

Forms, CGI

Objectives

- ▶ The basics of HTML forms
- ▶ How form content is submitted
 - ▶ GET, POST
- ▶ Elements that you can have in forms
- ▶ Responding to forms
 - ▶ CGI – the Common Gateway Interface
 - ▶ Later: Servlets
- ▶ Generation of dynamic Web content

HTML forms

- ▶ In most internet programming, you need the user to enter data
- ▶ HTML forms offer the basic user interface elements inside HTML
- ▶ Forms have a method which corresponds to the HTTP command that will be sent when the form is submitted
- ▶ Forms have an action which denotes the URL loaded when the form is sent.
 - ▶ The action URL is typically a CGI or a servlet
- ▶ Inside the form you can have normal HTML and inputs (user interface elements)

Form example

```
<html>
  <body>
    <form action="http://localhost/response.html" method="get">
      your name:
      <input name="someText" type="text" value="change me!" />
      <br />
      your password: <input name="somePass" type="password" />
      <br />
      <input name="theButton" type="submit" value="click me" />
      <br />
    </form>
  </body>
</html>
```

We submit the form to the `SimpleHttpServer` that we wrote last time (an improved version to also accommodate POST)

Form example . . .

your name:

your password:

Form submission

- ▶ Upon submission, the form will generate the following HTTP:
`GET/response.html?someText=change+me%21&somePass=sddsfs&theButton=click+me%21 HTTP/1.1`
`Host: localhost`
`Connection: Keep-Alive`
...and other headers
- ▶ The data of the form is thus sent in the HTTP command, after form's action and ?
- ▶ The format of the data (`inputName=value&...`) is called a query string
- ▶ In the GET method the query string is limited to 65535 chars
- ▶ The GET query string is visible in the browser. Beware of passwords!

POST form

Simply indicate the method POST

```
<html>
  <body>
    <form action="http://localhost/response.html" method="post">
      your name:
      <input name="someText" type="text" value="change me!" />
      <br />
      your password: <input name="somePass" type="password" />
      <br />
      <input name="theButton" type="submit" value="click me!" />
      <br />
    </form>
  </body>
</html>
```

POST form submission

```
POST /response.html HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
...
Host: localhost
```

```
someText=change+me%21&somePass=sdfdsf&theButton=click+me%21
```

- ▶ When sending data with the POST method, the query string is sent after the HTTP empty line marking the end of the HTTP header.
 - So the query string is HTTP content
- ▶ By doing that, the POST method lets you send content with any length (e.g. upload large files)
- ▶ The POST query string is not visible in the browser!
 - You can have both GET-style and POST query strings by

```
<form action="someScript?p1=v1&p2=v2" method="post">
```

Form `<input>`

- ▶ For all HTML inputs you can indicate CSS styles, etc
 - <http://www.htmlhelp.com/reference/html40/forms/input.html>
- ▶ `type="text"` and `type="password"` was demonstrated
- ▶ `type="submit"` creates a submit button.
If you don't set any value, it will be `submit query`
- ▶ Most inputs have a `name=` (not necessarily needed for `type=submit`)
- ▶ Most inputs have a `type=` that determines the user interface element type
- ▶ Most inputs have a `value=` to indicate initial value
- ▶ `type="reset"` creates a button that brings all inputs to their initial values

Form `<textarea>` and `<select>`

Dedicated input elements:

▶ `<textarea name="aText">`
 initial text
 multiline
`</textarea>`
<http://www.htmlhelp.com/reference/html40/forms/textarea.html>

▶ `<select name="aChoice" >`
 `<option value="1">option title</option>`
 `<option value="two">second</option>`
`</select>`
<http://www.htmlhelp.com/reference/html40/forms/select.html>

To indicate an initial value, options can be declared `<option selected ...>`

If the select is declared `<select multiple ...>`, multiple options can be sent

▶ The query string looks like `aChoice=1&aChoice=two` etc, i.e. the name repeats for each value

Checkboxes and radio buttons

- ▶ `<input type="checkbox" name="x" value="y" />`
 - ▶ Typically you will have more checkboxes with the same name
 - ▶ All of the checked boxes will be sent in the query string, with the same name and the respective values, as for `<select multiple >`
- ▶ `<input type="radio" name="x" value="y"/>`
 - ▶ Typically you will have more radio buttons with the same name
 - ▶ Normally only one radio button can be checked

Common Gateway Interface (CGI)

- ▶ CGI is a standard that allows us to write programs that respond to forms
- ▶ A standard HTTP server responds to every request
- ▶ For some requests (typically starting with /cgi-bin/) the server will start a program
- ▶ CGI is the interface between the HTTP server and our program
- ▶ CGI lets us to find out what has been in the HTTP request that the server got
`http://hoofoo.ncsa.uiuc.edu/cgi/overview.html`
`http://www.cgi-resources.com`

The input/output paradigm

- ▶ Normally in DOS or Unix a program reads an input stream (so-called standard input) and writes to an output stream (so-called standard output)
- ▶ A DOS or Unix program also reads its command line arguments, and its environment variables
 - ▶ In DOS you can set an env variable like `set varName=value`
 - ▶ In Unix, it depends on your shell (command line interpreter),
 - ▶ In bash `export varName=value`
 - ▶ In csh `setenv varName value`
 - ▶ For example the PATH environment variable tells the system where to find programs
- ▶ So for input there are: standard input, command line arguments and environment variables
- ▶ The standard output is the only place for program output

CGI program input/output

- ▶ Input: a number of *environment variables* set by the WWW server
- ▶ One of the variables (the `QUERY_STRING`) contains arguments in the form `arg1=value1&arg2=value2&...`
 - ▶ In the GET method the query string is read from the URL, after the '?' sign
`http://yourServer:port/cgi-bin/scriptName?arg1=value1&arg2=value2`
 - ▶ In the POST method the standard input gives the query string
- ▶ Output: the standard output of the CGI program will be sent back to the browser!
 - ▶ Both the HTTP headers and content
 - ▶ Headers, empty line, content
 - ▶ Content is typically HTML but not necessarily

CGI environment variables

- ▶ `SERVER_SOFTWARE`: type of the server
- ▶ `SERVER_NAME`: e.g. `www.nada.kth.se`
- ▶ `SERVER_PORT`: e.g. `80`
- ▶ `REQUEST_METHOD`: `GET` or `POST`
- ▶ `PATH_INFO`: path to your program in the URL, like `/cgi-bin/prog`
- ▶ `PATH_TRANSLATED`: path of the program on disk
- ▶ `SCRIPT_NAME`: name of the CGI program
- ▶ `QUERY_STRING`: actual path of the program
- ▶ `REMOTE_HOST`: host where the request comes from
- ▶ `AUTH_TYPE`: authentication if the user logged-in (e.g. `BASIC`)
- ▶ `REMOTE_USER`: username if the user logged-in
- ▶ `CONTENT_TYPE`: the content-type HTTP header
- ▶ `CONTENT_LENGTH`: the content-length HTTP header (useful in `POST`)

CGI at NADA

- ▶ Put your CGI program in your CGI dir at NADA (if it's activated)

```
/afs/nada.kth.se/public/www.student/cgi-bin/yourUserName/yourProgram
```

- ▶ Make sure that the file has execution rights

```
chmod uo+x yourProgram
cd /afs/nada.kth.se/public/www.student/cgi-bin/yourUserName/
fs setacl . system:anyuser rl
```

- ▶ You should first test your program without a browser
Set the CGI variables by hand using `setenv` (csh) or `export` (bash)

```
setenv QUERY_STRING a=b&c=d
call yourProgram
```

- ▶ When it works, test it with a browser

```
http://cgi.student.nada.kth.se/cgi-bin/yourUserName/yourProgram
```

- ▶ You can check the server error log and try to find your error between other people's errors

```
http://cgi.student.nada.kth.se/cgi-bin/get-errlog
```

A CGI example in PERL

- ▶ PERL = the Practical Extraction and Report Language
 - ▶ `www.perl.com`
 - ▶ `http://broadcast.oreilly.com/2008/09/a-beginners-introduction-to-pe.html`
- ▶ An interpreted programming language inspired by C and shellsript (bash, csh)
- ▶ Available on many platforms but inspired by and started on Unix
- ▶ Very strong pattern matching
- ▶ Easy to use e.g. to make a simple CGI
- ▶ But not for larger applications
- ▶ We just illustrate the CGI principle with PERL
- ▶ Java is not a good language to write CGI in, because CGI makes one process/HTTP access and a Java Virtual Machine has a large footprint (30 Meg)
- ▶ Servlets are the solution in Java

A form to respond to

```
<FORM ACTION="/cgi-bin/test.pl" METHOD="GET">  
  Write a message: <INPUT TYPE="text"  
                    NAME="message" SIZE=20  
                    MAXLENGTH=40 VALUE="">  
  <INPUT TYPE = "submit" VALUE= "Send it!">  
  <INPUT TYPE= "reset" VALUE= "Remove it!" >  
</FORM>
```

Responding to a form in a PERL CGI

```
#!/usr/local/bin/perl
print "Content-type: text/html\\n\\n";
## CGIs must print HTTP headers AND empty line!
$REQUEST_METHOD = $ENV\{'REQUEST_METHOD'\};
$QUERY_STRING = $ENV\{'QUERY_STRING'\};
## Reading environment variables
if($REQUEST_METHOD ne "GET") \{
    print"Sorry, i can only do <code>GET</code><br />Bye!";
    exit(0); \}
($COMMAND, $MESSAGE) = split(/=/, $QUERY_STRING);
## Split the query string via PERL pattern matching.
if($COMMAND eq "message") \{
    print "<h1>You sent:</h1>";
    print "Message: $MESSAGE";
    exit(0);
\} exit(0);
```

Dynamic Web content

- ▶ Content generated by CGI is different from normal HTTP serving
- ▶ It's not a static file or image that's being served
- ▶ Instead, a dynamic content is generated
- ▶ You can use CGI to generate dynamic content even if you don't respond to a form
- ▶ Or you can use Java servlets for the same purpose