World Wide Web, etc.

Alex S.[*]

Raw data-packets wouldn't be much use to humans if there weren't many application level protocols, such as SMTP (for e-mail), HTTP & HTML (for www), etc.

1 The Web

The following discussion applies to CGI, ASP, JSP, Servlets, Web Services, etc. Anything that works with 'the web'.

1.1 Introduction

What people call the WWW (World Wide Web) is just a huge collection of web-servers, which you access via web-clients (or web-browsers).

The process works like this: a person types in some address in the address bar of their web-browser. The address is in the form of a URL, which has both the hostname and the resource name (along with the protocol and possibly a port).

(sample URL with port and resource:

```
http://www.theparticle.com:80/profphreak/profphreak.html
```

where `http` is the protocol, `www.theparticle.com` is the hostname, `80` is the port address on that host [80 is the default for web-servers] and `/profphreak/profphreak.html` is the resource, an html file in our case.)

The web-browser connects to the host, and requests the resource. The server either returns the resource or an error, and user's web-browser displays whatever the server returned.

Simple? That's all there is. Well, as far as the simple situation is concerned.

Very often however, we are not just dealing with static resources such as a web page, but with programs. In that case, the situation works like this: the web-browser requests a resource, which the web-server determines to be an executable program - in which case, the server executes the program, and returns the output as "the resource".

Common Gateway Interface (or CGI for short), defines the environment in which such programs execute. It defines how these 'web' programs get their parameters, how their input and output is processed (how they talk to the web-server), etc.

[*] `alex@theparticle.com`

CGI is system and language independent, meaning that you can write these web-programs in any language for any system (yes, even UNIX and Windows, and yes, using Perl).

1.2 The Request (technical)

All client/server interactions start with a client request. In HTTP, a request is a fairly basic command to the server to fetch some resource. The usual request would look something like this:

```
GET /somefile.cgi HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
.NET CLR 1.0.3705)
Host: localhost:8080
Connection: Keep-Alive
```

This is a request using Microsoft Internet Explorer 6.0 (as you can see from the `User-Agent` header variable). The URL used is:

```
http://localhost:8080/somefile.cgi
```

Which you can also spot inside the header (the `Host` has the "host", and the first `GET` line has the resource.

When the web-server gets such a request, it knows what to `GET` and what types of content the user's browser is willing to accept (which includes Microsoft Word documents, among other things).

The same URL but requested via the Mozilla 1.1 web-browser produces the request:

```
GET /somefile.cgi HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.1) Gecko/20020826
Accept: text/xml, application/xml, application/xhtml+xml,
text/html;q=0.9, text/plain;q=0.8, video/x-mng, image/png,
image/jpeg, image/gif;q=0.2, text/css, */*;q=0.1
Accept-Language: en-us, en;q=0.50
Accept-Encoding: gzip, deflate, compress;q=0.9
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
```

As you can see, even though the requests appear different, they do contain fundamentally the same information (like `Host`, and `GET` line). This is how the server can be platform independent (and be able to talk to various web-clients).

1.3 GET, POST, etc.

What we've seen so far is a `GET` request. There are about half a dozen different requests which are standard. Apart from `GET`, another request types stands out as being very common: the `POST`.

The `POST` request is usually used to send form data to the server. It is also different from `GET` in that it has a body. (`GET` just had the header; all user data has to be crammed into the few lines of the request header). With `POST`, we can send a LOT more data (like upload files), and submit forms with a lot of fields.

To test `POST`, let's create an HTML form, and post data to our URL from the previous section.

```
<HTML>
    <HEAD>
        <TITLE>This is a test.</TITLE>
    </HEAD>
    <BODY>
        <H3>Enter Your Info Form</H3>
        <FORM ACTION="http://localhost:8080/somefile.cgi"
METHOD="POST">
            <P>Name: <INPUT TYPE="TEXT" NAME="USERNAME"><BR>
            Age: <INPUT TYPE="TEXT" NAME="AGE"><BR>
            <INPUT TYPE="SUBMIT" VALUE="Send Info">
        </FORM>
    </BODY>
</HTML>
```

Notice that the `FORM` has a `METHOD="POST"` (this is the key part). When we post data using this form, the request becomes:

```
POST /somefile.cgi HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
.NET CLR 1.0.3705)
```

```
Host: localhost:8080
Content-Length: 27
Connection: Keep-Alive
Cache-Control: no-cache

USERNAME=Prof.Phreak&AGE=25
```

Notice that the request stars with a `POST` and not a `GET` and that now we also have a header argument named `Content-Length` which has a value of `27`.

This `27` represents how much data there is in the body of the request. It is the length of `USERNAME=Prof.Phreak&AGE=25` (this is the body; since the request skips a line right before it).

Now let's do the same exact form with a `GET`:

```
<HTML>
    <HEAD>
        <TITLE>This is a test.</TITLE>
    </HEAD>
    <BODY>
        <H3>Enter Your Info Form</H3>
        <FORM ACTION="http://localhost:8080/somefile.cgi"
METHOD="GET">
            <P>Name: <INPUT TYPE="TEXT" NAME="USERNAME"><BR>
            Age: <INPUT TYPE="TEXT" NAME="AGE"><BR>
            <INPUT TYPE="SUBMIT" VALUE="Send Info">
        </FORM>
    </BODY>
</HTML>
```

Which produces a request to our server:

```
GET /somefile.cgi?USERNAME=Prof.Phreak&AGE=25 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
.NET CLR 1.0.3705)
Host: localhost:8080
Connection: Keep-Alive
```

Notice that this request (the `GET`) has no `Content-Length`, and no request body. All the information is encoded as part of the resource:

```
/somefile.cgi?USERNAME=Prof.Phreak&AGE=25
```

So, now we know that if we use `GET`, our data becomes part of the resource string; and if we use `POST` it is appended to the request as the request body (and we are given the length).

1.4 The Response (technical)

The response is a bit simpler than the request; since now we don't have to worry about various ways of sending data; there is basically one way, and that's in the response body.

An OK response to the web-browser from the web-server looks something like this:

```
HTTP/1.1 200 OK
Date: Fri, 08 Nov 2002 01:31:36 GMT
Server: Apache/2.0.43 (Win32) PHP/4.3.0-pre2
Last-Modified: Fri, 08 Nov 2002 01:21:28 GMT
ETag: "1c5bf-194-d53f0a48"
Accept-Ranges: bytes
Content-Length: 404
Content-Type: text/html; charset=ISO-8859-1

<HTML>
    <HEAD>
        <TITLE>This is a test.</TITLE>
    </HEAD>
    <BODY>
        <H3>Enter Your Info Form</H3>
        <FORM ACTION="http://localhost:8080/somefile.cgi"
METHOD="GET">
            <P>Name: <INPUT TYPE="TEXT" NAME="USERNAME"><BR>
            Age: <INPUT TYPE="TEXT" NAME="AGE"><BR>
            <INPUT TYPE="SUBMIT" VALUE="Send Info">
        </FORM>
    </BODY>
</HTML>
```

Notice that there is an HTTP header, followed by a blank line, then followed by the HTTP body (the content) of the reply. The header has a `Content-Length`, which tells us exactly how many types we can read to get the body. (note that the `404` content length is just a coincidence, it has nothing to do with the `404` error).

An error response would look something like this:

```
HTTP/1.1 404 Not Found
Date: Tue, 12 Nov 2002 06:25:48 GMT
Server: Apache/2.0.43 (Win32) PHP/4.3.0-pre2
```

```
Content-Length: 294
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
 <head>
 <title>404 Not Found</title>
 </head>
 <body>
 <h1>Not Found</h1>
 <p>The requested URL /somefilewhichisnotthere.html was
not found on this server. </p>
      <hr />
      <address>Apache/2.0.43 Server at localhost Port 80</address>
   </body>
</html>
```

Yep, this is the famous 404 Page Not Found error (how our browser sees it in raw form).

1.5 Web Programming

Web programming (or programming for the web - however you want to refer to it), is utilizing the HTTP protocol and the structure we've learned about in these notes.

CGI allows us to define our own recipients of these HTTP messages. We can have our own script that would accept form parameters, process them, and just as easily return results in a form that a browser can understand.