HTTP message format

```
<start-line>
<headers>

<entity-body>
```

HTTP Request message format

```
<method> <request-URL> <version>\r\n
<headers>\r\n
\r\n
<entity-body>
```

<method> is the operation to perform on URL
<request-URL> can be full URL or just the path part
<version> is of the form HTTP/<major>.<minor>
<entity-body> is a stream of bytes (could be empty)

```
GET /test/hi-there.txt HTTP/1.1
Accept: text/*
Host: www.joes-hardware.com
```

HTTP Response message format

```
<version> <status> <reason-phrase>\r\n
<headers>\r\n
\r\n
<entity-body>
```

<version> is of the form HTTP/<major>.<minor>
<status> is a 3-digit number indicating status of request
<reason-phrase> human-readable description of status code
<entity-body> is a stream of bytes (could be empty)

```
HTTP/1.0 200 OK
Content-type: text/plain
Content-length: 18

Hi! I’m a message!
```

HTTP Request Methods

- GET – Retrieve document from server
- PUT – Store document on server
- DELETE – Remove document from server
- POST – Send data to server for processing
- HEAD – Retrieve document headers from server
- OPTIONS – Determine what methods the server supports
- TRACE – Trace the path taken by a request through proxy servers on the way to the destination server

HTTP Response status codes

- 100-199 Informational
- 200-299 Successful
- 300-399 Redirection
- 400-499 Client error
- 500-599 Server error

- 200 OK
- 401 Unauthorized to access resource
- 404 Requested resource does not exist

HTTP Headers

- List of name/value pairs
- `Name: Value\r\n`
- Empty line separates headers and entity body

- General headers
  - `Date: Tue, 3 Oct 1974 02:16:00 GMT`
    - Time at which message was generated
  - `Connection: close`
    - Client or server can specify options about the underlying connection

HTTP Request Headers

- `Host: www.joes-hardware.com`
  - Host from the request URL
- `User-Agent: Mozilla/4.0`
  - Client application making the request
- `Accept: text/html, text/xml`
  - MIME types the client can handle
- `Referer: http://www.joes-hardware.com/index.html`
  - Page that contained the link currently being requested
- `If-Modified-Since: Tue, 3 Oct 1974 02:16:00 GMT`
  - Conditional request; only send the document if it changed since I last retrieved it

HTTP Response Headers

- `Content-length: 15023`
  - Length of response entity body measured in bytes
- `Content-type: text/html`
  - MIME type of response entity body
- `Server: Apache/1.2b6`
  - Server software that handled the request
- `Cache-Control: no-cache`
  - Clients must not cache the response document

HTTP is a “stateless” protocol

- Each request/response transaction is unrelated to all previous transactions
- HTTP stores no server-side state between requests
- Any information needed by the server to process a request must be provided as part of the request
- If the server goes down and comes back up, the client can pick up right where it left off (no need to synchronize its state with the server)
- Stateless protocols are simpler, but web sites like to remember things about clients across multiple requests
  - Names, preferences, account information, etc.

HTTP Cookies

- HTTP allows the server to store a small amount of state on the client
- Each request from the client to the server contains the state previously stored by that server on the client
- This way the server can “remember” who the client is and something about them

HTTP Cookies

- Server stores a name/value pair on the client with a Set-Cookie response header
- `Set-Cookie: name=value [; expires=date] [; path=path] [; domain=domain] [;secure]`
- `Set-Cookie: name=“mary”; expires= Tue, 3 Oct 1974 02:16:00 GMT; path=/booksales; domain=amazon.com; secure`
- If “expires” is not specified, the cookie will expire when the user’s session ends (i.e., when the browser is closed)

HTTP Cookies

- Client stores all unexpired cookies from all web sites the user has visited
- When the client sends a request to a server, all cookies that apply to the request URL are placed in the request using Cookie request headers
- `Cookie: name="mary"; account="1234"`
- Server uses the cookie values to personalize the user's interaction with the web site

Web Sessions

- HTTP is stateless
- Web applications are very stateful
- Web “session”
  - Login
  - Use the application => many HTTP requests
  - Logout
- Application needs server-side state to track user activities across multiple requests
- Web servers layer facilities for tracking server-side state on top of HTTP

Cookie-based web sessions

- When user logs in, web application creates a state object for the current user session
- Web application assigns an ID to the session and stores the state object in a fast data structure, using the session ID as the lookup key
- Web application sends the session ID back to the client as a Cookie
  - `Set-Cookie: sessionid=83746`
- Client sends session ID in Cookie header with each HTTP request
  - `Cookie: sessionid=83746`
- Web application extracts session ID from Cookie header and uses it to look up the session's state object
- We now have server-side state

Fat URL-based web sessions

- Some users turn off HTTP cookies, but we still want stateful web applications to work
- Instead of using Cookies to pass the session ID between client and server, the web application embeds the session ID in all URLs passed back to the client
  - `<A href=“http://www.amazon.com/books/computers/83746”> Computer Books</A>`
- Requires web application to embed session ID in all URLs
  - Sometimes called “URL rewriting”