Uniform Resource Locators (URL)

A Uniform Resource Locator (URL) is a compact representation of the location and access method for a resource available via the Internet.

General form of a URL:
< scheme> :< scheme-specific-part>

URL Schemes:

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Default Port</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp</td>
<td>21</td>
<td>ftp://&lt;user&gt;:@&lt;host&gt;[:&lt;port&gt;]/&lt;cwd1&gt;[/&lt;cwd2&gt;]/.../&lt;cwdN&gt;/&lt;name&gt;; type=&lt;typecode&gt;</td>
</tr>
<tr>
<td>http</td>
<td>80</td>
<td>http://&lt;host&gt;:&lt;port&gt;/&lt;path&gt;?&lt;searchpart&gt;</td>
</tr>
<tr>
<td>gopher</td>
<td>70</td>
<td>gopher://&lt;host&gt;:&lt;port&gt;/&lt;selector&gt; or gopher://&lt;host&gt;:&lt;port&gt;/&lt;selector&gt;/%09&lt;search&gt; or gopher://&lt;host&gt;:&lt;port&gt;/&lt;selector&gt;/%09&lt;search&gt;/%09&lt;gopher+_.string&gt; or mailto:&lt;rfc822-addr-spec&gt;</td>
</tr>
<tr>
<td>mailto</td>
<td>—</td>
<td>mailto:&lt;rfc822-addr-spec&gt;</td>
</tr>
<tr>
<td>news</td>
<td>—</td>
<td>news:&lt;newsreader-name&gt; or news:&lt;message-id&gt;</td>
</tr>
<tr>
<td>nntp</td>
<td>119</td>
<td>nntp://&lt;host&gt;:&lt;port&gt;/&lt;newsreader-name&gt;/&lt;article-number&gt;</td>
</tr>
<tr>
<td>telnet</td>
<td>23</td>
<td>telnet://&lt;user&gt;:@&lt;host&gt;[:&lt;port&gt;]</td>
</tr>
<tr>
<td>wais</td>
<td>210</td>
<td>wais://&lt;host&gt;:&lt;port&gt;/&lt;database&gt; or wais://&lt;host&gt;:&lt;port&gt;/&lt;database&gt;/?&lt;search&gt; or wais://&lt;host&gt;:&lt;port&gt;/&lt;database&gt;/<em>&lt;type&gt;/</em>&lt;path&gt; or file://&lt;host&gt;/&lt;path&gt;</td>
</tr>
<tr>
<td>file</td>
<td>—</td>
<td>file://&lt;host&gt;/&lt;path&gt;</td>
</tr>
<tr>
<td>prospero</td>
<td>1525</td>
<td>prospero://&lt;host&gt;://&lt;port&gt;/<em>&lt;hostname&gt;/</em>&lt;field&gt;/*&lt;value&gt;</td>
</tr>
</tbody>
</table>

ftp://myhost.sci.brooklyn.cuny.edu/ziegler/myfile.txt
http://www.brooklyn.cuny.edu/index.html
mailto:webmaster@brooklyn.cuny.edu
Hypertext Transfer Protocol (HTTP)

HTTP is a protocol for transmitting information with the efficiency necessary for making hypertext jumps.

The data transferred can be plain text, hypertext, audio, images, or any Internet accessible information.

HTTP is a transaction-oriented, client-server protocol.

HTTP uses TCP. Nevertheless, HTTP is 'stateless'; i.e., each transaction is treated independently.

HTTP is the foundation protocol of the World Wide Web (WWW). The most typical use of HTTP is between a Web browser and a Web server.

HTTP can be used in any client-server application involving hypertext.

Universal Resource Identifier (URI):
A URI is a term for a generic WWW identifier. The URL is a type of URI, in which an access protocol is designated and a specific Internet address is provided. (The HTTP standard refers to URIs.)
Key Terms related to HTTP

Cache
A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server while it is acting as a tunnel.

Client
An application program that establishes connections for the purpose of sending requests.

Connection
A transport layer virtual circuit established between two application programs for the purposes of communication.

Entity
A particular representation or rendition of a data resource, or reply from a service resource, that may be enclosed within a request or response message. An entity consists of entity headers and an entity body.

Gateway
A server that acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the original server for the requested resource; the requesting client may not be aware that it is communicating with a gateway. Gateways are often used as server-side portals through network firewalls and as protocol translators for access to resources stored on non-HTTP systems.

Message
The basic unit of HTTP communication, consisting of a structured sequence of octets transmitted via the connection.

Origin Server
The server on which a given resource resides or is to be created.

Proxy
An intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them, with possible translation, on to other servers. A proxy must interpret and, if necessary, rewrite a request message before forwarding it. Proxies are often used as client-side portals through network firewalls and as helper applications for handling requests via protocols not implemented by the user agent.

Resource
A network data object or service that can be identified by a URI.

Server
An application program that accepts connections in order to service requests by sending back responses.

Tunnel
A tunnel is an intermediary program that is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. A tunnel ceases to exist when both ends of the relayed connections are closed. Tunnels are used when a portal is necessary and the intermediary cannot, or should not, interpret the relayed communication.

User Agent
The client that initiates a request. These are often browsers, editors, spiders, or other end-user tools.
Examples of HTTP Operation

Examples of HTTP operation.

a) Direct Connection
b) Connection through one or more Intermediate Systems (proxy, gateway, tunnel).

c) Example of a Cache.
Intermediate HTTP Systems

Proxy:
- Security Intermediary
  On client side of firewall
- Different versions of HTTP:

Gateway:
- Security Intermediary:
  On server side of firewall
- Non-HTTP Server

Tunnel:
A relay point between two TCP connections (e.g., a firewall through which a client or server external to a protected network can establish and maintain an authenticated connection for HTTP transactions.)
HTTP Messages

HTTP-Message = Simple-Request | Simple-Response | Full-Request | Full-Response

Full-Request = Request-Line
              *( General-Header | Request-Header | Entity-Header )
              CRLF
              [ Entity-Body ]

Full-Response = Status-Line
               *( General-Header | Response-Header | Entity-Header )
               CRLF
               [ Entity-Body ]

Simple-Request = "GET" SP Request-URI CRLF
Simple-Response = [ Entity-Body ]

General structure of HTTP messages.
HTTP Message Fields

! Request-Line:
   Identifies the message type and the requested response.

! Status Line:
   Provides status information about this response.

! General-Header:
   Contains fields that are applicable to both request and response messages, but which do not apply to the entity being transferred.

! Request-Header:
   Contains information about the request and the client.

! Entity-Header:
   Contains information about the response.

! Entity-Body:
   The body of the message.
Augmented BNF notation used in URL and HTTP specifications.

- Words in lower case represent variables or names of rules.
- A rule has the form
  
  name = definition

- DIGIT is any decimal digit; CRLF is carriage return, line feed; SP is one or more spaces.
- Quotation marks enclose literal text.
- Angle brackets, "<" ">", may be used within a definition to enclose a rule name when their presence will facilitate clarity.
- Elements separated by bar ("|") are alternatives.
- Ordinary parentheses are used simply for grouping.
- The character "*" preceding an element indicates repetition. The full form is
  
  <1>*<J>element

  indicating at least I and at most J occurrences of the element. *element allows any number, including 0; 1*element requires at least one element; and 1*2element allows 1 or 2 elements; <N>element means exactly N elements.
- Square brackets, "[" "]", enclose optional elements.
- The construct "#" is used to define, with the following form,
  
  <I>#+<J>element

  indicating at least I and at most J elements, each separated by a comma and optional linear white space.
- A semicolon at the right of a rule starts a comment that continues to the end of the line.
## HTTP Elements

HTTP elements.

### ALL MESSAGES

<table>
<thead>
<tr>
<th>GENERAL HEADER FIELDS</th>
<th>ENTITY HEADER FIELDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache-Control</td>
<td>Allow</td>
</tr>
<tr>
<td></td>
<td>Derived-From</td>
</tr>
<tr>
<td>Connection</td>
<td>Content-Encoding</td>
</tr>
<tr>
<td></td>
<td>Expires</td>
</tr>
<tr>
<td>Data</td>
<td>Content-Language</td>
</tr>
<tr>
<td></td>
<td>Last-Modified</td>
</tr>
<tr>
<td>Forwarded</td>
<td>Content-Length</td>
</tr>
<tr>
<td></td>
<td>Link</td>
</tr>
<tr>
<td></td>
<td>Content-MD5</td>
</tr>
<tr>
<td></td>
<td>Title</td>
</tr>
<tr>
<td></td>
<td>Content-Range</td>
</tr>
<tr>
<td></td>
<td>Transfer-Encoding</td>
</tr>
<tr>
<td></td>
<td>Content-Type</td>
</tr>
<tr>
<td></td>
<td>URI-Header</td>
</tr>
<tr>
<td></td>
<td>Content-Version</td>
</tr>
<tr>
<td></td>
<td>extention-header</td>
</tr>
</tbody>
</table>

### REQUEST MESSAGES

<table>
<thead>
<tr>
<th>REQUEST METHODS</th>
<th>REQUEST HEADER FIELDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>Accept</td>
</tr>
<tr>
<td>GET</td>
<td>If-Modified-Since</td>
</tr>
<tr>
<td>HEAD</td>
<td>Accept-Charset</td>
</tr>
<tr>
<td>POST</td>
<td>Proxy-Authentication</td>
</tr>
<tr>
<td>PUT</td>
<td>Accept-Encoding</td>
</tr>
<tr>
<td>PATCH</td>
<td>Range</td>
</tr>
<tr>
<td>COPY</td>
<td>Accept-Language</td>
</tr>
<tr>
<td></td>
<td>Referer</td>
</tr>
<tr>
<td></td>
<td>Authorization</td>
</tr>
<tr>
<td></td>
<td>Unless</td>
</tr>
<tr>
<td></td>
<td>From</td>
</tr>
<tr>
<td></td>
<td>User-Agent</td>
</tr>
</tbody>
</table>

### RESPONSE MESSAGES

<table>
<thead>
<tr>
<th>RESPONSE STATUS CODES</th>
<th>RESPONSE HEADER FIELDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue</td>
<td>Request Timeout</td>
</tr>
<tr>
<td>Switching Protocols</td>
<td>Location</td>
</tr>
<tr>
<td>OK</td>
<td>Proxy-Authenticate</td>
</tr>
<tr>
<td>Created</td>
<td>Public</td>
</tr>
<tr>
<td>Accepted</td>
<td>Retry-After</td>
</tr>
<tr>
<td>Non-Authoritative</td>
<td>Server</td>
</tr>
<tr>
<td>Information</td>
<td>WWW-Authenticate</td>
</tr>
<tr>
<td>No Content</td>
<td>Moved Temporarily</td>
</tr>
<tr>
<td>Reset Content</td>
<td>See Other</td>
</tr>
<tr>
<td>Partial Content</td>
<td>Conflict</td>
</tr>
<tr>
<td>Multiple Choices</td>
<td>Not Modified</td>
</tr>
<tr>
<td>Moved Permanently</td>
<td>Use Proxy</td>
</tr>
<tr>
<td></td>
<td>Bad Request</td>
</tr>
<tr>
<td></td>
<td>Unauthorized</td>
</tr>
<tr>
<td></td>
<td>Payment Required</td>
</tr>
<tr>
<td></td>
<td>Forbidden</td>
</tr>
<tr>
<td></td>
<td>Not Found</td>
</tr>
<tr>
<td></td>
<td>Method Not Allowed</td>
</tr>
<tr>
<td></td>
<td>None Acceptable</td>
</tr>
<tr>
<td></td>
<td>Proxy Authentication</td>
</tr>
<tr>
<td></td>
<td>Requested</td>
</tr>
</tbody>
</table>

Copyright © December 21, 2004 by Chaim Ziegler, Ph.D.
Some HTTP Request Methods

! Built-in HTTP Request Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request to read a Web page</td>
</tr>
<tr>
<td>HEAD</td>
<td>Request to read a Web page’s header</td>
</tr>
<tr>
<td>PUT</td>
<td>Request to store a Web page</td>
</tr>
<tr>
<td>POST</td>
<td>Append to a named resource (e.g., a Web page)</td>
</tr>
<tr>
<td>DELETE</td>
<td>Remove the Web page</td>
</tr>
<tr>
<td>TRACE</td>
<td>Echo the incoming request</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Query certain options</td>
</tr>
</tbody>
</table>

! Status Code Response Groups:

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xx</td>
<td>Information</td>
<td>100 = server agrees to handle client’s request</td>
</tr>
<tr>
<td>2xx</td>
<td>Success</td>
<td>200 = request succeeded; 204 = no content present</td>
</tr>
<tr>
<td>3xx</td>
<td>Redirection</td>
<td>301 = page moved; 304 = cached page still valid</td>
</tr>
<tr>
<td>4xx</td>
<td>Client error</td>
<td>403 = forbidden page; 404 = page not found</td>
</tr>
<tr>
<td>5xx</td>
<td>Server error</td>
<td>500 = internal server error; 503 = try again later</td>
</tr>
</tbody>
</table>
## Some HTTP Message Headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Type</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Agent</td>
<td>Request</td>
<td>Information about the browser and its platform</td>
</tr>
<tr>
<td>Accept</td>
<td>Request</td>
<td>The type of pages the client can handle</td>
</tr>
<tr>
<td>Accept-Charset</td>
<td>Request</td>
<td>The character sets that are acceptable to the client</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>Request</td>
<td>The page encodings the client can handle</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>Request</td>
<td>The natural languages the client can handle</td>
</tr>
<tr>
<td>Host</td>
<td>Request</td>
<td>The server’s DNS name</td>
</tr>
<tr>
<td>Authorization</td>
<td>Request</td>
<td>A list of the client’s credentials</td>
</tr>
<tr>
<td>Cookie</td>
<td>Request</td>
<td>Sends a previously set cookie back to the server</td>
</tr>
<tr>
<td>Date</td>
<td>Both</td>
<td>Date and time the message was sent</td>
</tr>
<tr>
<td>Upgrade</td>
<td>Both</td>
<td>The protocol the sender wants to switch to</td>
</tr>
<tr>
<td>Server</td>
<td>Response</td>
<td>Information about the server</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>Response</td>
<td>How the content is encoded (e.g., gzip)</td>
</tr>
<tr>
<td>Content-Language</td>
<td>Response</td>
<td>The natural language used in the page</td>
</tr>
<tr>
<td>Content-Length</td>
<td>Response</td>
<td>The page’s length in bytes</td>
</tr>
<tr>
<td>Content-Type</td>
<td>Response</td>
<td>The page’s MIME type</td>
</tr>
<tr>
<td>Last-Modified</td>
<td>Response</td>
<td>Time and date the page was last changed</td>
</tr>
<tr>
<td>Location</td>
<td>Response</td>
<td>A command to the client to send its request elsewhere</td>
</tr>
<tr>
<td>Accept-Ranges</td>
<td>Response</td>
<td>The server will accept byte range requests</td>
</tr>
<tr>
<td>Set-Cookie</td>
<td>Response</td>
<td>The server wants the client to save a cookie</td>
</tr>
</tbody>
</table>
Sample HTTP Request and Response

Request: GET /rfc.html HTTP/1.1 Host:www.ietf.org
Response:

Trying 4.17.168.6...
Connected to www.ietf.org.
Escape character is ' \''.
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: avoid browser bug

<html>
<head>
<title>IETF RFC Page</title>

<script language="javascript">
function url() {
    var x = document.form1.number.value
    if (x.length == 1) {x = "000" + x }
    if (x.length == 2) {x = "00" + x }
    if (x.length == 3) {x = "0" + x }
    document.form1.action = "/rfc/rfc" + x + ".txt"
    document.form1.submit
}
</script>

</head>
The World Wide Web (WWW)

! The World Wide Web is a large scale, on-line repository of information that users can search using an interactive application program called a browser.

! Technically, the web is a distributed hypermedia system that supports interactive access.

! Information is stored in a set of documents.

! Besides the basic information, a document can contain pointers to other documents in the set.

! A hypermedia document available on the Web is called a page. The root page for an organization or individual is called the homepage.

! Each Web page that contains a hypermedia document uses a standard representation known as Hypertext markup language (HTML).

! HTML allows an author to give general guidelines for display and to specify the contents of the page. (Consequently, two browsers may display a page differently.)
Browser Architecture

A browser interacts with a Web server using HTTP.

A browser consists of a set of clients, a set of interpreters, and a controller that manages them.

Major components of a Web browser. Dark arrows show the flow of data; other arrows show control paths. The data paths from clients to interpreters are not shown.
HTML Format and Representation

! Each HTML document consists of two parts: a **head** followed by a **body**.

! Each HTML document is represented as a text file that contains **tags** along with other information.

! HTML tags provide structure for the document as well as formatting hints.

! General Form of an HTML Document:

```
< HTML>
  < HEAD>
    < TITLE>
      text that forms the document title
    < /TITLE>
  < /HEAD>

  < BODY>
    body of the document appears here
  < /BODY>
< /HTML>
```

! Equivalent Document:

```
< HTML> < HEAD> < TITLE> text that forms the document title< /TITLE> < /HEAD> < BODY> body of the document appears here< /BODY> < /HTML>
```
Example HTML Formatting Tags

! Line Breaks:
Hello there.<br>This is an example<br>of HTML

Hello there.
this is an example
of HTML

! Headings:
Hello.<br><h1>Heading</h1><br>normal

Hello.
Heading
normal

! Lists:
Here is a list of 4 names:
<ul>
<li>Scott</li>
<li>Sharon</li>
<li>Jan</li>
<li>Rebecca</li>
</ul>
This text occurs after the list

Here is a list of 4 names:
! Scott
! Sharon
! Jan
! Rebecca
This text occurs after the list.
Embedding Graphical Images Into a Web Page

Example:
Here is a picture. `<IMG SRC= "myphoto.gif" ALIGN= middle>`

Illustration of image alignment. As requested in the tag, text on the line is positioned in the middle of the image.
Hypertext Links to Other Documents

! The HTML mechanism for specifying a hypertext reference is called an anchor.

! Example:
This book is published by
< A HREF= "http://www.prenhall.com" >
Prentice-Hall, < /A > one of the larger publishers of
Computer Science textbooks.

! Example Output:
This book is published by Prentice-Hall, one of the
larger publishers of Computer Science textbooks.
Optional Clients

Example of an FTP Link:
Many of the examples in this text are available online. Source code from
< A HREF= "ftp://ftp.site.com/netbook/prog.c"> an example C program
< /A>
or code from a
< A HREF= "ftp://ftp.site.com/netbook/prog2.c"> second example C program
< /A>
are available

Example Output:
Many of the examples in this text are available online. Source code from an example C program or
code from a second example C program are available
A Selection of Common HTML Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;html&gt; ... &lt;/html&gt;</td>
<td>Declares the Web page to be written in HTML</td>
</tr>
<tr>
<td>&lt;head&gt; ... &lt;/head&gt;</td>
<td>Delimits the page’s head</td>
</tr>
<tr>
<td>&lt;title&gt; ... &lt;/title&gt;</td>
<td>Defines the title (not displayed on the page)</td>
</tr>
<tr>
<td>&lt;body&gt; ... &lt;/body&gt;</td>
<td>Delimits the page’s body</td>
</tr>
<tr>
<td>&lt;h n&gt; ... &lt;/h n&gt;</td>
<td>Delimits a level $n$ heading</td>
</tr>
<tr>
<td>&lt;b&gt; ... &lt;/b&gt;</td>
<td>Set ... in boldface</td>
</tr>
<tr>
<td>&lt;i&gt; ... &lt;/i&gt;</td>
<td>Set ... in italics</td>
</tr>
<tr>
<td>&lt;center&gt; ... &lt;/center&gt;</td>
<td>Center ... on the page horizontally</td>
</tr>
<tr>
<td>&lt;ul&gt; ... &lt;/ul&gt;</td>
<td>Brackets an unordered (bulleted) list</td>
</tr>
<tr>
<td>&lt;ol&gt; ... &lt;/ol&gt;</td>
<td>Brackets a numbered list</td>
</tr>
<tr>
<td>&lt;li&gt; ... &lt;/li&gt;</td>
<td>Brackets an item in an ordered or numbered list</td>
</tr>
<tr>
<td>&lt;br&gt;</td>
<td>Forces a line break here</td>
</tr>
<tr>
<td>&lt;p&gt;</td>
<td>Starts a paragraph</td>
</tr>
<tr>
<td>&lt;hr&gt;</td>
<td>Inserts a horizontal rule</td>
</tr>
<tr>
<td>&lt;img src=&quot;...&quot;&gt;</td>
<td>Displays an image here</td>
</tr>
<tr>
<td>&lt;a href=&quot;...&quot;&gt; ... &lt;/a&gt;</td>
<td>Defines a hyperlink</td>
</tr>
</tbody>
</table>
Sample HTML Page

<html>
<head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit Amalgamated Widget's home page. We hope you will find all the information you need here. <br>
Below we have links to information about our many fine products. You can order electronically (by WWW), by telephone, or by fax. </p>
<br>
<h2> Product information </h2>
<ul>
<li> <a href="http://widget.com/products/big"> Big widgets </a> 
<li> <a href="http://widget.com/products/little"> Little widgets </a>
</ul>
<h2> Telephone numbers </h2>
<ul>
<li> By telephone: 1-800-WIDGETS 
<li> By fax: 1-415-765-4321 
</ul>
</body>
</html>
Sample HTML Table

Some Differences between HTML Versions

<table>
<thead>
<tr>
<th>Item</th>
<th>HTML 1.0</th>
<th>HTML 2.0</th>
<th>HTML 3.0</th>
<th>HTML 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperlinks</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Images</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Lists</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Active Maps and Images</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Forms</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Equations</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Toolbars</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Tables</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accessibility features</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Object embedding</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Scripting</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Sample HTML Form

<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/widgetorder" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street Address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size=4> Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10> Expires <input name="expires" size=4> 
M/C <input name="cc" type=radio value="mastercard"> VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive"> Little <input name="product" type=radio value="cheap"> Ship by express courier <input name="express" type=checkbox> </p>
<p></p>
<p> Thank you for ordering an AWI widget, the best widget money can buy! </p>
</form>
</body>
</html>
Sample Browser Form Response

! When the user clicks the "Submit" button, the browser packages the collected information into a single long line and sends it back to the server for processing.

! The '&' separates fields and the '+' is used to represent a space.

customer=John+Doe&address=100+Main+St.&city=White+Plains&state=NY&country=USA&cardno=1234567890&expires=6/98&cc=mastercard&product=cheap&express=on
XML and XSL

XML (eXtensible Markup Language) and XSL (eXtensible Style Language) allow Web pages to be structured for automated processing.

XML describes content and XSL describes the formatting independently of the content.

Sample XML Web Page:

```xml
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="b5.xsl"?>
<book_list>
  <book>
    <title> Computer Networks, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2003 </year>
  </book>
  <book>
    <title> Modern Operating Systems, 2/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 2001 </year>
  </book>
  <book>
    <title> Structured Computer Organization, 4/e </title>
    <author> Andrew S. Tanenbaum </author>
    <year> 1999 </year>
  </book>
</book_list>
```
Sample XSL Style Sheet

```xml
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <html>
      <body>
        <table border="2">
          <tr>
            <th>Title</th>
            <th>Author</th>
            <th>Year</th>
          </tr>
          <xsl:for-each select="book_list/book">
            <tr>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="author"/></td>
              <td><xsl:value-of select="year"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```
XHTML - eXtended HTML

! XHTML is the new Web standard and should be used for maximum portability.

! XHTML is essentially HTML 4 reformulated in XML. (To get the HTML effect, a definition is needed in the XSL file.

! There are six major differences between XHTML and HTML 4:

1. XHTML pages and browsers must strictly conform to the standard.

2. All tags and attributes must be in lower case.

3. Closing tags are required.

4. Attributes must be contained within quotation marks.

5. Tags must nest properly.

Type of Web Documents

! Static
A static Web document resides in a file at a Web server. The contents do not change. Each request for the document results in exactly the same response.

! Dynamic
A dynamic Web document does not exist in a predefined form. It is created by a Web server whenever a browser asks for the document. The contents of a dynamic document can vary from one request to another.

! Active
An active document is not fully specified by the server. An active document consists of a computer program that understands how to compute and display values. The browser runs the program locally. Thus, the contents of an active document can change continuously.
The Common Gateway Interface (CGI)

The CGI standard specifies how a server interacts with an application program that implements a dynamic document. The application is called a **CGI program**.

CGI programs can generate arbitrary document types (e.g., plain text, HTML text, digital image, etc.). A CGI program can place a header on its output to describe the document type.

**Sample CGI Program:**

```bash
#!/bin/sh

# CGI script that prints the date and time at which it was run
#
# Output the document header followed by a blank line

echo Content/type: text/plain

echo

# Output the date

echo This document was created on `date`

An example CGI program written in UNIX shell language.
```

**Program Output:**

Content/type: text/plain

This document was created on Wed Dec 12 14:19:37 EST 1996
CGI Parameters and Environment Variables

CGI allows a server to pass arguments to a CGI program whenever the program is invoked.

<table>
<thead>
<tr>
<th>Name of Variable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVER_NAME</td>
<td>The domain name of the computer running the server.</td>
</tr>
<tr>
<td>GATEWAY_INTERFACE</td>
<td>The version of the CGI software the server is using.</td>
</tr>
<tr>
<td>SCRIPT_NAME</td>
<td>The path in the URL after the server name.</td>
</tr>
<tr>
<td>QUERY_STRING</td>
<td>Information following &quot;?&quot; in the URL.</td>
</tr>
<tr>
<td>REMOTE_ADDR</td>
<td>The IP address of the computer running the browser that sent the request.</td>
</tr>
</tbody>
</table>

Examples of environment variables passed to a CGI program.
A CGI Script with Long-Term State Information

```bash
#!/bin/sh
FILE=ipaddr

echo Content/type: text/plain
echo

# See if IP address of browser's computer appears in our file
if grep -s $REMOTE_ADDR $FILE >/dev/null 2>&1 then
    echo Computer $REMOTE_ADDR has requested this URL previously.
else
    # Append browser's address to the file
    echo $REMOTE_ADDR >> $FILE
echo This is the first contact from computer $REMOTE_ADDR
fi

An example CGI program that stores long-term state information.
```

! First Access:
    This is the first contact from computer 128.10.2.36

! Subsequent Accesses:
    Computer 128.10.2.26 has requested this URL previously.
A CGI Script with Short-Term State Information

```sh
#!/bin/sh

echo Content-type: text/html
echo

N=$QUERY_STRING
echo "<HTML>"

case "x$N" in
  x)
    N=1
    echo "This is the initial page.<BR><BR>"
    ;;
  x[0-9]*)
    N=`expr $N + 1`
    echo "You have refreshed this page $N times.<BR><BR>"
    ;;
  *)
    echo "The URL you used is invalid.</HTML>"
    exit 0
    ;;
esac

echo "<A HREF=""http://$SERVER_NAME$SCRIPT_NAME?$N"">"
echo "Click here to refresh the page.</A> </HTML>"
```

An example CGI program that maintains short-term state.
Program Output:
Content-type: text/html

< HTML>
This is the initial page.< BR> < BR>
< A HREF= "http://www.nonexist.com/cgi/ex4?1">
Click here to refresh the page.</ A> < /HTML>

Browser Interpretation:
This is the initial page.

Click here to refresh the page.

Program Output After First Invocation:
Content-type: text/html

< HTML>
You have refreshed this page 1 times.< BR> < BR>
< A HREF= "http://www.nonexist.com/cgi/ex4?2">
Click here to refresh the page.</ A> < /HTML>

Browser Interpretation:
You have refreshed this page 1 times.

Click here to refresh the page.
PHPl Hypertext Precprocessor

PHP embeds small scripts inside HTML pages and has them executed by the server to generate the page.

Example: Web page with form - PHP script + Output

```html
<html>
<body>
<form action="action.php" method="post">
<p>Please enter your name: <input type="text" name="name"/></p>
<p>Please enter your age: <input type="text" name="age"/></p>
<input type="submit"/>
</form>
</body>
</html>

(a)

<html>
<body>
<h1>Reply: </h1>
Hello <?php echo $name; ?>.
Prediction: next year you will be <?php echo $age + 1; ?>
</body>
</html>

(b)

<html>
<body>
<h1>Reply: </h1>
Hello Barbara.
Prediction: next year you will be 25
</body>
</html>

(c)
```
Active Documentation Representation and Translation

Illustration of three active document representations and the programs that translate or transport the document. The darkened arrow shows the direction a document moves.
Java Technology

Java is the name of a specific technology used to run active documents.

Java uses the **applet** to describe active document programs.

Key Components:
- Programming Language:
  Java includes a new programming language.
- Runtime Environment:
  Provides the facilities needed to run a Java program.
- Class Library:
  A library of software to perform many of the chores that an applet performs.

A browser that runs Java contains an interpreter to run java applets.

Invoking the Applet:

```html
<applet codebase= "www.nonexist.edu/example" code= bbb.class>
```
import java.applet.*;
import java.awt.*;

public class clickcount extends Applet {
    int count;
    TextField f;

    public void init() {
        count = 0;
        add(new Button("Click Here"));
        f = new TextField("The button has not been clicked at all.");
        f.setEditable(false);
        add(f);
    }

    public boolean action(Event e, Object arg) {
        if (((Button) e.target).getLabel() == "Click Here") {
            count += 1;
            f.setText("The button has been clicked " + count + " times.");
        }
        return true;
    }
}

An example applet that counts the number of times a user clicks a button.
An Example Java Applet cont.

! Initial Applet Output:

![Click Here] The button has not been clicked at all.

Illustration of the display after the applet begins execution.

! Subsequent Applet Output:

![Click Here] The button has been clicked 1 times.

The display after the user has clicked the button once.
import java.applet.*;
import java.net.*;
import java.awt.*;

public class buttons extends Applet {

    public void init() {
        add(new Button("Ying"));
        add(new Button("Yang"));
    }

    public boolean action(Event e, Object arg) {
        if (((Button) e.target).getLabel() == "Ying") {
            try {
                getAppletContext().showDocument(new URL("http://www.nonexist.com/yang"));
            }
            catch( Exception ex ) {
                // note: code to handle the exception goes here //
            }
        } else if (((Button) e.target).getLabel() == "Yang") {
            try {
                getAppletContext().showDocument(new URL("http://www.other.com/yang"));
            }
            catch( Exception ex ) {
                // note: code to handle the exception goes here //
            }
        } return true;
    }

    An example applet that interacts with the HTTP and HTML facilities in a browser.
Applet Output

Illustration of the display produced by the applet
Client-Side Scripting: JavaScript

HTML 4.0 allows the embedding of scripts to be executed on the client machine.

The most popular client-side scripting language is JavaScript.

Example: JavaScript for Form Processing:

```html
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
    var person = test_form.name.value;
    var years = eval(test_form.age.value) + 1;
    document.open();
    document.writeln("<html> <body>");
    document.writeln("Hello " + person + ".<br>");
    document.writeln("Prediction: next year you will be " + years + ".");
    document.writeln("</body> </html>");
    document.close();
}
</script>
</head>
<body>
<form>
Please enter your name: <input type="text" name="name">
<p>
Please enter your age: <input type="text" name="age">
<p>
<input type="button" value="submit" onclick="response(this.form)"
</form>
</body>
</html>
```
Example: Page that Responds to Mouse Movements

```html
<html>
<head>
<script language="javascript" type="text/javascript">
if (!document.myurl) document.myurl = new Array();
function pop(m) {
    var urx = "http://www.cs.vu.nl/ast/im/cat.jpg";
    popupwin = window.open(document.myurl[m],"mywind","width=250,height=250");
}
</script>
</head>
<body>
<p><a href="#" onMouseover="pop(0); return false;" >Kitten</a> </p>
<p><a href="#" onMouseover="pop(1); return false;" >Puppy</a> </p>
<p><a href="#" onMouseover="pop(2); return false;" >Bunny</a> </p>
</body>
</html>
```
Summary of Ways to Generate and Display Content

Client machine
- XSL interpreter
- XML interpreter
- HTML interpreter
- JavaScript interpreter
- Plug-in

Browser

Helper

Server machine
- Server
- CGI script
- PHP
- JSP
- ASP

Copyright © December 21, 2004 by Chaim Ziegler, Ph.D.