# Web Foundations (URI and HTTP)

## Web Architecture and Information Management [./]
Spring 2009 — INFO 190-02 (CCN 42509)

**Erik Wilde, UC Berkeley School of Information**

2009-02-25

This work is licensed under a [CC Attribution 3.0 Unported License](http://creativecommons.org/licenses/by/3.0/)

## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>2</td>
</tr>
<tr>
<td>Web Server Service</td>
<td>3</td>
</tr>
<tr>
<td>1 Uniform Resource Identifier (URI)</td>
<td>5</td>
</tr>
<tr>
<td>▪ Resource Identification</td>
<td>5</td>
</tr>
<tr>
<td>▪ URI Schemes</td>
<td>6</td>
</tr>
<tr>
<td>2 Hypertext Transfer Protocol (HTTP)</td>
<td>8</td>
</tr>
<tr>
<td>▪ DNS &amp; HTTP</td>
<td>8</td>
</tr>
<tr>
<td>▪ The Web's Protocol</td>
<td>9</td>
</tr>
<tr>
<td>▪ 2.1 HTTP Basics</td>
<td>11</td>
</tr>
<tr>
<td>▪ HTTP Messages</td>
<td>11</td>
</tr>
<tr>
<td>▪ HTTP Header Fields</td>
<td>12</td>
</tr>
<tr>
<td>▪ HTTP Requests</td>
<td>13</td>
</tr>
<tr>
<td>▪ HTTP GET</td>
<td>14</td>
</tr>
<tr>
<td>▪ HTTP Responses</td>
<td>15</td>
</tr>
<tr>
<td>▪ HTTP Performance</td>
<td>16</td>
</tr>
<tr>
<td>▪ HTTP Connection Handling</td>
<td>17</td>
</tr>
<tr>
<td>▪ 2.2 HTTP Authentication</td>
<td>19</td>
</tr>
<tr>
<td>▪ HTTP Access Control</td>
<td>19</td>
</tr>
<tr>
<td>▪ HTTP Authentication</td>
<td>20</td>
</tr>
<tr>
<td>▪ Basic HTTP Authentication</td>
<td>21</td>
</tr>
<tr>
<td>▪ Repeated Access</td>
<td>22</td>
</tr>
<tr>
<td>▪ Login Page</td>
<td>23</td>
</tr>
<tr>
<td>▪ Conclusions</td>
<td>24</td>
</tr>
</tbody>
</table>
Abstract

The Web's architecture has very simple principles revolving around the ideas of placing a heavy emphasis on a consistent and global identification mechanism for resources, a standardized way of how resource representations can be retrieved, and a standardized way of how resource representations should be usable by using standardized media types. Based on the Internet, the Web's transport protocol transmits representations of resources identified by a Uniform Resource Identifier (URI) between Web servers and clients. The most important protocols for data transfer on the Web is the Hypertext Transfer Protocol (HTTP).

Web Server Service

- Web servers do more than just "deliver files"
- They receive a request for acting on a resource
  - this may be a simple file retrieval
  - additional information is available from the request's header fields [HTTP Header Fields (1)]
  - the request URI may contain additional query information
  - the request may transmit complex data (such as a form submission)
- Processing can mean anything, it is transparent for the client
  - the result of processing yields a resource representation
  - in many cases, a Web server is just part of an application
  - the application server is the application-specific logic
Uniform Resource Identifier (URI)

Resource Identification

- The Web is centered around resources
  - HTTP has been designed to manipulate resources
  - HTTP provides methods for getting, putting, updating, and even deleting resources
- Resources are useful abstractions for interfaces
  - instead of an API, interaction is built around manipulating resources
  - APIs change and bind closely, documents can better withstand change and bind loosely
  - the whole Web is built around resources, not APIs

URI Schemes

URI = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

http://dret.net/lectures/web-spring09/foundations#uri-schemes

- URIs in their general case are very simple
  - the scheme identifies how resources are identified
  - the identification may be hierarchical or non-hierarchical
- Many URI schemes are hierarchical
  - it is then possible to use relative URIs such as in a href="../"
  - the slash character is not just a character, in URIs it has semantics
- Query components specify additional information
  - it is non-hierarchical information further identifying the resource
  - in most cases, it can be regarded as "input" to the resource
DNS & HTTP

The two basic protocols which every Web browser must implement are DNS (Domain Name System, which is required as the foundation for HTTP) and HTTP (Hypertext Transfer Protocol). However, most operating systems provide an API for DNS access, so the browser can use this service locally and only has to implement HTTP. TCP (Transmission Control Protocol) is usually provided by the operating system.
HTTP Basics

HTTP Messages

- HTTP needs a reliable connection
  - the foundation for HTTP is the Transmission Control Protocol (TCP) [Internet Architecture; Transmission Control Protocol (TCP) (1)]
  - DNS resolution yields an IP address
  - open TCP connection to port 80 or port specified in URI
    - [http://rosetta.sims.berkeley.edu:8085/]
- HTTP is a text-based protocol
  - the connection is used to transmit text messages
  - all HTTP messages are human-readable (not all entities, though)
  - basic HTTP operations can be carried out by hand

start-line
message-header *
message-body ?

HTTP Header Fields

- Header fields contain information about the message
  - general header: Date as the message origination date
  - request header: Accept-Language indicates language preferences
  - response header: Server contains system information
  - entity header: Content-Type specifies the media type of the entity
- HTTP defines a number of header fields [http://www.cs.tut.fi/~jkorpela/http.html]
  - unknown fields must be ignored (extensibility)
  - unstandardized fields should use a "X-" prefix
- HTTP is about acting on these fields
  - HTTP defines what HTTP implementations must or should do
HTTP Requests

- After opening a connection, the client sends a request
  - the method indicates the action to be performed on the resource
  - HTTP's most interesting methods are: GET, HEAD, POST
  - other interesting methods are: PUT, DELETE
- The URI identifies the resource to which the request should be applied
  - absolute URIs are required when contacting proxies
  - absolute paths are required when contacting a server directly
  - the URI may contain query information
- The Host header field must be included in every request

Method Request-URI HTTP/Major.Minor
[Header]^*
[Entity]?
HTTP Responses

- The server’s response to interpreting a request
  - the status code is given numerically and as text
  - 2xx for variations of “ok”
  - 3xx for redirections
  - 4xx are different client side problems (404: not found)
  - 5xx are different server side problems
- Header fields specify additional information
  - information about the server
  - information about the entity (media type, encoding, language)

HTTP Performance

- HTTP/1.0 allowed one transaction per connection
  - TCP connection setup and teardown are expensive
  - TCP's slow start slows down the initial phase of data transfer
  - typical Web pages use between 10-20 resources (HTML + images + CSS + scripts)
  - typically, these resources are stored on the same server
- HTTP/1.1 introduces persistent connections
  - the TCP connection stays open for some time (10 sec is a popular choice)
  - additional requests to the same server use the same TCP connection
- HTTP/1.1 introduces pipelined connections
  - instead of waiting for a response, requests can be queued
  - the server responds as fast as possible
  - the order may not be changed (there is no sequence number)
HTTP Basics

HTTP Connection Handling

HTTP Access Control

HTTP servers can deny access because of access control:
- 401 Unauthorized means the resource is access controlled
- 403 Forbidden means the resource is inaccessible
- 405 Method Not Allowed signals a request using the wrong request method

Two different approaches to unauthorized access are possible:
- repeat the HTTP request with the proper authentication credentials
- redirect to a Login Page and establish an authenticated Session
HTTP Authentication

![Diagram of HTTP Authentication Process]

Basic HTTP Authentication

- Authentication is based on **authentication realms**
  - a set of resources for which the authentication is required
  - an opaque name which is used to signal which login is required
  - username/password often is specific for a given realm
- Users supply username and password through the client
  - sent as [Base64](http://en.wikipedia.org/wiki/Base64) encoded "username:password" string
  - username and password are **not transmitted securely** ([http://www.google.com/search?q=base64+decoder](http://www.google.com/search?q=base64+decoder))
  - basic authentication should **always** use HTTPS ([Security & Privacy; HTTP over SSL (HTTPS)](http://en.wikipedia.org/wiki/Basic_access_authentication))
- Authorization is handled on the server side

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

`GET /private/index.html HTTP/1.0` Authorizaton: Basic `QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

[http://en.wikipedia.org/wiki/Basic_access_authentication]
Repeate Access (22)

- Clients typically access more than one protected resource
  - a perfectly stateless client would always request authentication from the user
  - using the realm clients can identify repeated accesses
- Web interactions by default are perfectly stateless
  - each request is completely independent from other requests
  - stateless interactions make the Web loosely coupled and scalable
  - concepts like the realm or State Management (Cookies) [State Management (Cookies)] introduce "state"
- Clients remember the authentication and replay it automatically
  - browsers provide little control over this feature
  - "logging out" of HTTP authenticated sessions is hard

Login Page (23)

- Basic HTTP Authentication [Basic HTTP Authentication (1)] works with browser controls (including the window)
  - no possibility to "log out" without using browser-specific controls
  - client side security depends on browser security measures
- Using HTML Forms [HTML Forms] gives more freedom in session management
  - Authentication [Security & Privacy; Authentication (1)] and Authorization [Security & Privacy; Authorization (1)] are completely application-based
  - if there were "secure personal browsers" this would not work very well
Conclusions

- HTTP is much more than file transfer
  - it is a protocol for the concept of resource manipulation
  - it is a distinct step away from the API approach to building distributed systems
- HTTP servers can be configured to deliver good or bad service
  - this is a question of how well they are configured on the HTTP level
  - it is also a question of how good the Web design is
  - both issues together are required to set up a good Web server