

# HTTP 2

---

## Recall

See previous handout...

HTML tags

HTML -- structure, not appearance

## Appearance on Client

Exact appearance determined on client...

Font metrics

Screen/window size

Preference of user (what size fonts do you like for regular reading)

"Pour" feature -- HTML pours into available screen width

## HTML vs. Designer Mindset

Traditional media designers, possibly biased by their print/film background, want to control the exact appearance on the client.

However, this is not what HTML does, and in fact, it's practically impossible given the variance in fonts, screen size, ...

Result designers spend a lot of energy fiddling with the HTML, and in fact it only has the look they had in mind on the browser/OS they tested on.

Conclusion to communicate with your audience, straight HTML is a great strategy. It's the least work, and it has the best chance of working on all platforms, phones, etc.. Tweaked HTML only makes sense if exact appearance is critical and there's a large enough budget to test on the many browser/OS combinations. (Note designers always think exact appearance is justified -- that's their mindset. There needs to be a more concrete justification of why straightforward HTML is not good enough.)

## URLs

See previous handout

HTTP URL: scheme, host, path

"Base" URL of a document

Relative URLs

Root relative URLs

% encoding, '+' for spaces

mailto scheme "mailto:nick@cs.stanford.edu?subject=hello+there"

<a href=xxxx target="\_blank">hello</a> -- open in new window

## Image Tag

`<img src = "URL of image">`

Each image is a separate HTTP request -- HTML page with 10 images = 11 requests total. One for the first page + 1 for each image.

HTTP 1.1 allows these multiple requests to use a single TCP connection, instead of making a new one for each request.

Include `width=xxx height=xxx` bindings in the `<IMG>` tag, so the browser can lay out the page correctly before the image loads

## Image Files

Image file formats -- compression

JPEG

public standard -- Joint Photographic Expert Group.

".jpg" or ".jpeg"

Best for photos

24 bit color depth

Used by digital cameras, scanning,

Adjustable lossy compression -- 10x or 20x compression. Does a very good job. At 10x, you can't even tell.

JPEG2000 revision is moderately better. Adoption has been slow, in part due to patent concerns. Hopefully, there will be a patent-free path for JPEG2000. (recall "submarine patents").

GIF

".gif"

Best for simple images (the older standard). Unisys charges for the use of the GIF compression scheme in source code, so there is a move afoot to ditch it.

Transparency

Interlaced

Animated (ugh!)

PNG

Portable Network Graphics

An up-and-coming public standard replacement for GIF

Somewhat better features, and avoids the patent problems of GIF

SVG

Scalable vector graphics

Like PDF, but more standard and lightweight

NOT a bitmap -- a series of drawing commands. 100x more compact than a pixel image + can support animation and other transforms driven by javascript.

Uses XML

Much smaller and better looking to distribute, say, a banner advertisement animation, as an SVG (drawing commands) than an animated gif (pixels)

Usually put the image file in the same directory as the referring HTML and then use relative URLs.

## Image Style

Images make a page fun and give it intuitive visual appeal.

However, slow load time is the #1 complaint of web users, and images load 100 times more slowly than text. Therefore, use text for main content, and sprinkle in a few images for fun decoration (that can fill-in after the main content has displayed).

e.g. yahoo.com, slashdot.org, nytime.com

Amateurish sites tend to have a lot of images, since making images is fun, and they're not really thinking about the usability of their site.

# HTTP

## HTTP Request

port 80

Request sent from client to server

Server sends back one response with header and body parts

<http://www.w3.org/Protocols/> -- all sorts of HTTP info

## HTTP "One shot" Transaction -- "stateless"

"Stateless" -- each request/reply pair (HTTP 1.0) uses its own connection. The dialog does not group logically related collections of requests.

The HTTP request/response protocol is "one shot" -- the server fulfills the request and closes the connection.

This keeps the server's job simple, but it makes it harder to design a sequence of pages that appear logically connected, since each HTTP request is separate.

The lack of continuity between one HTTP request and the next is one of the harder things to absorb when designing HTTP client/server systems.

## GET Request

Request a resource from the server

```
GET request HTTP\r\n\r\n
```

```
GET /foo.html HTTP/1.0\r\n\r\n
```

Actually use `\015\012` in your code in case `"\n"` has been mapped to the local end-of-line.

## Complex URLs -- path/file/suffix

The simple description is that everything after the host is the "path"

The more complex description divides things after the host into the path, file, and suffix. That's what we'll use. See the RFC for an even more complex

decomposition of the URL. <http://www.w3.org/Protocols/rfc2616/rfc2616.txt>

```
http://foo.com/a/b/bar.html?a=b
```

```
http://foo.com/a/bar.html#binky
```

scheme or protocol

http

host

foo.com

**path**

/a/b/

**file**after the last /  
bar.html**query suffix**

begins with a ?

Used to send "pi=3.14" bindings to a CGI program on the server

http://foo.com/boo?pi=3.14&amp;name=nick

Specify the "GET" method in a form --&gt; "?" query suffix is used for the parameters

Query parameters are sent as part of the HTTP GET request

**fragment suffix**

May contain almost anything

Safest strategy: search for a '#' left-to-right in the URL. From # onwards is the fragment.

The fragment is not sent as part of the GET query. It is used by the browser to scroll to the right spot.

The fragment name is defined in the target HTML with a name binding: &lt;a name=here&gt; -- refer to this spot as foo.html#here

## Request

The request is basically the path part of the URL http://foo.com/dir/b.html :

request = /dir/b.html

**? Query Suffix**

A "?a=b&amp;c=13" suffix is sent as the end part of the request -- the server (typically a CGI) does use it as part of the request.

http://foo.com/dir/b.html?bar=http://www.yahoo.com/

request = /dir/b.html?bar=http://www.yahoo.com/

The suffixes may contain weird characters, so identify the "#" or "?" scanning from the left.

**# Fragment Suffix**A "#foo" suffix is **not** sent as part of the request

The "#" part of the url is handled on the client side.

http://foo.com/dir/b.html#bar : request = /dir/b.html

## Basic GET

**telnet 80**

Lab exercise: try telneting to port 80 of your favorite web server. Type the GET directive in by hand and watch what happens. On unix, the command looks like...

&gt;telnet www.stanford.edu 80

**Get Options**

Some servers are confused if the "GET" is in lowercase

Usually the GET includes optional information on subsequent lines. Newer servers may need an HTTP 1.1 request to work...

GET request HTTP/1.0

-- blank line --