World Wide Web: URLs, HTML, HTTP

World Wide Web (WWW)

- created in 1989 at CERN by Tim Berners-Lee
- ideas (hypertext, information store) are much older
- linked retrieval of text, images, sounds, movies, ...
- also executable programs (Java)
- predecessor: gopher, whois
- hypertext = linking objects (text) *across a network*
- integrates data access protocols
- graphical browser ➠ Internet for non-techies
- currently most important Internet application (next to email)

WWW components

names: Universal Resource Locator (URL)

retrieval: Hypertext transfer protocol (HTTP)

hypertext: Hypertext markup language (HTML)

browser: retrieve URL via HTTP, ftp, telnet, …from server

server: stores objects for one or more host names

➠ just a text-based RPC mechanism…

URL

scheme://*host*[:*port*]/*path*/[;*parameters*][#*fragment*][?*query*]

```
http://www.cs.columbia.edu:80/~hgs/
http://www.research.att.com/cgi-bin/search?www
mailto:president@whitehouse.gov
```

scheme: http, ftp, gopher, telnet, mailto, news, ...

host: host name or dotted quad

port: TCP port, if not default

path: file name path

parameters: for partial retrieval, etc.

fragment: named position in text

query: parameter for database queries

➠ no clear semantic distinction, but matters to server interfaces

HTML

- simple document type for SGML (standard general markup language)
- describes document structure, not formatting (kind of…)
- provision for hyperlinks
- can be …
  - written by hand
  - generated by special authoring tools
  - translated from LaTeX, nroff, Microsoft Word, …

HTML structure

- `<tag parameter=value ...>` text `</tag>`
- values in `"` if not strictly alphabetic
- uses ISO 8859-1 (including ä,ö,ü, etc.)
- can be searched locally ($\leftrightarrow$ PostScript)
- display reasonably on wide range of hardware (ASCII terminal, different resolutions, braille, audio output, ...)
- has tables, but currently no mathematics
- some standardized (HTML 2.0, HTML 3.2), much vendor-introduced (Netscape, Microsoft)
- constantly evolving

HTML document structure

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Draft//EN">
<HTML>
<HEAD>
<TITLE>Title of document</TITLE>
</HEAD>
<BODY>
The body of the document
</BODY>
</HTML>
```

- tags are case insensitive
- all but TITLE optional (but. . .)
- few use DOCTYPE

HTML tags

- headings: `H1` ... `H6`
- paragraph: `P`
- preformatted text: `PRE`
- quote: `BLOCKQUOTE`
- unordered list:

```
<UL>
<LI>first list item
<LI>second list item
</UL>
```

- ordered list

```
<OL>
```

```
<LI>first list item
<LI>second list item
</OL>
```

- definition list

```
<DL>
<DT>HTML
<DD>Hypertext markup language
<DT>HTTP
<DD>Hypertext transport protocol
</DL>
```

HTML phrase markup

- typographic elements (“how to show this”)
  - **bold**: `<B>bold</B>`
  - *italic*: `<I>italic</I>`
  - `teletype: <tt>text</tt>`
  - blinking text: `<blink>avoid!</blink>`
- idiomatic elements (“what is this”)
  - citation: `<cite>`
  - emphasis: `<em>`
  - strong emphasis: `<strong>`
  - variable: `<var>`
  - ...

HTML miscellaneous

- horizontal rule: `<HR>`
- line break: `<BR>`
- automatic playback of sounds
- backgrounds
- frames
- font size changes
- ...

HTML links

- source: `<a href="url">word</a>` ➠ usually shown underlined and blue, followed when clicked on
- destination within document (#fragment): `<a name="frag">word</a>` ➠ not visible to user
- `<IMG SRC="jim.jpg" ALT="Jim's picture">` displays GIF, JPEG *concurrently*, without user intervention

Style Sheets

```
<HTML>
  <HEAD>
    <TITLE>title</TITLE>
    <LINK REL=STYLESHEET TYPE="text/css"
      HREF="http://style.com/cool" TITLE="Cool">
    <STYLE TYPE="text/css">
      @import url(http://style.com/basic);
      H1 { color: blue }
    </STYLE>
  </HEAD>
  <BODY>
    <H1>Headline is blue</H1>
    <P STYLE="color: green">paragraph is green.
  </BODY>
</HTML>

BODY {
  color: black;
  background: url(texture.gif) white;
}

H1, H2, H3 {   /* all big headers */
```

```
    font-weight: bold;
    font-size: 12pt;
    line-height: 14pt;
    font-family: helvetica;
    font-style: normal
}
H1 EM { color: red }
UL UL LI { font-size: x-small }
P:first-line { font-style: small-caps }
```

Cascading: user, document, …

nesting ➠ inheritance

HTTP

- simple ASCII-based transfer protocol
- methods:

  GET: retrieve object, no side effects

  HEAD: get descriptive information

  PUT: replace object on server

  POST:

  others: PATCH, COPY, MOVE, …
- stateless ➠ every query independent
- one request ➠ one TCP connection (client opens, server closes) ➠ inefficient ➠ HTTP 1.1 connections
- uses ftp-style error codes

HTTP request

```
GET /index.html HTTP/1.0
Accept: text/html, */*
If-modified-since: Sat, 29 Oct 1994 19:43:31 GMT
Referer: http://www.w3.org/hypertext/
From: webmaster@w3.org
```

(empty line marks end of request)

HTTP response

```
HTTP/1.0 200 Document follows
Date: Wed, 31 Jan 1996 20:45:17 GMT
Server: NCSA/1.5
Content-type: text/html
Content-language: en
Last-Modified: Wed, 31 Jan 1996 20:00:10 GMT

<html>
<head>
...
</html>
```

(server closes connection)

HTTP/1.1

RFC 2068

- `Host:` field for vanity URLs
- unless `Connection: close`, keep open
- chunking: blocks of data for incremental generation
- better cache control

Web Security

- authentication
  - basic authentication
  - digest authentication
- privacy ➠ SSL (aka TSL)

HTTP access control - basic

- client attempts access (GET, PUT, . . .) normally
- server returns

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: Basic realm="WallyWorld"
```

- client tries again

```
Authorization: Basic base64-encoded-user:password
```

- passwords in the clear ➠ not secure

HTTP access control - digest

RFC 2069
First attempt for `http://www.nowhere.org/dir/index.html`:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="testrealm@host.com",
                  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                  opaque="5ccc069c403ebaf9f0171e9517f40e41",
                  algorithm=MD5
```

Browser prompts for username (Mufasa) and password (CircleOfLife), retries:

```
Authorization: Digest username="Mufasa",
               realm="testrealm@host.com",
               nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
               uri="/dir/index.html",
               response="e966c932a9242554e42c8ee200cec7f6",
               opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

- `nonce`: (hex, base64) is opaque to client
- nonce = H(client-IP ":" time-stamp ":" only-server-knows)

- `opaque`: return unchanged
- `algorithm`: algorithm for hash H()
- KD(secret, data) = H(concat(secret, ":", data))
- `response`: KD(H(A1), nonce ":" H(method:uri)
- A1 = user:realm:password
- server only needs H(A1)

Caching and proxies

- avoid repeated transfers ➠ cache object in memory, disk or on a proxy
- proxy:
  - only for HTTP (not ftp, etc.)
  - receive normal HTTP request, but with full (host) URL
  - check if available locally and still current (HEAD If-modified-since)
  - if not, retrieve, return to client and store
- also used for firewalls

cgi-bin

- e.g.: http://www.cs.columbia.edu/ hgs/test/test-cgi.cgi/foo?bar
- web server invokes as script if

```
SERVER_SOFTWARE = Apache/1.1.1
SERVER_NAME = www.cs.columbia.edu
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.0
SERVER_PORT = 80
REQUEST_METHOD = GET
HTTP_ACCEPT = image/gif, image/jpeg, */*
PATH_INFO = /foo
PATH_TRANSLATED = /import/http/html/foo
SCRIPT_NAME = /~hgs/test/test-cgi.cgi
QUERY_STRING = bar
REMOTE_HOST = erlang.cs.columbia.edu
REMOTE_ADDR = 128.59.27.35
```

- Request headers ➠ `HTTP_`.
- argv contain queries
- stdin: entity of POST, PUT, ...
- `nph-`: generate full headers
- otherwise: Script outputs entity header, CRLF, content

HTML forms

- allows user to input text, select choices, files, ...
- choices:

  action: URL (usu. cgi-bin), mailto

  method: GET, POST

  encoding: structure

```
<FORM METHOD="POST" ACTION="http://www.w3.org/cgi-bin/form">
Your name: <INPUT NAME="name" SIZE="48"><p>
Male: <INPUT NAME="gender" TYPE=RADIO VALUE="male"><p>
Female: <INPUT NAME="gender" TYPE=RADIO VALUE="female"><p>
Cities in which you maintain a residence:<p>
<ul>
<li>Kent <INPUT NAME="city" TYPE="checkbox" VALUE="kent">
<li>Miami <INPUT NAME="city" TYPE="checkbox" VALUE="miami">
</ul>
<p><INPUT TYPE=SUBMIT> <INPUT TYPE=RESET></FORM>
```

Action also: `mailto:`

Forms: encoding types

Method can be GET or POST. See RFC 1867.

application/x-www-form-urlencoded: (RFC 1866)

- *field1=value1&field2=value2*
- space $\rightarrow$ +, specials, + $\rightarrow$ %HH
- unpack using (e.g.) uncgi (➠ env. variables), Perl library, …

multipart/form-data: for uploads (RFC 1867)

Multipart-Formdata: Example

```
CONTENT_TYPE=multipart/form-data;
  boundary=---------------------------28032664210092

-----------------------------28032664210092
Content-Disposition: form-data; name="user"

your name
-----------------------------28032664210092
Content-Disposition: form-data; name="flavor"

a
-----------------------------28032664210092
Content-Disposition: form-data; name="flavor"

b
-----------------------------28032664210092
Content-Disposition: form-data; name="submit"

POST, form-data
-----------------------------28032664210092--
```

State management: cookies

RFC 2109
Server - browser:

```
HTTP/1.1 200 OK
Set-Cookie: Customer="WILE_E_COYOTE"; Version="1"; Path="/acme"
```

Browser to Server: Pick item for shopping basket

```
POST /acme/pickitem HTTP/1.1
Cookie: $Version="1"; Customer="WILE_E_COYOTE"; $Path="/acme"

HTTP/1.1 200 OK
Set-Cookie: Part_Number="Rocket_Launcher_0001";
  Version="1"; Path="/acme"

POST /acme/shipping HTTP/1.1
Cookie: $Version="1"; $Version="1";
  Customer="WILE_E_COYOTE"; $Path="/acme";
  Part_Number="Rocket_Launcher_0001"; $Path="/acme"

HTTP/1.1 200 OK
Set-Cookie: Shipping="FedEx"; Version="1"; Path="/acme"

POST /acme/process HTTP/1.1
```

```
Cookie: $Version="1";
   Customer="WILE_E_COYOTE"; $Path="/acme";
   Part_Number="Rocket_Launcher_0001"; $Path="/acme";
   Shipping="FedEx"; $Path="/acme"
```

JavaScript

JavaScript	Java
interpreted by client	compiled on server to byte code
object-based, no classes	object-oriented
embedded in HTML	applets
loose typing	strong typing
dynamic binding	static binding
cannot write to disk	same

JavaScript

```
<SCRIPT LANGUAGE="JavaScript">
document.write("Hello net.")
</SCRIPT>
```

JavaScript Functions

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function square(i) {
    document.write("The call passed ", i ," to the
function.","<HR>")
    return i * i
  }
  document.write("The function returned ", square(5), ".")
</SCRIPT>
</HEAD>
<BODY>
<BR>
All done.
</BODY>
```

Scripting

Add event handlers to HTML tags:

```
<INPUT TYPE="button" VALUE="Calculate"
  onClick="compute(this.form)">
```

Events: blur (remove focus), click, change, focus, load, mouseover, select, submit, unload

Notes

- variables are global, unless declared with `var`
- strings in ' or "
- eval for evaluation
- comments: //, /* */
- statements: basically C

http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html

Objects

```
myCar.make = "Ford"
myCar.model = "Mustang"
myCar.year = 69;
```

Equivalent:

```
mycar["make"] = "Ford
myCar["model"] = "Mustang"
myCar["year"] = 69;
```

Methods

```
object.methodname = function_name
object.methodname(params);
```

Problems

- locations may change ➠ “dangling” pointers ➠ indirection (URNs)
- little descriptive information ➠ URC
- no privacy, strong authentication ➠ SSL (generic TCP), SHTTP
- finding objects ➠ directories, “spiders”
- caching, proxies $\leftrightarrow$ charging, counting “hits”