of a list of alternative representations, each with its own Content-Type. The client can select the best supported component.		RFC 1341
multipart/appledouble	Apple Macintosh files contain "resource forks" and other desktop data that describes the actual file contents. This multipart content sends the Apple metadata in one part and the actual content in another part.		*http://www.isi.edu/in-notes/iana/assignments/media-types/multipart/appledouble*
multipart/byteranges	When an HTTP message includes the content of multiple ranges, these are transmitted in a "multipart/byteranges" object. This media type includes two or more parts, separated by MIME boundaries, each with its own Content-Type and Content-Range fields.		RFC 2068

Table D-9. "Multipart" MIME types (continued)

MIME type	Description	Extension	Contact and reference
multipart/digest	Contains a collection of individual email messages, in an easy-to-read form.		RFC 1341
multipart/encrypted	Uses two parts to support cryptographically encrypted content. The first part contains the control information necessary to decrypt the data in the second body part and is labeled according to the value of the protocol parameter. The second part contains the encrypted data in type application/octet-stream.		RFC 1847
multipart/form-data	Used to bundle up a set of values as the result of a user filling out a form.		RFC 2388
multipart/header-set	Separates user data from arbitrary descriptive metadata.		*http://www.isi.edu/in-notes/iana/assignments/media-types/multipart/header-set*
multipart/mixed	A collection of objects.		RFC 1341
multipart/parallel	Syntactically identical to multipart/mixed, but all of the parts are intended to be presented simultaneously, on systems capable of doing so.		RFC 1341
multipart/related	Intended for compound objects consisting of several interrelated body parts. The relationships between the body parts distinguish them from other object types. These relationships often are represented by links internal to the object's components that reference the other components.		RFC 2387
multipart/report	Defines a general container type for electronic mail reports of any kind.		RFC 1892
multipart/signed	Uses two parts to support cryptographically signed content. The first part is the content, including its MIME headers. The second part contains the information necessary to verify the digital signature.		RFC 1847
multipart/voice-message	Provides a mechanism for packaging a voice message into one container that is tagged as VPIM v2–compliant.		RFCs 2421 and 2423

text/*

Text media types contain characters and potential formatting information. Table D-10 summarizes text MIME types.

Table D-10. "Text" MIME types

MIME type	Description	Extension	Contact and reference
text/calendar	Supports the iCalendar calendaring and scheduling standard.		RFC 2445
text/css	Cascading Style Sheets.	css	RFC 2318
text/directory	Holds record data from a directory database, such as LDAP.		RFC 2425
text/enriched	Simple formatted text, supporting fonts, colors, and spacing. SGML-like tags are used to begin and end formatting.		RFC 1896
text/html	HTML file.	html, htm	RFC 2854
text/parityfec	Forward error correction for text streamed in an RTP stream.		RFC 3009
text/plain	Plain old text.	asc, txt	
text/prs.lines.tag	Supports tagged forms, as used for email registration.	tag, dsc	John Lines *john@paladin.demon.co.uk* *http://www.paladin.demon.co.uk/tag-types/*
text/rfc822-headers	Used to bundle a set of email headers, such as when sending mail failure reports.		RFC 1892
text/richtext	Older form of enriched text. See text/enriched.	rtx	RFC 1341
text/rtf	The Rich Text Format (RTF) is a method of encoding formatted text and graphics for transfer between applications. The format is widely supported by word-processing applications on the MS-DOS, Windows, OS/2, and Macintosh platforms.	rtf	
text/sgml	SGML markup files.	sgml, sgm	RFC 1874
text/t140	Supports standardized T.140 text, as used in synchronized RTP multimedia.		RFC 2793
text/tab-separated-values	TSV is a popular method of data interchange among databases and spreadsheets and word processors. It consists of a set of lines, with fields separated by tab characters.	tsv	*http://www.isi.edu/in-notes/iana/assignments/media-types/text/tab-separated-values*
text/uri-list	Simple, commented lists of URLs and URNs used by URN resolvers, and any other applications that need to communicate bulk URI lists.	uris, uri	RFC 2483

Table D-10. "Text" MIME types (continued)

MIME type	Description	Extension	Contact and reference
text/vnd.abc	ABC files are a human-readable format for musical scores.	abc	*http://www.gre.ac.uk/~c.walshaw/abc/* *http://home1.swipnet.se/~w-11382/abcbnf.htm*
text/vnd.curl	Provides a set of content definition languages interpreted by the CURL runtime plug-in.	curl	Tim Hodge *thodge@curl.com*
text/vnd.DMClientScript	CommonDM Client Script files are used as hyperlinks to non-http sites (such as BYOND, IRC, or telnet) accessed by the Dream Seeker client application.	dms	Dan Bradley *dan@dantom.com* *http://www.byond.com/code/ref/*
text/vnd.fly	Fly is a text preprocessor that uses a simple syntax to create an interface between databases and web pages.	fly	John-Mark Gurney *jmg@flyidea.com* *http://www.flyidea.com*
text/vnd.fmi.flexstor	For use in the SUVDAMA and UVRAPPF projects.	flx	*http://www.ozone.fmi.fi/SUVDAMA/* *http://www.ozone.fmi.fi/UVRAPPF/*
text/vnd.in3d.3dml	For In3D Player.	3dml, 3dm	Michael Powers *powers@insideout.net*
text/vnd.in3d.spot	For In3D Player.	spot, spo	Same as above
text/vnd.IPTC.NewsML	NewsML format specified by the International Press Telecommunications Council (IPTC).	xml	David Allen *m_director_iptc@dial.pipex.com* *http://www.iptc.org*
text/vnd.IPTC.NITF	NITF format specified by the IPTC.	xml	Same as above *http://www.nitf.org*
text/vnd.latex-z	Supports LaTeX documents containing Z notation. Z notation (pronounced "zed"), is based on Zermelo-Fraenkel set theory and first order predicate logic, and it is useful for describing computer systems.		*http://www.comlab.ox.ac.uk/archive/z/*
text/vnd.motorola.reflex	Provides a common method for submitting simple text messages from ReFLEX™ wireless devices.		Mark Patton *fmp014@email.mot.com* Part of the FLEXsuite™ of Enabling Protocols specification available from Motorola under the licensing agreement
text/vnd.ms-mediapackage	This type is intended to be handled by the Microsoft application programs MStore.exe and 7 storDB.exe.	mpf	Jan Nelson *jann@microsoft.com*

Table D-10. "Text" MIME types (continued)

MIME type	Description	Extension	Contact and reference
text/vnd.wap.si	Service Indication (SI) objects contain a message describing an event and a URI describing where to load the corresponding service.	si, xml	WAP Forum Ltd *http://www.wapforum.org*
text/vnd.wap.sl	The Service Loading (SL) content type provides a means to convey a URI to a user agent in a mobile client. The client itself automatically loads the content indicated by that URI and executes it in the addressed user agent without user intervention when appropriate.	sl, xml	Same as above
text/vnd.wap.wml	Wireless Markup Language (WML) is a markup language, based on XML, that defines content and user interface for narrow-band devices, including cellular phones and pagers.	wml	Same as above
text/vnd.wap.wmlscript	WMLScript is an evolution of JavaScript for wireless devices.	wmls	Same as above
text/x-setext	From Apache *mime.types*.	etx	
text/xml	Extensible Markup Language format file (use application/xml if you want the browser to save to file when downloaded).	xml	RFC 2376

video/*

Table D-11 lists some popular video movie formats. Note that some video formats are classified as application types.

Table D-11. "Video" MIME types

MIME type	Description	Extension	Contact and reference
video/MP4V-ES	MPEG-4 video payload, as carried by RTP.		RFC 3016
video/mpeg	Video encoded per the ISO 11172 CD MPEG standard.	mpeg, mpg, mpe	RFC 1341
video/parityfec	Forward error correcting video format for data carried through RTP streams.		RFC 3009
video/pointer	Transporting pointer position information for presentations.		RFC 2862
video/quicktime	Apple Quicktime video format.	qt, mov	*http://www.apple.com*
video/vnd.fvt	Video format from FAST Search & Transfer.	fvt	Arild Fuldseth *Arild.Fuldseth@fast.no*

Table D-11. "Video" MIME types (continued)

MIME type	Description	Extension	Contact and reference
video/vnd.motorola.video video/vnd.motorola.videop	Proprietary formats used by products from Motorola ISG.		Tom McGinty Motorola ISG *tmcginty@dma.isg.mot*
video/vnd.mpegurl	This media type consists of a series of URLs of MPEG Video files.	mxu	Heiko Recktenwald *uzs106@uni-bonn.de* "Power and Responsibility: Conversations with Contributors," Guy van Belle, et al., LMJ 9 (1999), 127–133, 129 (MIT Press)
video/vnd.nokia.interleaved-multimedia	Used in Nokia 9210 Communicator video player and related tools.	nim	Petteri Kangaslampi *petteri.kangaslampi@nokia.com*
video/x-msvideo	Microsoft AVI movies.	avi	*http://www.microsoft.com*
video/x-sgi-movie	Silicon Graphics's movie format.	movie	*http://www.sgi.com*

Experimental Types

The set of primary types supports most content types. Table D-12 lists one experimental type, for conferencing software, that is configured in some web servers.

Table D-12. Extension MIME types

MIME type	Description	Extension	Contact and reference
x-conference/x-cooltalk	Collaboration tool from Netscape	ice	

APPENDIX E

Base-64 Encoding

Base-64 encoding is used by HTTP, for basic and digest authentication, and by several HTTP extensions. This appendix explains base-64 encoding and provides conversion tables and pointers to Perl software to help you correctly use base-64 encoding in HTTP software.

Base-64 Encoding Makes Binary Data Safe

The base-64 encoding converts a series of arbitrary bytes into a longer sequence of common text characters that are all legal header field values. Base-64 encoding lets us take user input or binary data, pack it into a safe format, and ship it as HTTP header field values without fear of them containing colons, newlines, or binary values that would break HTTP parsers.

Base-64 encoding was developed as part of the MIME multimedia electronic mail standard, so MIME could transport rich text and arbitrary binary data between different legacy email gateways.* Base-64 encoding is similar in spirit, but more efficient in space, to the uuencode and BinHex standards for textifying binary data. Section 6.8 of MIME RFC 2045 details the base-64 algorithm.

Eight Bits to Six Bits

Base-64 encoding takes a sequence of 8-bit bytes, breaks the sequence into 6-bit pieces, and assigns each 6-bit piece to one of 64 characters comprising the base-64 alphabet. The 64 possible output characters are common and safe to place in HTTP header fields. The 64 characters include upper- and lowercase letters, numbers, +,

* Some mail gateways would silently strip many "non-printing" characters with ASCII values between 0 and 31. Other programs would interpret some bytes as flow control characters or other special control characters, or convert carriage returns to line feeds and the like. Some programs would experience fatal errors upon receiving international characters with a value above 127 because the software was not "8-bit clean."

and /. The special character = also is used. The base-64 alphabet is shown in Table E-1.

Note that because the base-64 encoding uses 8-bit characters to represent 6 bits of information, base 64–encoded strings are about 33% larger than the original values.

Table E-1. Base-64 alphabet

0	A	8	I	16	Q	24	Y	32	g	40	o	48	w	56	4
1	B	9	J	17	R	25	Z	33	h	41	p	49	x	57	5
2	C	10	K	18	S	26	a	34	i	42	q	50	y	58	6
3	D	11	L	19	T	27	b	35	j	43	r	51	z	59	7
4	E	12	M	20	U	28	c	36	k	44	s	52	0	60	8
5	F	13	N	21	V	29	d	37	l	45	t	53	1	61	9
6	G	14	O	22	W	30	e	38	m	46	u	54	2	62	+
7	H	15	P	23	X	31	f	39	n	47	v	55	3	63	/

Figure E-1 shows a simple example of base-64 encoding. Here, the three-character input value "Ow!" is base 64–encoded, resulting in the four-character base 64–encoded value "T3ch". It works like this:

1. The string "Ow!" is broken into 3 8-bit bytes (0x4F, 0x77, 0x21).
2. The 3 bytes create the 24-bit binary value 010011110111011100100001.
3. These bits are segmented into the 6-bit sequences 010011, 110111, 01110, 100001.
4. Each of these 6-bit values represents a number from 0 to 63, corresponding to one of 64 characters in the base-64 alphabet. The resulting base 64–encoded string is the 4-character string "T3ch", which can then be sent across the wire as "safe" 8-bit characters, because only the most portable characters are used (letters, numbers, etc.).

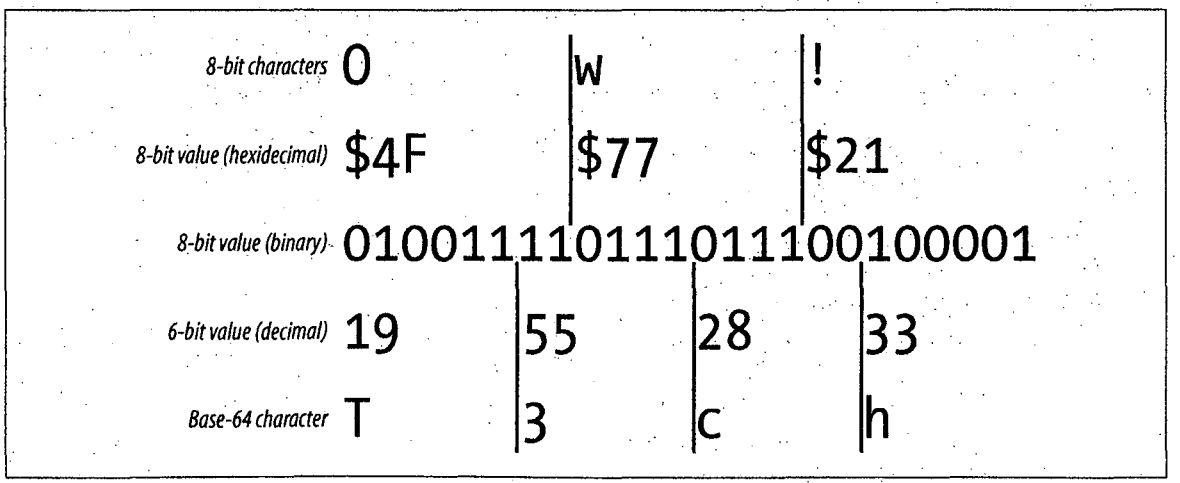


Figure E-1. Base-64 encoding example

Base-64 Padding

Base-64 encoding takes a sequence of 8-bit bytes and segments the bit stream into 6-bit chunks. It is unlikely that the sequence of bits will divide evenly into 6-bit pieces. When the bit sequence does not divide evenly into 6-bit pieces, the bit sequence is padded with zero bits at the end to make the length of the bit sequence a multiple of 24 (the least common multiple of 6 and 8 bits).

When encoding the padded bit string, any group of 6 bits that is completely padding (containing no bits from the original data) is represented by a special 65th symbol: "=". If a group of 6 bits is partially padded, the padding bits are set to zero.

Table E-2 shows examples of padding. The initial input string "a:a" is 3 bytes long, or 24 bits. 24 is a multiple of 6 and 8, so no padding is required. The resulting base 64–encoded string is "YTph".

Table E-2. Base-64 padding examples

Input data	Binary sequence (padding noted as "x")	Encoded data
a:a	011000 010011 101001 100001	YTph
a:aa	011000 010011 101001 100001 011000 01xxxx xxxxxx xxxxxx	YTphYQ==
a:aaa	011000 010011 101001 100001 011000 010110 0001xx xxxxxx	YTphYWE=
a:aaaa	011000 010011 101001 100001 011000 010110 000101 100001	YTphYWFh

However, when another character is added, the input string grows to 32 bits long. The next smallest multiple of 6 and 8 is 48 bits, so 16 bits of padding are added. The first 4 bits of padding are mixed with data bits. The resulting 6-bit group, 01xxxx, is treated as 010000, 16 decimal, or base-64 encoding Q. The remaining two 6-bit groups are all padding and are represented by "=".

Perl Implementation

MIME::Base64 is a Perl module for base-64 encoding and decoding. You can read about this module at *http://www.perldoc.com/perl5.6.1/lib/MIME/Base64.html*.

You can encode and decode strings using the MIME::Base64 encode_base64 and decode_base64 methods:

```
use MIME::Base64;

$encoded = encode_base64('Aladdin:open sesame');
$decoded = decode_base64($encoded);
```

For More Information

For more information on base-64 encoding, see:

http://www.ietf.org/rfc/rfc2045.txt
: Section 6.8 of RFC 2045, "MIME Part 1: Format of Internet Message Bodies," provides an official specification of base-64 encoding.

http://www.perldoc.com/perl5.6.1/lib/MIME/Base64.html
: This web site contains documentation for the MIME::Base64 Perl module that provides encoding and decoding of base-64 strings.

APPENDIX F
Digest Authentication

This appendix contains supporting data and source code for implementing HTTP digest authentication facilities.

Digest WWW-Authenticate Directives

WWW-Authenticate directives are described in Table F-1, paraphrased from the descriptions in RFC 2617. As always, refer to the official specifications for the most up-to-date details.

Table F-1. Digest WWW-Authenticate header directives (from RFC 2617)

Directive	Description
realm	A string to be displayed to users so they know which username and password to use. This string should contain at least the name of the host performing the authentication and might additionally indicate the collection of users who might have access. An example might be "registered_users@gotham.news.com".
nonce	A server-specified data string that should be uniquely generated each time a 401 response is made. It is recommended that this string be base-64 or hexadecimal data. Specifically, because the string is passed in the header lines as a quoted string, the double-quote character is not allowed. The contents of the nonce are implementation-dependent. The quality of the implementation depends on a good choice. A nonce might, for example, be constructed as the base-64 encoding of: `time-stamp H(time-stamp ":" ETag ":" private-key)` where *time-stamp* is a server-generated time or other nonrepeating value, *ETag* is the value of the HTTP ETag header associated with the requested entity, and *private-key* is data known only to the server. With a nonce of this form, a server would recalculate the hash portion after receiving the client Authentication header and reject the request if it did not match the nonce from that header or if the time-stamp value is not recent enough. In this way, the server can limit the time of the nonce's validity. The inclusion of the ETag prevents a replay request for an updated version of the resource. (Note: including the IP address of the client in the nonce appears to offer the server the ability to limit the reuse of the nonce to the same client that originally got it. However, that would break proxy farms, where requests from a single user often go through different proxies in the farm. Also, IP address spoofing is not that hard.) An implementation might choose not to accept a previously used nonce or a previously used digest, to protect against replay attacks, or it might choose to use one-time nonces or digests for POST or PUT requests and time-stamps for GET requests.

Table F-1. Digest WWW-Authenticate header directives (from RFC 2617) (continued)

Directive	Description
domain	A quoted, space-separated list of URIs (as specified in RFC 2396, "Uniform Resource Identifiers: Generic Syntax") that define the protection space. If a URI is an abs_path, it is relative to the canonical root URL of the server being accessed. An absolute URI in this list may refer to a different server than the one being accessed. The client can use this list to determine the set of URIs for which the same authentication information may be sent: any URI that has a URI in this list as a prefix (after both have been made absolute) may be assumed to be in the same protection space. If this directive is omitted or its value is empty, the client should assume that the protection space consists of all URIs on the responding server. This directive is not meaningful in Proxy-Authenticate headers, for which the protection space is always the entire proxy; if present, it should be ignored.
opaque	A string of data, specified by the server, that should be returned by the client unchanged in the Authorization header of subsequent requests with URIs in the same protection space. It is recommended that this string be base-64 or hexadecimal data.
stale	A flag indicating that the previous request from the client was rejected because the nonce value was stale. If stale is TRUE (case-insensitive), the client may want to retry the request with a new encrypted response, without reprompting the user for a new username and password. The server should set stale to TRUE only if it receives a request for which the nonce is invalid but has a valid digest (indicating that the client knows the correct username/password). If stale is FALSE, or anything other than TRUE, or the stale directive is not present, the username and/or password are invalid, and new values must be obtained.

algorithm	A string indicating a pair of algorithms used to produce the digest and a checksum. If this is not present, it is assumed to be "MD5". If the algorithm is not understood, the challenge should be ignored (and a different one used, if there is more than one).
	In this document, the string obtained by applying the digest algorithm to the data "data" with secret "secret" will be denoted by "KD(secret, data)", and the string obtained by applying the checksum algorithm to the data "data" will be denoted "H(data)". The notation "unq(X)" means the value of the quoted string "X" without the surrounding quotes.
	For the MD5 and MD5-sess algorithms:
	`H(data) = MD5(data)`
	`HD(secret, data) = H(concat(secret, ":", data))`
	I.e., the digest is the MD5 of the secret concatenated with a colon concatenated with the data. The MD5-sess algorithm is intended to allow efficient third-party authentication servers.

qop	This directive is optional but is made so only for backward compatibility with RFC 2069 [6]; it should be used by all implementations compliant with this version of the digest scheme. If present, it is a quoted string of one or more tokens indicating the "quality of protection" values supported by the server. The value "auth" indicates authentication; the value "auth-int" indicates authentication with integrity protection. Unrecognized options must be ignored.
<extension>	This directive allows for future extensions. Any unrecognized directives must be ignored.

Digest Authorization Directives

Each of the Authorization directives is described in Table F-2, paraphrased from the descriptions in RFC 2617. Refer to the official specifications for the most up-to-date details.

Table F-2. Digest Authorization header directives (from RFC 2617)

Directive	Description
username	The user's name in the specified realm.
realm	The realm passed to the client in the WWW-Authenticate header.
nonce	The same nonce passed to the client in the WWW-Authenticate header.
uri	The URI from the request URI of the request line; duplicated because proxies are allowed to change the request line in transit, and we may need the original URI for proper digest verification calculations.
response	This is the actual digest—the whole point of digest authentication! The response is a string of 32 hexadecimal digits, computed by a negotiated digest algorithm, which proves that the user knows the password.
algorithm	A string indicating a pair of algorithms used to produce the digest and a checksum. If this is not present, it is assumed to be "MD5".
opaque	A string of data, specified by the server in a WWW-Authenticate header, that should be returned by the client unchanged in the Authorization header of subsequent requests with URIs in the same protection space.
cnonce	This must be specified if a qop directive is sent and must not be specified if the server did not send a qop directive in the WWW-Authenticate header field. The cnonce value is an opaque quoted string value provided by the client and used by both client and server to avoid chosen plaintext attacks, to provide mutual authentication, and to provide some message-integrity protection. See the descriptions of the response-digest and request-digest calculations later in this appendix.
qop	Indicates what "quality of protection" the client has applied to the message. If present, its value must be one of the alternatives the server indicated it supports in the WWW-Authenticate header. These values affect the computation of the request digest. This is a single token, not a quoted list of alternatives, as in WWW-Authenticate. This directive is optional, to preserve backward compatibility with a minimal implementation of RFC 2069, but it should be used if the server indicated that qop is supported by providing a qop directive in the WWW-Authenticate header field.
nc	This must be specified if a qop directive is sent and must not be specified if the server did not send a qop directive in the WWW-Authenticate header field. The value is the hexadecimal count of the number of requests (including the current request) that the client has sent with the nonce value in this request. For example, in the first request sent in response to a given nonce value, the client sends nc="00000001". The purpose of this directive is to allow the server to detect request replays by maintaining its own copy of this count—if the same nc value is seen twice, the request is a replay.
<extension>	This directive allows for future extensions. Any unrecognized directive must be ignored.

Digest Authentication-Info Directives

Each of the Authentication-Info directives is described in Table F-3, paraphrased from the descriptions in RFC 2617. Refer to the official specifications for the most up-to-date details.

Table F-3. Digest Authentication-Info header directives (from RFC 2617)

Directive	Description
nextnonce	The value of the nextnonce directive is the nonce the server wants the client to use for a future authentication response. The server may send the Authentication-Info header with a nextnonce field as a means of implementing one-time or otherwise changing nonces. If the nextnonce field is present the client should use it when constructing the Authorization header for its next request. Failure of the client to do so may result in a reauthentication request from the server with "stale=TRUE". Server implementations should carefully consider the performance implications of the use of this mechanism; pipelined requests will not be possible if every response includes a nextnonce directive that must be used on the next request received by the server. Consideration should be given to the performance versus security trade-offs of allowing an old nonce value to be used for a limited time to permit request pipelining. Use of the nonce count can retain most of the security advantages of a new server nonce without the deleterious effects on pipelining.
qop	Indicates the "quality of protection" options applied to the response by the server. The value "auth" indicates authentication; the value "auth-int" indicates authentication with integrity protection. The server should use the same value for the qop directive in the response as was sent by the client in the corresponding request.

rspauth	The optional response digest in the "response auth" directive supports mutual authentication—the server proves that it knows the user's secret, and, with qop="auth-int", it also provides limited integrity protection of the response. The "response-digest" value is calculated as for the "request-digest" in the Authorization header, except that if qop="auth" or qop is not specified in the Authorization header for the request, A2 is:
	`A2 = ":" digest-uri-value`
	and if qop="auth-int", A2 is:
	`A2 = ":" digest-uri-value ":" H(entity-body)`
	where *digest-uri-value* is the value of the uri directive on the Authorization header in the request. The cnonce and nc values must be the same as the ones in the client request to which this message is a response. The rspauth directive must be present if qop="auth" or qop="auth-int" is specified.

cnonce	The cnonce value must be the same as the one in the client request to which this message is a response. The cnonce directive must be present if qop="auth" or qop="auth-int" is specified.
nc	The nc value must be the same as the one in the client request to which this message is a response. The nc directive must be present if qop="auth" or qop="auth-int" is specified.
<extension>	This directive allows for future extensions. Any unrecognized directive must be ignored.

Reference Code

The following code implements the calculations of H(A1), H(A2), request-digest, and response-digest, from RFC 2617. It uses the MD5 implementation from RFC 1321.

File "digcalc.h"

```
#define HASHLEN 16
typedef char HASH[HASHLEN];
#define HASHHEXLEN 32
typedef char HASHHEX[HASHHEXLEN+1];
#define IN
```

```
#define OUT
/* calculate H(A1) as per HTTP Digest spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    );

/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1,           /* H(A1) */
    IN char * pszNonce,       /* nonce from server */
    IN char * pszNonceCount,  /* 8 hex digits */
    IN char * pszCNonce,      /* client nonce */
    IN char * pszQop,         /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod,      /* method from the request */
    IN char * pszDigestUri,   /* requested URL */
    IN HASHHEX HEntity,       /* H(entity body) if qop="auth-int" */
    OUT HASHHEX Response      /* request-digest or response-digest */
    );
```

File "digcalc.c"

```
#include <global.h>
#include <md5.h>
#include <string.h>
#include "digcalc.h"

void CvtHex(
    IN HASH Bin,
    OUT HASHHEX Hex
    )
{
    unsigned short i;
    unsigned char j;
    for (i = 0; i < HASHLEN; i++) {
        j = (Bin[i] >> 4) & 0xf;
        if (j <= 9)
            Hex[i*2] = (j + '0');
         else
            Hex[i*2] = (j + 'a' - 10);
        j = Bin[i] & 0xf;
        if (j <= 9)
            Hex[i*2+1] = (j + '0');
         else
            Hex[i*2+1] = (j + 'a' - 10);
    };
    Hex[HASHHEXLEN] = '\0';
};
```

```
/* calculate H(A1) as per spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    )
{
      MD5_CTX Md5Ctx;
      HASH HA1;
      MD5Init(&Md5Ctx);
      MD5Update(&Md5Ctx, pszUserName, strlen(pszUserName));
      MD5Update(&Md5Ctx, ":", 1);
      MD5Update(&Md5Ctx, pszRealm, strlen(pszRealm));
      MD5Update(&Md5Ctx, ":", 1);
      MD5Update(&Md5Ctx, pszPassword, strlen(pszPassword));
      MD5Final(HA1, &Md5Ctx);
      if (stricmp(pszAlg, "md5-sess") == 0) {
            MD5Init(&Md5Ctx);
            MD5Update(&Md5Ctx, HA1, HASHLEN);
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
            MD5Final(HA1, &Md5Ctx);
      };
      CvtHex(HA1, SessionKey);
};
/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1,           /* H(A1) */
    IN char * pszNonce,       /* nonce from server */
    IN char * pszNonceCount,  /* 8 hex digits */
    IN char * pszCNonce,      /* client nonce */
    IN char * pszQop,         /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod,      /* method from the request */
    IN char * pszDigestUri,   /* requested URL */
    IN HASHHEX HEntity,       /* H(entity body) if qop="auth-int" */
    OUT HASHHEX Response      /* request-digest or response-digest */
    )
{
      MD5_CTX Md5Ctx;
      HASH HA2;
      HASH RespHash;
       HASHHEX HA2Hex;
      // calculate H(A2)
      MD5Init(&Md5Ctx);
      MD5Update(&Md5Ctx, pszMethod, strlen(pszMethod));
      MD5Update(&Md5Ctx, ":", 1);
      MD5Update(&Md5Ctx, pszDigestUri, strlen(pszDigestUri));
      if (stricmp(pszQop, "auth-int") == 0) {
```

```
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, HEntity, HASHHEXLEN);
        };
        MD5Final(HA2, &Md5Ctx);
         CvtHex(HA2, HA2Hex);
        // calculate response
        MD5Init(&Md5Ctx);
        MD5Update(&Md5Ctx, HA1, HASHHEXLEN);
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
        MD5Update(&Md5Ctx, ":", 1);
        if (*pszQop) {
            MD5Update(&Md5Ctx, pszNonceCount, strlen(pszNonceCount));
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, pszQop, strlen(pszQop));
            MD5Update(&Md5Ctx, ":", 1);
        };
        MD5Update(&Md5Ctx, HA2Hex, HASHHEXLEN);
        MD5Final(RespHash, &Md5Ctx);
        CvtHex(RespHash, Response);
};
```

File "digtest.c"

```
#include <stdio.h>
#include "digcalc.h"

void main(int argc, char ** argv) {
        char * pszNonce = "dcd98b7102dd2f0e8b11d0f600bfb0c093";
        char * pszCNonce = "0a4f113b";
        char * pszUser = "Mufasa";
        char * pszRealm = "testrealm@host.com";
        char * pszPass = "Circle Of Life";
        char * pszAlg = "md5";
        char szNonceCount[9] = "00000001";
        char * pszMethod = "GET";
        char * pszQop = "auth";
        char * pszURI = "/dir/index.html";
        HASHHEX HA1;
        HASHHEX HA2 = "";
        HASHHEX Response;
        DigestCalcHA1(pszAlg, pszUser, pszRealm, pszPass,
            pszNonce, pszCNonce, HA1);
        DigestCalcResponse(HA1, pszNonce, szNonceCount, pszCNonce, pszQop,
            pszMethod, pszURI, HA2, Response);
        printf("Response = %s\n", Response);
};
```

APPENDIX G

Language Tags

Language tags are short, standardized strings that name spoken languages—for example, "fr" (French) and "en-GB" (Great Britain English). Each tag has one or more parts, separated by hyphens, called *subtags*. Language tags were described in detail in the section "Language Tags and HTTP" in Chapter 16.

This appendix summarizes the rules, standardized tags, and registration information for language tags. It contains the following reference material:

- Rules for the first (primary) subtag are summarized in "First Subtag Rules."
- Rules for the second subtag are summarized in "Second Subtag Rules."
- IANA-registered language tags are shown in Table G-1.
- ISO 639 language codes are shown in Table G-2.
- ISO 3166 country codes are shown in Table G-3.

First Subtag Rules

If the first subtag is:

- Two characters long, it's a language code from the ISO 639* and 639-1 standards
- Three characters long, it's a language code listed in the ISO 639-2† standard
- The letter "i," the language tag is explicitly IANA-registered
- The letter "x," the language tag is a private, nonstandard, extension subtag

The ISO 639 and 639-2 names are summarized in Table G-2.

* See ISO standard 639, "Codes for the representation of names of languages."

† See ISO 639-2, "Codes for the representation of names of languages—Part 2: Alpha-3 code."

Second Subtag Rules

If the second subtag is:

- Two characters long, it's a country/region defined by ISO 3166*
- Three to eight characters long, it may be registered with the IANA
- One character long, it is illegal

The ISO 3166 country codes are summarized in Table G-3.

IANA-Registered Language Tags

Table G-1. Language tags

IANA language tag	Description
i-bnn	Bunun
i-default	Default language context
i-hak	Hakka
i-klingon	Klingon
i-lux	Luxembourgish
i-mingo	Mingo
i-navajo	Navajo
i-pwn	Paiwan
i-tao	Tao
i-tay	Tayal
i-tsu	Tsou
no-bok	Norwegian "Book language"
no-nyn	Norwegian "New Norwegian"
zh-gan	Kan or Gan
zh-guoyu	Mandarin or Standard Chinese
zh-hakka	Hakka
zh-min	Min, Fuzhou, Hokkien, Amoy, or Taiwanese
zh-wuu	Shanghaiese or Wu
zh-xiang	Xiang or Hunanese
zh-yue	Cantonese

* The country codes AA, QM–QZ, XA–XZ and ZZ are reserved by ISO 3166 as user-assigned codes. These must not be used to form language tags.

ISO 639 Language Codes

Table G-2. ISO 639 and 639-2 language codes

Language	ISO 639	ISO 639-2
Abkhazian	ab	abk
Achinese		ace
Acoli		ach
Adangme		ada
Afar	aa	aar
Afrihili		afh
Afrikaans	af	afr
Afro-Asiatic (Other)		afa
Akan		aka
Akkadian		akk
Albanian	sq	alb/sqi
Aleut		ale
Algonquian languages		alg
Altaic (Other)		tut
Amharic	am	amh
Apache languages		apa
Arabic	ar	ara
Aramaic		arc
Arapaho		arp
Araucanian		arn
Arawak		arw
Armenian	hy	arm/hye
Artificial (Other)		art
Assamese	as	asm
Athapascan languages		ath
Austronesian (Other)		map
Avaric		ava
Avestan		ave
Awadhi		awa
Aymara	ay	aym
Azerbaijani	az	aze
Aztec		nah
Balinese		ban
Baltic (Other)		bat

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Baluchi		bal
Bambara		bam
Bamileke languages		bai
Banda		bad
Bantu (Other)		bnt
Basa		bas
Bashkir	ba	bak
Basque	eu	baq/eus
Beja		bej
Bemba		bem
Bengali	bn	ben
Berber (Other)		ber
Bhojpuri		bho
Bihari	bh	bih
Bikol		bik
Bini		bin
Bislama	bi	bis
Braj		bra
Breton	be	bre
Buginese		bug
Bulgarian	bg	bul
Buriat		bua
Burmese	my	bur/mya
Byelorussian	be	bel
Caddo		cad
Carib		car
Catalan	ca	cat
Caucasian (Other)		cau
Cebuano		ceb
Celtic (Other)		cel
Central American Indian (Other)		cai
Chagatai		chg
Chamorro		cha
Chechen		che
Cherokee		chr
Cheyenne		chy

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Chibcha		chb
Chinese	zh	chi/zho
Chinook jargon		chn
Choctaw		cho
Church Slavic		chu
Chuvash		chv
Coptic		cop
Cornish		cor
Corsican	co	cos
Cree		cre
Creek		mus
Creoles and Pidgins (Other)		crp
Creoles and Pidgins, English-based (Other)		cpe
Creoles and Pidgins, French-based (Other)		cpf
Creoles and Pidgins, Portuguese-based (Other)		cpp
Cushitic (Other)		cus
Croatian	hr	
Czech	cs	ces/cze
Dakota		dak
Danish	da	dan
Delaware		del
Dinka		din
Divehi		div
Dogri		doi
Dravidian (Other)		dra
Duala		dua
Dutch	nl	dut/nla
Dutch, Middle (ca. 1050-1350)		dum
Dyula		dyu
Dzongkha	dz	dzo
Efik		efi
Egyptian (Ancient)		egy
Ekajuk		eka
Elamite		elx
English	en	eng
English, Middle (ca. 1100-1500)		enm

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
English, Old (ca. 450-1100)		ang
Eskimo (Other)		esk
Esperanto	eo	epo
Estonian	et	est
Ewe		ewe
Ewondo		ewo
Fang		fan
Fanti		fat
Faroese	fo	fao
Fijian	fj	fij
Finnish	fi	fin
Finno-Ugrian (Other)		fiu
Fon		fon
French	fr	fra/fre
French, Middle (ca. 1400-1600)		frm
French, Old (842- ca. 1400)		fro
Frisian	fy	fry
Fulah		ful
Ga		gaa
Gaelic (Scots)		gae/gdh
Gallegan	gl	glg
Ganda		lug
Gayo		gay
Geez		gez
Georgian	ka	geo/kat
German	de	deu/ger
German, Middle High (ca. 1050-1500)		gmh
German, Old High (ca. 750-1050)		goh
Germanic (Other)		gem
Gilbertese		gil
Gondi		gon
Gothic		got
Grebo		grb
Greek, Ancient (to 1453)		grc
Greek, Modern (1453-)	el	ell/gre
Greenlandic	kl	kal

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Guarani	gn	grn
Gujarati	gu	guj
Haida		hai
Hausa	ha	hau
Hawaiian		haw
Hebrew	he	heb
Herero		her
Hiligaynon		hil
Himachali		him
Hindi	hi	hin
Hiri Motu		hmo
Hungarian	hu	hun
Hupa		hup
Iban		iba
Icelandic	is	ice/isl
Igbo		ibo
Ijo		ijo
Iloko		ilo
Indic (Other)		inc
Indo-European (Other)		ine
Indonesian	id	ind
Interlingua (IALA)	ia	ina
Interlingue	ie	ine
Inuktitut	iu	iku
Inupiak	ik	ipk
Iranian (Other)		ira
Irish	ga	gai/iri
Irish, Old (to 900)		sga
Irish, Middle (900 - 1200)		mga
Iroquoian languages		iro
Italian	it	ita
Japanese	ja	jpn
Javanese	jv/jw	jav/jaw
Judeo-Arabic		jrb
Judeo-Persian		jpr
Kabyle		kab

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Kachin		kac
Kamba		kam
Kannada	kn	kan
Kanuri		kau
Kara-Kalpak		kaa
Karen		kar
Kashmiri	ks	kas
Kawi		kaw
Kazakh	kk	kaz
Khasi		kha
Khmer	km	khm
Khoisan (Other)		khi
Khotanese		kho
Kikuyu		kik
Kinyarwanda	rw	kin
Kirghiz	ky	kir
Komi		kom
Kongo		kon
Konkani		kok
Korean	ko	kor
Kpelle		kpe
Kru		kro
Kuanyama		kua
Kumyk		kum
Kurdish	ku	kur
Kurukh		kru
Kusaie		kus
Kutenai		kut
Ladino		lad
Lahnda		lah
Lamba		lam
Langue d'Oc (post-1500)	oc	oci
Lao	lo	lao
Latin	la	lat
Latvian	lv	lav
Letzeburgesch		ltz

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Lezghian		lez
Lingala	ln	lin
Lithuanian	lt	lit
Lozi		loz
Luba-Katanga		lub
Luiseno		lui
Lunda		lun
Luo (Kenya and Tanzania)		luo
Macedonian	mk	mac/mak
Madurese		mad
Magahi		mag
Maithili		mai
Makasar		mak
Malagasy	mg	mlg
Malay	ms	may/msa
Malayalam		mal
Maltese	ml	mlt
Mandingo		man
Manipuri		mni
Manobo languages		mno
Manx		max
Maori	mi	mao/mri
Marathi	mr	mar
Mari		chm
Marshall		mah
Marwari		mwr
Masai		mas
Mayan languages		myn
Mende		men
Micmac		mic
Minangkabau		min
Miscellaneous (Other)		mis
Mohawk		moh
Moldavian	mo	mol
Mon-Kmer (Other)		mkh
Mongo		lol

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Mongolian	mn	mon
Mossi		mos
Multiple languages		mul
Munda languages		mun
Nauru	na	nau
Navajo		nav
Ndebele, North		nde
Ndebele, South		nbl
Ndongo		ndo
Nepali	ne	nep
Newari		new
Niger-Kordofanian (Other)		nic
Nilo-Saharan (Other)		ssa
Niuean		niu
Norse, Old		non
North American Indian (Other)		nai
Norwegian	no	nor
Norwegian (Nynorsk)		nno
Nubian languages		nub
Nyamwezi		nym
Nyanja		nya
Nyankole		nyn
Nyoro		nyo
Nzima		nzi
Ojibwa		oji
Oriya	or	ori
Oromo	om	orm
Osage		osa
Ossetic		oss
Otomian languages		oto
Pahlavi		pal
Palauan		pau
Pali		pli
Pampanga		pam
Pangasinan		pag
Panjabi	pa	pan

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Papiamento		pap
Papuan-Australian (Other)		paa
Persian	fa	fas/per
Persian, Old (ca 600 - 400 B.C.)		peo
Phoenician		phn
Polish	pl	pol
Ponape		pon
Portuguese	pt	por
Prakrit languages		pra
Provencal, Old (to 1500)		pro
Pushto	ps	pus
Quechua	qu	que
Rhaeto-Romance	rm	roh
Rajasthani		raj
Rarotongan		rar
Romance (Other)		roa
Romanian	ro	ron/rum
Romany		rom
Rundi	rn	run
Russian	ru	rus
Salishan languages		sal
Samaritan Aramaic		sam
Sami languages		smi
Samoan	sm	smo
Sandawe		sad
Sango	sg	sag
Sanskrit	sa	san
Sardinian		srd
Scots		sco
Selkup		sel
Semitic (Other)		sem
Serbian	sr	
Serbo-Croatian	sh	scr
Serer		srr
Shan		shn
Shona	sn	sna

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Sidamo		sid
Siksika		bla
Sindhi	sd	snd
Singhalese	si	sin
Sino-Tibetan (Other)		sit
Siouan languages		sio
Slavic (Other)		sla
Siswant	ss	ssw
Slovak	sk	slk/slo
Slovenian	sl	slv
Sogdian		sog
Somali	so	som
Songhai		son
Sorbian languages		wen
Sotho, Northern		nso
Sotho, Southern	st	sot
South American Indian (Other)		sai
Spanish	es	esl/spa
Sukuma		suk
Sumerian		sux
Sudanese	su	sun
Susu		sus
Swahili	sw	swa
Swazi		ssw
Swedish	sv	sve/swe
Syriac		syr
Tagalog	tl	tgl
Tahitian		tah
Tajik	tg	tgk
Tamashek		tmh
Tamil	ta	tam
Tatar	tt	tat
Telugu	te	tel
Tereno		ter
Thai	th	tha
Tibetan	bo	bod/tib

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Tigre		tig
Tigrinya	ti	tir
Timne		tem
Tivi		tiv
Tlingit		tli
Tonga (Nyasa)	to	tog
Tonga (Tonga Islands)		ton
Truk		tru
Tsimshian		tsi
Tsonga	ts	tso
Tswana	tn	tsn
Tumbuka		tum
Turkish	tr	tur
Turkish, Ottoman (1500–1928)		ota
Turkmen	tk	tuk
Tuvinian		tyv
Twi	tw	twi
Ugaritic		uga
Uighur	ug	uig
Ukrainian	uk	ukr
Umbundu		umb
Undetermined		und
Urdu	ur	urd
Uzbek	uz	uzb
Vai		vai
Venda		ven
Vietnamese	vi	vie
Volapük	vo	vol
Votic		vot
Wakashan languages		wak
Walamo		wal
Waray		war
Washo		was
Welsh	cy	cym/wel
Wolof	wo	wol
Xhosa	xh	xho

Table G-2. ISO 639 and 639-2 language codes (continued)

Language	ISO 639	ISO 639-2
Yakut		sah
Yao		yao
Yap		yap
Yiddish	yi	yid
Yoruba	yo	yor
Zapotec		zap
Zenaga		zen
Zhuang	za	zha
Zulu	zu	zul
Zuni		zun

ISO 3166 Country Codes

Table G-3. ISO 3166 country codes

Country	Code
Afghanistan	AF
Albania	AL
Algeria	DZ
American Samoa	AS
Andorra	AD
Angola	AO
Anguilla	AI
Antarctica	AQ
Antigua and Barbuda	AG
Argentina	AR
Armenia	AM
Aruba	AW
Australia	AU
Austria	AT
Azerbaijan	AZ
Bahamas	BS
Bahrain	BH
Bangladesh	BD
Barbados	BB
Belarus	BY
Belgium	BE

Table G-3. ISO 3166 country codes (continued)

Country	Code
Belize	BZ
Benin	BJ
Bermuda	BM
Bhutan	BT
Bolivia	BO
Bosnia and Herzegovina	BA
Botswana	BW
Bouvet Island	BV
Brazil	BR
British Indian Ocean Territory	IO
Brunei Darussalam	BN
Bulgaria	BG
Burkina Faso	BF
Burundi	BI
Cambodia	KH
Cameroon	CM
Canada	CA
Cape Verde	CV
Cayman Islands	KY
Central African Republic	CF
Chad	TD
Chile	CL
China	CN
Christmas Island	CX
Cocos (Keeling) Islands	CC
Colombia	CO
Comoros	KM
Congo	CG
Congo (Democratic Republic of the)	CD
Cook Islands	CK
Costa Rica	CR
Cote D'Ivoire	CI
Croatia	HR
Cuba	CU
Cyprus	CY
Czech Republic	CZ

Table G-3. ISO 3166 country codes (continued)

Country	Code
Denmark	DK
Djibouti	DJ
Dominica	DM
Dominican Republic	DO
East Timor	TP
Ecuador	EC
Egypt	EG
El Salvador	SV
Equatorial Guinea	GQ
Eritrea	ER
Estonia	EE
Ethiopia	ET
Falkland Islands (Malvinas)	FK
Faroe Islands	FO
Fiji	FJ
Finland	FI
France	FR
French Guiana	GF
French Polynesia	PF
French Southern Territories	TF
Gabon	GA
Gambia	GM
Georgia	GE
Germany	DE
Ghana	GH
Gibraltar	GI
Greece	GR
Greenland	GL
Grenada	GD
Guadeloupe	GP
Guam	GU
Guatemala	GT
Guinea	GN
Guinea-Bissau	GW
Guyana	GY
Haiti	HT

Table G-3. ISO 3166 country codes (continued)

Country	Code
Heard Island and Mcdonald Islands	HM
Holy See (Vatican City State)	VA
Honduras	HN
Hong Kong	HK
Hungary	HU
Iceland	IS
India	IN
Indonesia	ID
Iran (Islamic Republic of)	IR
Iraq	IQ
Ireland	IE
Israel	IL
Italy	IT
Jamaica	JM
Japan	JP
Jordan	JO
Kazakstan	KZ
Kenya	KE
Kiribati	KI
Korea (Democratic People's Republic of)	KP
Korea (Republic of)	KR
Kuwait	KW
Kyrgyzstan	KG
Lao People's Democratic Republic	LA
Latvia	LV
Lebanon	LB
Lesotho	LS
Liberia	LR
Libyan Arab Jamahiriya	LY
Liechtenstein	LI
Lithuania	LT
Luxembourg	LU
Macau	MO
Macedonia (The Former Yugoslav Republic of)	MK
Madagascar	MG
Malawi	MW

Table G-3. ISO 3166 country codes (continued)

Country	Code
Malaysia	MY
Maldives	MV
Mali	ML
Malta	MT
Marshall Islands	MH
Martinique	MQ
Mauritania	MR
Mauritius	MU
Mayotte	YT
Mexico	MX
Micronesia (Federated States of)	FM
Moldova (Republic of)	MD
Monaco	MC
Mongolia	MN
Montserrat	MS
Morocco	MA
Mozambique	MZ
Myanmar	MM
Namibia	NA
Nauru	NR
Nepal	NP
Netherlands	NL
Netherlands Antilles	AN
New Caledonia	NC
New Zealand	NZ
Nicaragua	NI
Niger	NE
Nigeria	NG
Niue	NU
Norfolk Island	NF
Northern Mariana Islands	MP
Norway	NO
Oman	OM
Pakistan	PK
Palau	PW
Palestinian Territory (Occupied)	PS
Panama	PA

Table G-3. ISO 3166 country codes (continued)

Country	Code
Papua New Guinea	PG
Paraguay	PY
Peru	PE
Philippines	PH
Pitcairn	PN
Poland	PL
Portugal	PT
Puerto Rico	PR
Qatar	QA
Reunion	RE
Romania	RO
Russian Federation	RU
Rwanda	RW
Saint Helena	SH
Saint Kitts and Nevis	KN
Saint Lucia	LC
Saint Pierre and Miquelon	PM
Saint Vincent and the Grenadines	VC
Samoa	WS
San Marino	SM
Sao Tome and Principe	ST
Saudi Arabia	SA
Senegal	SN
Seychelles	SC
Sierra Leone	SL
Singapore	SG
Slovakia	SK
Slovenia	SI
Solomon Islands	SB
Somalia	SO
South Africa	ZA
South Georgia and the South Sandwich Islands	GS
Spain	ES
Sri Lanka	LK
Sudan	SD
Suriname	SR
Svalbard and Jan Mayen	SJ

Table G-3. ISO 3166 country codes (continued)

Country	Code
Swaziland	SZ
Sweden	SE
Switzerland	CH
Syrian Arab Republic	SY
Taiwan, Province of China	TW
Tajikistan	TJ
Tanzania (United Republic of)	TZ
Thailand	TH
Togo	TG
Tokelau	TK
Tonga	TO
Trinidad and Tobago	TT
Tunisia	TN
Turkey	TR
Turkmenistan	TM
Turks and Caicos Islands	TC
Tuvalu	TV
Uganda	UG
Ukraine	UA
United Arab Emirates	AE
United Kingdom	GB
United States	US
United States Minor Outlying Islands	UM
Uruguay	UY
Uzbekistan	UZ
Vanuatu	VU
Venezuela	VE
Viet NAM	VN
Virgin Islands (British)	VG
Virgin ISLANDS (U.S.)	VI
Wallis and Futuna	WF
Western Sahara	EH
Yemen	YE
Yugoslavia	YU
Zambia	ZM

Language Administrative Organizations

ISO 639 defines a maintenance agency for additions to and changes in the list of languages in ISO 639. This agency is:

International Information Centre for Terminology (Infoterm)
P.O. Box 130
A-1021 Wien
Austria

Phone: +43 1 26 75 35 Ext. 312
Fax: +43 1 216 32 72

ISO 639-2 defines a maintenance agency for additions to and changes in the list of languages in ISO 639-2. This agency is:

Library of Congress
Network Development and MARC Standards Office
Washington, D.C. 20540
USA

Phone: +1 202 707 6237
Fax: +1 202 707 0115
URL: *http://www.loc.gov/standards/iso639/*

The maintenance agency for ISO 3166 (country codes) is:

ISO 3166 Maintenance Agency Secretariat
c/o DIN Deutsches Institut fuer Normung
Burggrafenstrasse 6
Postfach 1107
D-10787 Berlin
Germany

Phone: +49 30 26 01 320
Fax: +49 30 26 01 231
URL: *http://www.din.de/gremien/nas/nabd/iso3166ma/*

APPENDIX H

MIME Charset Registry

This appendix describes the MIME charset registry maintained by the Internet Assigned Numbers Authority (IANA). A formatted table of charsets from the registry is provided in Table H-1.

MIME Charset Registry

MIME charset tags are registered with the IANA (*http://www.iana.org/numbers.htm*). The charset registry is a flat-file text database of records. Each record contains a charset name, reference citations, a unique MIB number, a source description, and a list of aliases. A name or alias may be flagged "preferred MIME name."

Here is the record for US-ASCII:

```
Name: ANSI_X3.4-1968                                    [RFC1345, KXS2]^
MIBenum: 3
Source: ECMA registry
Alias: iso-ir-6
Alias: ANSI_X3.4-1986
Alias: ISO_646.irv:1991
Alias: ASCII
Alias: ISO646-US
Alias: US-ASCII (preferred MIME name)
Alias: us
Alias: IBM367
Alias: cp367
Alias: csASCII
```

The procedure for registering a charset with the IANA is documented in RFC 2978 (*http://www.ietf.org/rfc/rfc2978.txt*).

Preferred MIME Names

Of the 235 charsets registered at the time of this writing, only 20 include "preferred MIME names"—common charsets used by email and web applications. These are:

Big5	EUC-JP	EUC-KR
GB2312	ISO-2022-JP	ISO-2022-JP-2
ISO-2022-KR	ISO-8859-1	ISO-8859-2
ISO-8859-3	ISO-8859-4	ISO-8859-5
ISO-8859-6	ISO-8859-7	ISO-8859-8
ISO-8859-9	ISO-8859-10	KOI8-R
Shift-JIS	US-ASCII	

Registered Charsets

Table H-1 lists the contents of the charset registry as of March 2001. Refer directly to *http://www.iana.org* for more information about the contents of this table.

Table H-1. IANA MIME charset tags

Charset tag	Aliases	Description	References
US-ASCII	ANSI_X3.4-1968, iso-ir-6, ANSI_X3.4-1986, ISO_646.irv:1991, ASCII, ISO646-US, us, IBM367, cp367, csASCII	ECMA registry	RFC1345, KXS2
ISO-10646-UTF-1	csISO10646UTF1	Universal Transfer Format (1)—this is the multibyte encoding that subsets ASCII-7; it does not have byte-ordering issues	
ISO_646.basic:1983	ref, csISO646basic1983	ECMA registry	RFC1345, KXS2
INVARIANT	csINVARIANT		RFC1345, KXS2
ISO_646.irv:1983	iso-ir-2, irv, csISO2IntlRefVersion	ECMA registry	RFC1345, KXS2
BS_4730	iso-ir-4, ISO646-GB, gb, uk, csISO4UnitedKingdom	ECMA registry	RFC1345, KXS2
NATS-SEFI	iso-ir-8-1, csNATSSEFI	ECMA registry	RFC1345, KXS2
NATS-SEFI-ADD	iso-ir-8-2, csNATSSEFIADD	ECMA registry	RFC1345, KXS2
NATS-DANO	iso-ir-9-1, csNATSDANO	ECMA registry	RFC1345, KXS2
NATS-DANO-ADD	iso-ir-9-2, csNATSDANOADD	ECMA registry	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
SEN_850200_B	iso-ir-10, FI, ISO646-FI, ISO646-SE, se, csISO10Swedish	ECMA registry	RFC1345, KXS2
SEN_850200_C	iso-ir-11, ISO646-SE2, se2, csISO11SwedishForNames	ECMA registry	RFC1345, KXS2
KS_C_5601-1987	iso-ir-149, KS_C_5601-1989, KSC_5601, korean, csKSC56011987	ECMA registry	RFC1345, KXS2
ISO-2022-KR	csISO2022KR	RFC 1557 (see also KS_C_5601-1987)	RFC1557, Choi
EUC-KR	csEUCKR	RFC 1557 (see also KS_C_5861-1992)	RFC1557, Choi
ISO-2022-JP	csISO2022JP	RFC 1468 (see also RFC 2237)	RFC1468, Murai
ISO-2022-JP-2	csISO2022JP2	RFC 1554	RFC1554, Ohta
ISO-2022-CN		RFC 1922	RFC1922
ISO-2022-CN-EXT		RFC 1922	RFC1922
JIS_C6220-1969-jp	JIS_C6220-1969, iso-ir-13, katakana, x0201-7, csISO13JISC6220jp	ECMA registry	RFC1345, KXS2
JIS_C6220-1969-ro	iso-ir-14, jp, ISO646-JP, csISO14JISC6220ro	ECMA registry	RFC1345, KXS2
IT	iso-ir-15, ISO646-IT, csISO15Italian	ECMA registry	RFC1345, KXS2
PT	iso-ir-16, ISO646-PT, csISO16Portuguese	ECMA registry	RFC1345, KXS2
ES	iso-ir-17, ISO646-ES, csISO17Spanish	ECMA registry	RFC1345, KXS2
greek7-old	iso-ir-18, csISO18Greek7Old	ECMA registry	RFC1345, KXS2
latin-greek	iso-ir-19, csISO19LatinGreek	ECMA registry	RFC1345, KXS2
DIN_66003	iso-ir-21, de, ISO646-DE, csISO21German	ECMA registry	RFC1345, KXS2
NF_Z_62-010_(1973)	iso-ir-25, ISO646-FR1, csISO25French	ECMA registry	RFC1345, KXS2
Latin-greek-1	iso-ir-27, csISO27LatinGreek1	ECMA registry	RFC1345, KXS2
ISO_5427	iso-ir-37, csISO5427Cyrillic	ECMA registry	RFC1345, KXS2
JIS_C6226-1978	iso-ir-42, csISO42JISC62261978	ECMA registry	RFC1345, KXS2
BS_viewdata	iso-ir-47, csISO47BSViewdata	ECMA registry	RFC1345, KXS2
INIS	iso-ir-49, csISO49INIS	ECMA registry	RFC1345, KXS2
INIS-8	iso-ir-50, csISO50INIS8	ECMA registry	RFC1345, KXS2
INIS-cyrillic	iso-ir-51, csISO51INISCyrillic	ECMA registry	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
ISO_5427:1981	iso-ir-54, ISO5427Cyrillic1981	ECMA registry	RFC1345, KXS2
ISO_5428:1980	iso-ir-55, csISO5428Greek	ECMA registry	RFC1345, KXS2
GB_1988-80	iso-ir-57, cn, ISO646-CN, csISO57GB1988	ECMA registry	RFC1345, K5, KXS2
GB_2312-80	iso-ir-58, chinese, csISO58GB231280	ECMA registry	RFC1345, KXS2
NS_4551-1	iso-ir-60, ISO646-NO, no, csISO60DanishNorwegian, csISO60Norwegian1	ECMA registry	RFC1345, KXS2
NS_4551-2	ISO646-NO2, iso-ir-61, no2, csISO61Norwegian2	ECMA registry	RFC1345, KXS2
NF_Z_62-010	iso-ir-69, ISO646-FR, fr, csISO69French	ECMA registry	RFC1345, KXS2
videotex-suppl	iso-ir-70, csISO70VideotexSupp1	ECMA registry	RFC1345, KXS2
PT2	iso-ir-84, ISO646-PT2, csISO84Portuguese2	ECMA registry	RFC1345, KXS2
ES2	iso-ir-85, ISO646-ES2, csISO85Spanish2	ECMA registry	RFC1345, KXS2
MSZ_7795.3	iso-ir-86, ISO646-HU, hu, csISO86Hungarian	ECMA registry	RFC1345, KXS2
JIS_C6226-1983	iso-ir-87, x0208, JIS_X0208-1983, csISO87JISX0208	ECMA registry	RFC1345, KXS2
greek7	iso-ir-88, csISO88Greek7	ECMA registry	RFC1345, KXS2
ASMO_449	ISO_9036, arabic7, iso-ir-89, csISO89ASMO449	ECMA registry	RFC1345, KXS2
iso-ir-90	csISO90	ECMA registry	RFC1345, KXS2
JIS_C6229-1984-a	iso-ir-91, jp-ocr-a, csISO91JISC62291984a	ECMA registry	RFC1345, KXS2
JIS_C6229-1984-b	iso-ir-92, ISO646-JP-OCR-B, jp-ocr-b, csISO92JISC62991984b	ECMA registry	RFC1345, KXS2
JIS_C6229-1984-b-add	iso-ir-93, jp-ocr-b-add, csISO93JIS62291984badd	ECMA registry	RFC1345, KXS2
JIS_C6229-1984-hand	iso-ir-94, jp-ocr-hand, csISO94JIS62291984hand	ECMA registry	RFC1345, KXS2
JIS_C6229-1984-hand-add	iso-ir-95, jp-ocr-hand-add, csISO95JIS62291984handadd	ECMA registry	RFC1345, KXS2
JIS_C6229-1984-kana	iso-ir-96, csISO96JISC62291984kana	ECMA registry	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
ISO_2033-1983	iso-ir-98, e13b, csISO2033	ECMA registry	RFC1345, KXS2
ANSI_X3.110-1983	iso-ir-99, CSA_T500-1983, NAPLPS, csISO99NAPLPS	ECMA registry	RFC1345, KXS2
ISO-8859-1	ISO_8859-1:1987, iso-ir-100, ISO_8859-1, latin1, l1, IBM819, CP819, csISOLatin1	ECMA registry	RFC1345, KXS2
ISO-8859-2	ISO_8859-2:1987, iso-ir-101, ISO_8859-2, latin2, l2, csISOLatin2	ECMA registry	RFC1345, KXS2
T.61-7bit	iso-ir-102, csISO102T617bit	ECMA registry	RFC1345, KXS2
T.61-8bit	T.61, iso-ir-103, csISO103T618bit	ECMA registry	RFC1345, KXS2
ISO-8859-3	ISO_8859-3:1988, iso-ir-109, ISO_8859-3, latin3, l3, csISOLatin3	ECMA registry	RFC1345, KXS2
ISO-8859-4	ISO_8859-4:1988, iso-ir-110, ISO_8859-4, latin4, l4, csISOLatin4	ECMA registry	RFC1345, KXS2
ECMA-cyrillic	iso-ir-111, csISO111ECMACyrillic	ECMA registry	RFC1345, KXS2
CSA_Z243.4-1985-1	iso-ir-121, ISO646-CA, csa7-1, ca, csISO121Canadian1	ECMA registry	RFC1345, KXS2
CSA_Z243.4-1985-2	iso-ir-122, ISO646-CA2, csa7-2, csISO122Canadian2	ECMA registry	RFC1345, KXS2
CSA_Z243.4-1985-gr	iso-ir-123, csISO123CSAZ24341985gr	ECMA registry	RFC1345, KXS2
ISO-8859-6	ISO_8859-6:1987, iso-ir-127, ISO_8859-6, ECMA-114, ASMO-708, arabic, csISOLatinArabic	ECMA registry	RFC1345, KXS2
ISO_8859-6-E	csISO88596E	RFC 1556	RFC1556, IANA
ISO_8859-6-I	csISO88596I	RFC 1556	RFC1556, IANA
ISO-8859-7	ISO_8859-7:1987, iso-ir-126, ISO_8859-7, ELOT_928, ECMA-118, greek, greek8, csISOLatinGreek	ECMA registry	RFC1947, RFC1345, KXS2
T.101-G2	iso-ir-128, csISO128T101G2	ECMA registry	RFC1345, KXS2
ISO-8859-8	ISO_8859-8:1988, iso-ir-138, ISO_8859-8, hebrew, csISOLatinHebrew	ECMA registry	RFC1345, KXS2
ISO_8859-8-E	csISO88598E	RFC 1556	RFC1556, Nussbacher

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
ISO_8859-8-I	csISO88598I	RFC 1556	RFC1556, Nussbacher
CSN_369103	iso-ir-139, csISO139CSN369103	ECMA registry	RFC1345, KXS2
JUS_I.B1.002	iso-ir-141, ISO646-YU, js, yu, csISO141JUSIB1002	ECMA registry	RFC1345, KXS2
ISO_6937-2-add	iso-ir-142, csISOTextComm	ECMA registry and ISO 6937-2:1983	RFC1345, KXS2
IEC_P27-1	iso-ir-143, csISO143IECP271	ECMA registry	RFC1345, KXS2
ISO-8859-5	ISO_8859-5:1988, iso-ir-144, ISO_8859-5, cyrillic, csISOLatinCyrillic	ECMA registry	RFC1345, KXS2
JUS_I.B1.003-serb	iso-ir-146, serbian, csISO146Serbian	ECMA registry	RFC1345, KXS2
JUS_I.B1.003-mac	macedonian, iso-ir-147, csISO147Macedonian	ECMA registry	RFC1345, KXS2
ISO-8859-9	ISO_8859-9:1989, iso-ir-148, ISO_8859-9, latin5, l5, csISOLatin5	ECMA registry	RFC1345, KXS2
greek-ccitt	iso-ir-150, csISO150, csISO150GreekCCITT	ECMA registry	RFC1345, KXS2
NC_NC00-10:81	cuba, iso-ir-151, ISO646-CU, csISO151Cuba	ECMA registry	RFC1345, KXS2
ISO_6937-2-25	iso-ir-152, csISO6937Add	ECMA registry	RFC1345, KXS2
GOST_19768-74	ST_SEV_358-88, iso-ir-153, csISO153GOST1976874	ECMA registry	RFC1345, KXS2
ISO_8859-supp	iso-ir-154, latin1-2-5, csISO8859Supp	ECMA registry	RFC1345, KXS2
ISO_10367-box	iso-ir-155, csISO10367Box	ECMA registry	RFC1345, KXS2
ISO-8859-10	iso-ir-157, l6, ISO_8859-10:1992, csISOLatin6, latin6	ECMA registry	RFC1345, KXS2
latin-lap	lap, iso-ir-158, csISO158Lap	ECMA registry	RFC1345, KXS2
JIS_X0212-1990	x0212, iso-ir-159, csISO159JISX02121990	ECMA registry	RFC1345, KXS2
DS_2089	DS2089, ISO646-DK, dk, csISO646Danish	Danish Standard, DS 2089, February 1974	RFC1345, KXS2
us-dk	csUSDK		RFC1345, KXS2
dk-us	csDKUS		RFC1345, KXS2
JIS_X0201	X0201, csHalfWidthKatakana	JIS X 0201-1976—1 byte only; this is equivalent to JIS/Roman (similar to ASCII) plus 8-bit half-width katakana	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
KSC5636	ISO646-KR, csKSC5636		RFC1345, KXS2
ISO-10646-UCS-2	csUnicode	The 2-octet Basic Multilingual Plane, a.k.a. Unicode—this needs to specify network byte order; the standard does not specify it (it is a 16-bit integer space)	
ISO-10646-UCS-4	csUCS4	The full code space (same comment about byte order; these are 31-bit numbers)	
DEC-MCS	dec, csDECMCS	VAX/VMS User's Manual, Order Number: AI-Y517A-TE, April 1986	RFC1345, KXS2
hp-roman8	roman8, r8, csHPRoman8	LaserJet IIP Printer User's Manual, HP part no 33471-90901, Hewlett-Packard, June 1989	HP-PCL5, RFC1345, KXS2
macintosh	mac, csMacintosh	The Unicode Standard v1.0, ISBN 0201567881, Oct 1991	RFC1345, KXS2
IBM037	cp037, ebcdic-cp-us, ebcdic-cp-ca, ebcdic-cp-wt, ebcdic-cp-nl, csIBM037	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM038	EBCDIC-INT, cp038, csIBM038	IBM 3174 Character Set Ref, GA27-3831-02, March 1990	RFC1345, KXS2
IBM273	CP273, csIBM273	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM274	EBCDIC-BE, CP274, csIBM274	IBM 3174 Character Set Ref, GA27-3831-02, March 1990	RFC1345, KXS2
IBM275	EBCDIC-BR, cp275, csIBM275	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM277	EBCDIC-CP-DK, EBCDIC-CP-NO, csIBM277	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM278	CP278, ebcdic-cp-fi, ebcdic-cp-se, csIBM278	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM280	CP280, ebcdic-cp-it, csIBM280	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM281	EBCDIC-JP-E, cp281, csIBM281	IBM 3174 Character Set Ref, GA27-3831-02, March 1990	RFC1345, KXS2
IBM284	CP284, ebcdic-cp-es, csIBM284	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM285	CP285, ebcdic-cp-gb, csIBM285	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM290	cp290, EBCDIC-JP-kana, csIBM290	IBM 3174 Character Set Ref, GA27-3831-02, March 1990	RFC1345, KXS2
IBM297	cp297, ebcdic-cp-fr, csIBM297	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
IBM420	cp420, ebcdic-cp-ar1, csIBM420	IBM NLS RM Vol2 SE09-8002-01, March 1990, IBM NLS RM p 11-11	RFC1345, KXS2
IBM423	cp423, ebcdic-cp-gr, csIBM423	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM424	cp424, ebcdic-cp-he, csIBM424	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM437	cp437, 437, csPC8CodePage437	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM500	CP500, ebcdic-cp-be, ebcdic-cp-ch, csIBM500	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM775	cp775, csPC775Baltic	HP PCL 5 Comparison Guide (P/N 5021-0329) pp B-13, 1996	HP-PCL5
IBM850	cp850, 850, csPC850Multilingual	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM851	cp851, 851, csIBM851	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM852	cp852, 852, csPCp852	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM855	cp855, 855, csIBM855	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM857	cp857, 857, csIBM857	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM860	cp860, 860, csIBM860	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM861	cp861, 861, cp-is, csIBM861	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM862	cp862, 862, csPC862LatinHebrew	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM863	cp863, 863, csIBM863	IBM keyboard layouts and code pages, PN 07G4586, June 1991	RFC1345, KXS2
IBM864	cp864, csIBM864	IBM keyboard layouts and code pages, PN 07G4586, June 1991	RFC1345, KXS2
IBM865	cp865, 865, csIBM865	IBM DOS 3.3 Ref (Abridged), 94X9575, Feb 1987	RFC1345, KXS2
IBM866	cp866, 866, csIBM866	IBM NLDG Vol2 SE09-8002-03, August 1994	Pond
IBM868	CP868, cp-ar, csIBM868	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM869	cp869, 869, cp-gr, csIBM869	IBM keyboard layouts and code pages, PN 07G4586, June 1991	RFC1345, KXS2
IBM870	CP870, ebcdic-cp-roece, ebcdic-cp-yu, csIBM870	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
IBM871	CP871, ebcdic-cp-is, csIBM871	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM880	cp880, EBCDIC-Cyrillic, csIBM880	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM891	cp891, csIBM891	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM903	cp903, csIBM903	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM904	cp904, 904, csIBBM904	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM905	CP905, ebcdic-cp-tr, csIBM905	IBM 3174 Character Set Ref, GA27-3831-02, March 1990	RFC1345, KXS2
IBM918	CP918, ebcdic-cp-ar2, csIBM918	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
IBM1026	CP1026, csIBM1026	IBM NLS RM Vol2 SE09-8002-01, March 1990	RFC1345, KXS2
EBCDIC-AT-DE	csIBMEBCDICATDE	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-AT-DE-A	csEBCDICATDEA	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-CA-FR	csEBCDICCAFR	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-DK-NO	csEBCDICDKNO	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-DK-NO-A	csEBCDICDKNOA	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-FI-SE	csEBCDICFISE	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-FI-SE-A	csEBCDICFISEA	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-FR	csEBCDICFR	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-IT	csEBCDICIT	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-PT		IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-ES	csEBCDICES	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-ES-A	csEBCDICESA	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
EBCDIC-ES-S	csEBCDICESS	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-UK	csEBCDICUK	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
EBCDIC-US	csEBCDICUS	IBM 3270 Char Set Ref Ch 10, GA27-2837-9, April 1987	RFC1345, KXS2
UNKNOWN-8BIT	csUnknown8BiT		RFC1428
MNEMONIC	csMnemonic	RFC 1345, also known as "mnemonic+ascii+38"	RFC1345, KXS2
MNEM	csMnem	RFC 1345, also known as "mnemonic+ascii+8200"	RFC1345, KXS2
VISCII	csVISCII	RFC 1456	RFC1456
VIQR	csVIQR	RFC 1456	RFC1456
KOI8-R	csKOI8R	RFC 1489, based on GOST-19768-74, ISO-6937/8, INIS-Cyrillic, ISO-5427	RFC1489
KOI8-U		RFC 2319	RFC2319
IBM00858	CCSID00858, CP00858, PC-Multilingual-850+euro	IBM (see .../assignments/character-set-info/IBM00858) [Mahdi]	
IBM00924	CCSID00924, CP00924, ebcdic-Latin9--euro	IBM (see .../assignments/character-set-info/IBM00924) [Mahdi]	
IBM01140	CCSID01140, CP01140, ebcdic-us-37+euro	IBM (see .../assignments/character-set-info/IBM01140) [Mahdi]	
IBM01141	CCSID01141, CP01141, ebcdic-de-273+euro	IBM (see .../assignments/character-set-info/IBM01141) [Mahdi]	
IBM01142	CCSID01142, CP01142, ebcdic-dk-277+euro, ebcdic-no-277+euro	IBM (see .../assignments/character-set-info/IBM01142) [Mahdi]	
IBM01143	CCSID01143, CP01143, ebcdic-fi-278+euro, ebcdic-se-278+euro	IBM (see .../assignments/character-set-info/IBM01143) [Mahdi]	
IBM01144	CCSID01144, CP01144, ebcdic-it-280+euro	IBM (see .../assignments/character-set-info/IBM01144) [Mahdi]	
IBM01145	CCSID01145, CP01145, ebcdic-es-284+euro	IBM (see .../assignments/character-set-info/IBM01145) [Mahdi]	
IBM01146	CCSID01146, CP01146, ebcdic-gb-285+euro	IBM (see .../assignments/character-set-info/IBM01146) [Mahdi]	
IBM01147	CCSID01147, CP01147, ebcdic-fr-297+euro	IBM (see .../assignments/character-set-info/IBM01147) [Mahdi]	
IBM01148	CCSID01148, CP01148, ebcdic-international-500+euro	IBM (see .../assignments/character-set-info/IBM01148) [Mahdi]	

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
IBM01149	CCSID01149, CP01149, ebcdic-is-871+euro	IBM (see .../assignments/character-set-info/IBM01149) [Mahdi]	
Big5-HKSCS	None	See (.../assignments/character-set-info/Big5-HKSCS) [Yick]	
UNICODE-1-1	csUnicode11	RFC 1641	RFC1641
SCSU	None	SCSU (see .../assignments/character-set-info/SCSU) [Scherer]	
UTF-7	None	RFC 2152	RFC2152
UTF-16BE	None	RFC 2781	RFC2781
UTF-16LE	None	RFC 2781	RFC2781
UTF-16	None	RFC 2781	RFC2781
UNICODE-1-1-UTF-7	csUnicode11UTF7	RFC 1642	RFC1642
UTF-8		RFC 2279	RFC2279
iso-8859-13		ISO (see ...assignments/character-set-info/iso-8859-13)[Tumasonis]	
iso-8859-14	iso-ir-199, ISO_8859-14:1998, ISO_8859-14, latin8, iso-celtic, l8	ISO (see ...assignments/character-set-info/iso-8859-14) [Simonsen]	
ISO-8859-15	ISO_8859-15	ISO	
JIS_Encoding	csJISEncoding	JIS X 0202-1991; uses ISO 2022 escape sequences to shift code sets, as documented in JIS X 0202-1991	
Shift_JIS	MS_Kanji, csShiftJIS	This charset is an extension of csHalfWidthKatakana—it adds graphic characters in JIS X 0208. The CCSs areJIS X0201:1997 and JIS X0208: 1997. The complete definition is shown in Appendix 1 of JISX0208:1997. This charset can be used for the top-level media type "text".	
EUC-JP	Extended_UNIX_Code_Packed_Format_for_Japanese, csEUCPkdFmtJapanese	Standardized by OSF, UNIX International, and UNIX Systems Laboratories Pacific. Uses ISO 2022 rules to select code set. code set 0: US-ASCII (a single 7-bit byte set); code set 1: JIS X0208-1990 (a double 8-bit byte set) restricted to A0–FF in both bytes; code set 2: half-width katakana (a single 7-bit byte set) requiring SS2 as the character prefix; code set 3: JIS X0212-1990 (a double 7-bit byte set) restricted to A0–FF in both bytes requiring SS3 as the character prefix.	

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
Extended_UNIX_Code_Fixed_Width_for_Japanese	csEUCFixWidJapanese	Used in Japan. Each character is 2 octets. code set 0: US-ASCII (a single 7-bit byte set), 1st byte = 00, 2nd byte = 20–7E; code set 1: JIS X0208-1990 (a double 7-bit byte set) restricted to A0–FF in both bytes; code set 2: half-width katakana (a single 7-bit byte set), 1st byte = 00, 2nd byte = A0–FF; code set 3: JIS X0212-1990 (a double 7-bit byte set) restricted to A0–FF in the first byte and 21–7E in the second byte.	
ISO-10646-UCS-Basic	csUnicodeASCII	ASCII subset of Unicode. Basic Latin = collection 1. See ISO 10646, Appendix A.	
ISO-10646-Unicode-Latin1	csUnicodeLatin1, ISO-10646	ISO Latin-1 subset of Unicode. Basic Latin and Latin-1. Supplement = collections 1 and 2. See ISO 10646, Appendix A, and RFC 1815.	
ISO-10646-J-1		ISO 10646 Japanese. See RFC 1815.	
ISO-Unicode-IBM-1261	csUnicodeIBM1261	IBM Latin-2, -3, -5, Extended Presentation Set, GCSGID: 1261	
ISO-Unicode-IBM-1268	csUnidoceIBM1268	IBM Latin-4 Extended Presentation Set, GCSGID: 1268	
ISO-Unicode-IBM-1276	csUnicodeIBM1276	IBM Cyrillic Greek Extended Presentation Set, GCSGID: 1276	
ISO-Unicode-IBM-1264	csUnicodeIBM1264	IBM Arabic Presentation Set, GCSGID: 1264	
ISO-Unicode-IBM-1265	csUnicodeIBM1265	IBM Hebrew Presentation Set, GCSGID: 1265	
ISO-8859-1-Windows-3.0-Latin-1	csWindows30Latin1	Extended ISO 8859-1 Latin-1 for Windows 3.0. PCL Symbol Set ID: 9U.	HP-PCL5
ISO-8859-1-Windows-3.1-Latin-1	csWindows31Latin1	Extended ISO 8859-1 Latin-1 for Windows 3.1. PCL Symbol Set ID: 19U.	HP-PCL5
ISO-8859-2-Windows-Latin-2	csWindows31Latin2	Extended ISO 8859-2. Latin-2 for Windows 3.1. PCL Symbol Set ID: 9E.	HP-PCL5
ISO-8859-9-Windows-Latin-5	csWindows31Latin5	Extended ISO 8859-9. Latin-5 for Windows 3.1. PCL Symbol Set ID: 5T.	HP-PCL5
Adobe-Standard-Encoding	csAdobeStandardEncoding	PostScript Language Reference Manual. PCL Symbol Set ID: 10J.	Adobe
Ventura-US	csVenturaUS	Ventura US-ASCII plus characters typically used in publishing, such as pilcrow, copyright, registered, trademark, section, dagger, and double dagger in the range A0 (hex) to FF (hex). PCL Symbol Set ID: 14J.	HP-PCL5

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
Ventura-International	csVenturaInternational	Ventura International. ASCII plus coded characters similar to Roman8. PCL Symbol Set ID: 13J.	HP-PCL5
PC8-Danish-Norwegian	csPC8DanishNorwegian	PC Danish Norwegian 8-bit PC set for Danish Norwegian. PCL Symbol Set ID: 11U.	HP-PCL5
PC8-Turkish	csPC8Turkish	PC Latin Turkish. PCL Symbol Set ID: 9T.	HP-PCL5
IBM-Symbols	csIBMSymbols	Presentation Set, CPGID: 259	IBM-CIDT
IBM-Thai	csIBMThai	Presentation Set, CPGID: 838	IBM-CIDT
HP-Legal	csHPLegal	PCL 5 Comparison Guide, Hewlett-Packard, HP part number 5961-0510, October 1992. PCL Symbol Set ID: 1U.	HP-PCL5
HP-Pi-font	csHPPiFont	PCL 5 Comparison Guide, Hewlett-Packard, HP part number 5961-0510, October 1992. PCL Symbol Set ID: 15U.	HP-PCL5
HP-Math8	csHPMath8	PCL 5 Comparison Guide, Hewlett-Packard, HP part number 5961-0510, October 1992. PCL Symbol Set ID: 8M.	HP-PCL5
Adobe-Symbol-Encoding	csHPPSMath	PostScript Language Reference Manual. PCL Symbol Set ID: 5M.	Adobe
HP-DeskTop	csHPDesktop	PCL 5 Comparison Guide, Hewlett-Packard, HP part number 5961-0510, October 1992. PCL Symbol Set ID: 7J.	HP-PCL5
Ventura-Math	csVenturaMath	PCL 5 Comparison Guide, Hewlett-Packard, HP part number 5961-0510, October 1992. PCL Symbol Set ID: 6M.	HP-PCL5
Microsoft-Publishing	csMicrosoftPublishing	PCL 5 Comparison Guide, Hewlett-Packard, HP part number 5961-0510, October 1992. PCL Symbol Set ID: 6J.	HP-PCL5
Windows-31J	csWindows31J	Windows Japanese. A further extension of Shift_JIS to include NEC special characters (Row 13), NEC selection of IBM extensions (Rows 89 to 92), and IBM extensions (Rows 115 to 119). The CCSs are JIS X0201:1997, JIS X0208:1997, and these extensions. This charset can be used for the top-level media type "text", but it is of limited or specialized use (see RFC 2278). PCL Symbol Set ID: 19K.	
GB2312	csGB2312	Chinese for People's Republic of China (PRC) mixed 1-byte, 2-byte set: 20–7E = 1-byte ASCII; A1–FE = 2-byte PRC Kanji. See GB 2312-80. PCL Symbol Set ID: 18C.	

Table H-1. IANA MIME charset tags (continued)

Charset tag	Aliases	Description	References
Big5	csBig5	Chinese for Taiwan Multibyte set. PCL Symbol Set id: 18T.	
windows-1250		Microsoft (see .../character-set-info/windows-1250) [Lazhintseva]	
windows-1251		Microsoft (see .../character-set-info/windows-1251) [Lazhintseva]	
windows-1252		Microsoft (see .../character-set-info/windows-1252) [Wendt]	
windows-1253		Microsoft (see .../character-set-info/windows-1253) [Lazhintseva]	
windows-1254		Microsoft (see .../character-set-info/windows-1254) [Lazhintseva]	
windows-1255		Microsoft (see .../character-set-info/windows-1255) [Lazhintseva]	
windows-1256		Microsoft (see .../character-set-info/windows-1256) [Lazhintseva]	
windows-1257		Microsoft (see .../character-set-info/windows-1257) [Lazhintseva]	
windows-1258		Microsoft (see .../character-set-info/windows-1258) [Lazhintseva]	
TIS-620		Thai Industrial Standards Institute (TISI)	[Tantsetthi]
HZ-GB-2312		RFC 1842, RFC 1843 [RFC1842, RFC1843]	

Index

We'd like to hear your suggestions for improving our indexes. Send email to *index@oreilly.com*.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

About the Authors

David Gourley is the Chief Technology Officer of Endeca, where he leads the research and development of Endeca's products. Endeca develops Internet and intranet information-access solutions that provide new ways to navigate and explore enterprise data. Prior to working at Endeca, David was a member of the founding engineering team at Inktomi, where he helped develop Inktomi's Internet search database and was a key developer of Inktomi's web caching products.

David earned a B.A. in Computer Science from the University of California at Berkeley, and he holds several patents in web technologies.

Brian Totty was most recently the Vice President of R&D at Inktomi Corporation (a company he helped found in 1996), where he led research and development of web caching, streaming media, and Internet search technologies. Formerly, he was a scientist at Silicon Graphics, where he designed and optimized software for high-performance networking and supercomputing systems. Before that, he held an engineering position at Apple Computer's Advanced Technology Group.

Brian holds a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign and a B.S. degree in Computer Science and Electrical Engineering from MIT, where he received the Organick award for computer systems research. He also has developed and taught award-winning courses on Internet technology for the University of California Extension system.

Marjorie Sayer writes about network caching software at Inktomi Corporation. After earning M.A. and Ph.C. degrees in Mathematics at the University of California at Berkeley, she worked on mathematics curriculum reform. Since 1990 she has written about energy resource management, parallel systems software, telephony, and networking.

Sailu Reddy currently leads the development of embedded performance-enhancing HTTP proxies at Inktomi Corporation. Sailu has been developing complex software systems for 12 years and has been deeply involved in web infrastructure research and development since 1995. He was a core engineer of Netscape's first web server and web proxy products and of several following generations. His technical experience includes HTTP applications, data compression techniques, database engines, and collaboration management. Sailu earned an M.S. in Information Systems from the University of Arizona and holds several patents in web technologies.

Anshu Aggarwal is a Director of Engineering at Inktomi Corporation. He leads the protocol-processing engineering teams for Inktomi's web caching products, and he has been involved in the design of web technologies at Inktomi since 1997. Anshu holds M.S. and Ph.D. degrees in Computer Science from the University of Colorado at Boulder, specializing in memory-consistent techniques for distributed multiprocessor machines. He also holds M.S. and B.S. degrees in Electrical Engineering. Anshu is the author of several technical papers and holds two patents.

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *HTTP: The Definitive Guide* is a thirteen-lined ground squirrel (*Spermophilus tridecemlineatus*), common to central North America. True to its name, the thirteen-lined ground squirrel has thirteen stripes with rows of light spots that run the length of its back. Its color pattern blends into its surroundings, protecting it from predators. Thirteen-lined ground squirrels are members of the squirrel family, which includes chipmunks, ground squirrels, tree squirrels, prairie dogs, and woodchucks. They are similar in size to the eastern chipmunk but smaller than the common gray squirrel, averaging about 11 inches in length (including a 5–6 inch tail).

Thirteen-lined ground squirrels go into hibernation in October and emerge in late March or early April. Each female usually produces one litter of 7–10 young each May. The young leave the burrows at four to five weeks of age and are fully grown at six weeks. Ground squirrels prefer open areas with short grass and well-drained sandy or loamy soils for burrows, and they avoid wooded areas—mowed lawns, golf courses, and parks are common habitats.

Ground squirrels can cause problems when they create burrows, dig up newly planted seeds, and damage vegetable gardens. However, they are important prey to several predators, including badgers, coyotes, hawks, weasels, and various snakes, and they benefit humans directly by feeding on many harmful weeds, weed seeds, and insects.

Rachel Wheeler was the production editor and copyeditor for *HTTP: The Definitive Guide*. Leanne Soylemez, Sarah Sherman, and Mary Anne Weeks Mayo provided quality control, and Derek Di Matteo and Brian Sawyer provided production assistance. John Bickelhaupt wrote the index.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is an original illustration created by Lorrie LeJeune. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato and Melanie Wang designed the interior layout, based on a series design by David Futato. Joe Wizda prepared the files for production in FrameMaker 5.5.6. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. This colophon was written by Rachel Wheeler.

CPSIA information can be obtained at www.ICGtesting.com
Printed in the USA
BVOW09s0637270916

463309BV00006BA/9/P

Web Programming

O'REILLY®

HTTP: The Definitive Guide

Behind every successful web transaction lurks the Hypertext Transfer Protocol (HTTP), the language by which web clients and servers exchange documents and information. HTTP is commonly known as the workhorse behind the browsers we use every day to access our company intranets, locate out-of-print books, or research census information. But HTTP is used for far more than browsing the Web: the simplicity and ubiquity of HTTP also have made it the choice protocol for many other networked applications, most notably through web services such as SOAP and XML-RPC.

As the title suggests, *HTTP: The Definitive Guide* explains the HTTP protocol: how it works and how to use it to develop web-based applications. However, this book is not just about HTTP; it's also about all the other core Internet technologies that HTTP depends on to work effectively. Although HTTP is at the center of the book, the essence of *HTTP: The Definitive Guide* is in understanding how the Web works and how to apply that knowledge to web programming and administration. The book explains the technical workings, motivations, performance considerations, and objectives of HTTP and the technologies around which it revolves.

HTTP: The Definitive Guide is the bible for the HTTP protocol and related web technologies. Topics covered include:

- HTTP methods, headers, and status codes
- Optimizing proxies and caches
- Strategies for designing web robots and crawlers
- Cookies, authentication, and Secure HTTP
- Internationalization and content negotiation
- Redirection and load-balancing strategies

Written by experts with years of practical experience, this book uses clear, concise language and a plethora of detailed illustrations to help readers visualize what goes on behind the scenes, providing a complete understanding of the story behind each query on the Web.

All web programmers, administrators, and application developers need to be familiar with HTTP in order to work effectively. There are many books that explain how to use the Web, but this is the book that explains how the Web works.

www.oreilly.com

US $54.99 **CAN $63.99**

ISBN: 978-1-56592-509-0

9 781565 925090 55499