Programming web-based services

Antonio Lioy
< lioy@polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

World Wide Web (WWW)

- **abbrevited also simply "web"**
- **set of:**
  - communication protocols
  - data formats
- **based on TCP/IP channels**

Protocols for the web

- **usable many already existent protocols (e.g. FTP)**
  - limitations / complications because not designed for the web
- **defined a new application protocol:**
  - HTTP
- **the functionality that can be obtained is dictated by the application protocol (e.g. with FTP only GET and PUT of file)**

The protocol HTTP 1.0

- **HyperText Transfer Protocol**
- **RFC-1945 (HTTP/1.0)**
- **default service: TCP/80**
- **stateless protocol**
- **the client is allowed to close the connection before having received the response or part of it**
- **closing of channel by the server**
- **data on 8 bit**
- **default alphabet: ISO-8859-1 (= Latin-1)**

Stateless connection (HTTP/1.0)

The links

- **URL (Uniform Resource Locator)**

schema : // *user* : *password* @ *host* : *port* / *path* # *anchor*

- **regular schemas:**
  - http, telnet, ftp, gopher, file
- **irregular schemas:**
  - news:newsgroup
  - mailto:postal-address
- **basic definition in RFC-1738, plus 1959+2255 (LDAP), 2017 (external-body), 2192 (IMAP), 2224 (NFS)**

Evolution of link

- URL are physical addresses (using CNAME something logical can be done, but not much)
- URN (Unique Resource Name) is the future evolution, to use logical names, replication, ...
- URI (Unique Resource Identifier)

URI = URL + URN

HTTP/1.0 protocol

- ASCII commands with lines terminated by CR+LF
- the data can be binary (because the protocol is "8-bit clean")
- messages composed by header + body
- header separated from body by means of an empty line (that is containing only CR+LF)

HTTP/1.0 Methods

- **GET *uri http-version***
  requests the web page associated to the URI specified (URI is complete only in case of a request to a proxy)
- **HEAD *uri http-version***
  returns only the header of the response, not the data
- **POST *uri http-version***
  sends data to the server so that the indicated URI can process them
- the responses must start with
  http-version status-code

GET Method

request

```
GET / HTTP/1.0
```

response

```
HTTP/1.0 200 OK
Date: Wed, 20 May 1998 09:58:21 GMT
Server: Apache/1.0.0
Content-Type: text/html
Content-Length: 1534
Last-modified: Fri, 5 May 1998 12:14:23 GMT

<HTML>
. . .
</HTML>
```

POST Method

request

```
POST /cgi-bin/form.cgi HTTP/1.0
Content-type: text/plain
Content-length: 12

Antonio Lioy
```

response

```
HTTP/1.0 200 OK
Date: Mon, 8 Mar 1999 21:30:24 GMT
Server: Apache/1.2.6
Content-type: text/plain

data received from stdin: Antonio Lioy
```

Status code HTTP

- all the responses contain a status code of 3 digits **XYZ**
- the first digit (X) provides the major status of the action requested
  - X=1 : informational
  - X=2 : success
  - X=3 : redirection
  - X=4 : client error
  - X=5 : server error
- the second and the third digit refine the status

Standard HTTP/1.0 status codes

200 OK	400 Bad request
201 Created	401 Unauthorized
202 Accepted	403 Forbidden
204 No Content	404 Not found
301 Moved permanently	500 Internal server error
302 Moved temporarily	501 Not implemented
304 Not modified	502 Bad gateway
	503 Service unavailable

Redirect

- **If the object requested is not available at the indicated URI, the server can indicate the new URI by means of a response with the header**
  Location: new-URI
- **the browser can:**
  - connect automatically the client to the new URI if the requested method was GET or HEAD …
  - … while it is forbidden for it to do it if the method was POST
- **to support the old version browsers (= that do not understand Redirect) it is recommended to put also a page that provides the new URL**

Header HTTP/1.0

- **general header:**
  - Date: *http-date*
  - Pragma: no-cache
- **request header:**
  - Authorization: *credentials*
  - From: *user-agent-mailbox*
  - If-Modified-Since: *http-date*
  - Referer: *URI*
  - User-Agent: *product*

- **response header:**
  - Location: *absolute-URI*
  - Server: *product*
  - WWW-Authenticate: *challenge*
- **entity body header:**
  - Allow: *method*
  - Content-Encoding: x-gzip | x-compress
  - Content-Length: *length*
  - Content-Type: *MIME-media-type*
  - Expires: *http-date*
  - Last-Modified: *http-date*

The protocol HTTP/0.9

- **simpler**
- **not documented in any RFC**
- **request: GET *URI***
- **response: *entity-body***

The HTTP/1.1 protocol

- **RFC-2616 (RFC-2068 is obsolete)**
- **possible persistent connections**
- **the format of body is negociated**
- **cache management (in a manner specified by the protocol)**
- **security at transport level (by means of digest)**
- **introduces four new methods:**
  PUT, DELETE, TRACE, OPTIONS

Manual test of HTTP

- open a command window (aka "window DOS")
- insert the following commands:

```
C:\> telnet
Microsoft Telnet> set local_echo
Microsoft Telnet> open server_web 80
GET / HTTP/1.0 <enter>
<enter>
```

- try various methods towards diverse web servers
- suggestion: remember to activate the scrolling on the command window to view the entire response of the server (Properties - Options - Command History - Buffer Size)

Cookie

- scope: memorize information generated by the server to save state among HTTP transactions locally on the client where the browser "runs"
- initially defined by Netscape
- became afterwards standard IETF as RFC-2695 "HTTP state management mechanism")
- RFC-2964 "Use of HTTP state management" contains various warnings on problems and harmful effects caused by the cookies (e.g. disclose web user information to third parties)

Characteristics of cookie

- data of small dimension stored on client (persistent information)
- the cookies have the format ***name=value***
  - browser save cookies in a file
- information sent to the server during subsequent accesses to server's resources
- used to create a connection among different HTTP requests towards the same server
  - allow to maintain infomation among subsequent requests of a client towards the same server
    - mechanism to preserve the status of the session (HTTP/1.0 is stateless)

Support of cookies in browsers

- Netscape Navigator
  - from version 2.0
  - stored in the file cookies.txt in (default in Netscape\Users\username)
- Microsoft IE
  - from version 3.0
  - stored in \Windows\Cookies or in the user profile

Limits of cookies

- limitations with respect to the number of cookies that can be stored
- on a client:
  - max 300 cookies
  - max 4 KB for a cookie
  - total = 1.2 MB max
- on a server:
  - max 20 cookies / client
  - max 4 KB for a cookie
  - total = 80 KB max / client

Syntax of cookies

- cookie inserted in the header of HTTP response
- syntax of cookie:

```
Set-Cookie:
  cookieName=cookieValue
  [ ; EXPIRES=dateValue ]
  [ ; DOMAIN=domainName ]
  [ ; PATH=pathName ]
  [ ; SECURE ]
```

© A.Lioy - Politecnico di Torino (2003-4)

Cookie: value, expiry period, secure

- **the name of the cookie and its relative value are any strings:**
  - with no comma, semicolon and blank space
  - these characters must be expressed in hexadecimal (%2C, %3B, %20)
- **EXPIRES: expiry data after which the client can cancel the cookie**
  - format Wdy, DD Mon YY HH:MM:SS GMT
  - if the expiry period is not specified, the cookie expires when the browser is closed
- **SECURE: cookie transmitted only on HTTPS channel**

Cookie: domain, path

- **DOMAIN indicates the domain authorised to manage the cookie (for example polito.it)**
  - only the requests sent to this domain cause the transmission of the cookie
  - if the domain is not specified, the browser uses the name of the server that sent the cookie
- **PATH indicates the path for which it is authorised the use of the cookie (for example /didattica/)**
  - only the requests to a URL in this path will cause the transmission of a cookie
  - if the path is not specified, the browser uses the path of the URL that caused the sending of the cookie

Transmission of a cookie to server

- **when the browser is about to access an URL, it checks if there exist cookies associated to the domain and to the path**
- **in affirmative case it includes in the header HTTP of the request all the couples name/value of the relative cookies**
- **if the URL specified in the HTTP request is that of a CGI, it will find this string in the environment variable HTTP_COOKIE**

```
Cookie: name1=value1; name2=value2; ...
```

Example

- **cookie used to manage the list of books selected by a user that accesses the web site of a virtual shop**
- **the client accesses the server and receives:**

```
Set-Cookie:
  customer=john_smith;
  expires=Saturday, 28-Aug-99 00:00:00 GMT;
  path=/cgi/bin/;
  domain=books.virtualShopping.com;
  secure
```

Example (2)

- **when the client accesses an URL in the path /cgi/bin it sends to the server a header that contains:**
  - Cookie: customer=john_smith
- **the client receives as response:**
  - Set-Cookie: part_number=book1; path=/cgi/bin ; ...
- **the client requests an URL in the path /cgi/bin and it sends:**
  - Cookie: customer=john_smith;part_number=book1

Example (3)

- **the client receives:**
  - Set-Cookie: shipping=fedex; path=/cgi/bin/foo; ...
- **the client accesses an URL in the path /cgi/bin/foo and it sends:**

  Cookie: customer=john_smith; part_number=book1; shipping=fedex

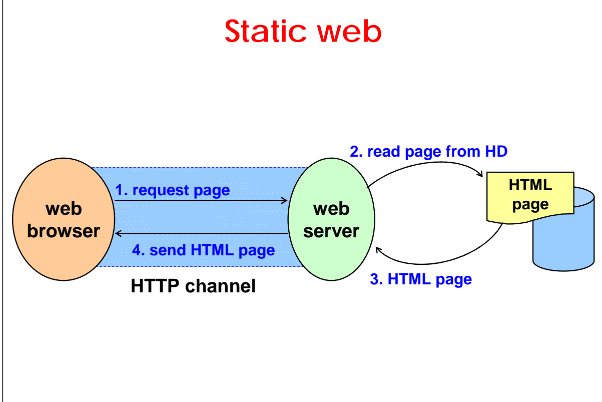
Problems of cookies

- **the mechanism based on cookies permits to create profiles of users**
- **user tracking = term used to indicate the possibility to trace the web sites a user visits and consequently to know his habits and interests**
- **example: if a user downloads an advertising banner from a web site, besides receiving an immage it receives also a cookie**
- **for all web sites belonging to a circuit it is possible to set and recover the values of the cookies regarding the web navigation preferences of a user**
- **the cookies can be disabled in the browser**

Disadvantages

- **authentication by means of cookies (e.g. commercial sites, associates the user to the shopping cart)**
- **attacks that permit to intercept the cookies:**
  - packet sniffing
  - web spoofing

Static web



Static web

- **web page does not change ever its content**
- **… until the author does not change it explicitly**
- **the content of a page:**
  - does not depend on the interaction with the user
  - does not depend on the information sent by the client to the server
  - does not depend on the instant of time when the page is requested
- **page implemented in HTML / CSS**

The static web: pro and contro

- **to each web page corresponds an HTML file**
- **(+) maximum efficiency (low load on CPU)**
- **(+) possibility to cache the pages:**
  - in RAM or on disk
  - on server
  - sul client
  - sui proxy
- **(+) search engines can index the pages**
- **(-) the property of data to be static**
- **(-) no adaptability to the clients and to their capabilities**

Static web with dynamic pages

- **the client processes the dynamic content of the page (script or applet Java)**

web browser — 1. request page — web server — 2. read page DHTML — DHTML page — 3. page DHTML — 4. send DHTML page — HTTP channel

Static web with dynamic pages

- **the page varies its content depending on the interaction with the user**
  - for example a content menù that appears when the mouse pass over a particular area
- **in general it is used the term DHTML:**
  - HTML 4.0 or further versions
  - CSS (Cascaded Style Sheet)
  - client side script language
    - ECMA 262 (EcmaScript Edition 3) implemented as JavaScript or JScript
    - VBScript (only for IE 4.0 or further versions)

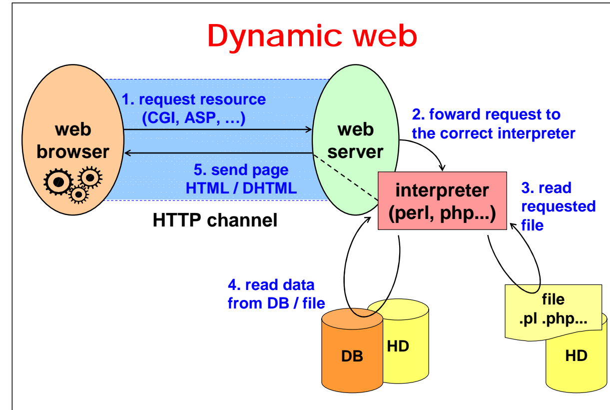
Dynamic pages: pro and contro

- **presentation of the content that varies**
- **(+) efficient (low load of CPU on server)**
- **(-) inefficient (load of CPU on client, especially for the applets)**
- **(~) possibility to cache the pages**
- **(~) search engines can index pages (but only static data …)**
- **(-) the property of data to be static**
- **(-) functionality depends on the capability of the client (scripting languages, types of applets, CPU, RAM, …)**

Dynamic pages: applet

- **two types of applets:**
  - applet Java (requires a JVM on browser)
  - ActiveX control (requires IE + Windows)
- **problems:**
  - compatibility (which version of language or of JVM?)
  - load (require execution of code)
  - security (execution of a real program):
    - applet Java runs in a "sandbox"
    - activeX installs a DLL (!)

Dynamic web



Dynamic web

- **page generated by the server dynamically**
- **varies its data content in base of:**
  - requests sent by the user
  - content of a database
  - istant of time when the request is made
- **techniques to implement dynamic web:**
  - CGI
  - server side scripting languages (JavaScript/Jscript, VBScript, PHP, PerlScript, Python, ...)
  - SSI (Server Side Include)
  - Servlet, JSP (Java Server Pages)

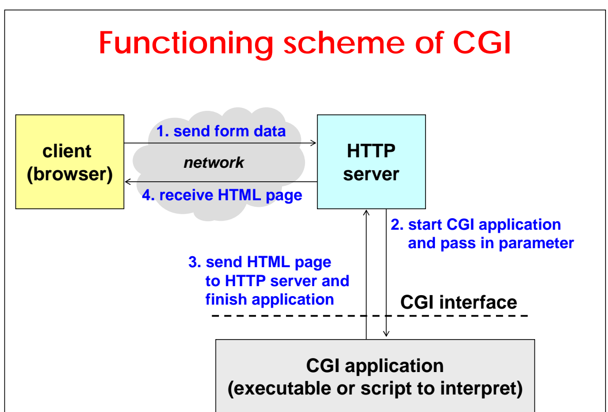
Dynamic web: pro and contro

- **adaptation of pages to variable conditions:**
  - input provided by the client
  - capability of the client
- **(+) maximum dynamism of data**
- **(+) very good adaptation to clients and to their capabilities**
- **(-) low efficiency (high load of CPU)**
- **(-) impossible to cache pages if the selection parameters are not in the URL (e.g. are in cookie)**
  - possible errors if the caching is activated
- **(-) search engines cannot index pages**

CGI

- **Common Gateway Interface**
- **http://hoohoo.ncsa.uiuc.edu/cgi/interface.html**
- **the web server:**
  - starts an application
  - pass in any available parameters:
    - by means of stdin (methods POST, PUT)
    - by means of a modified URL (method GET)
  - receives the results by means of stdout
  - the results must be in web format (HTML/CSS/scripting client-side)

Functioning scheme of CGI

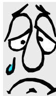


CGI: pro

- **general method**
- **available on all web servers (IIS, Apache, …)**
- **application implemented as**
  - executable file
  - script to be interpreted

CGI: contro

- **each access requires the activation of a process:**
  - high cost of initialization
  - high latency
  - creation / destruction of many processes
- **load of memory proportional to the number of processes simultaneously active**
- **difficulty of communication among the web server and the application (different memory spaces)**



CGI: contro (II)

- **lack the mechanisms for sharing the resources among CGI programs**
  - each access to a resource requires “opening” and “closing” of the resource
  - does not exist the concepts of session or of transaction
- **the graphical interface of the web application (that is the HTML tags) is included in the code**
- **paradigm unsuitable for applications with many users, simultaneously active and that require lo[w] response times**

CGI: possible improvements

- **use of environment variables to communicate among the web server and the application**
- **inclusion of one or more interpreters in the web server:**
  - (+) better speed to start up programs
  - (+) better communication with the application
  - (+) reduced memory load
  - (-) bigger dimension of server
- **pre-activation of application (in N copies) and inclusion in the server of a specific module to choose a free copy and communicate with it (exist an implementation of this idea?)**

Passing of input parameters to a CGI

- **three techniques to transmit form data:**
  - standard input (when POST or PUT are used)
  - modified URL (when GET is used):
    - initial URL of CGI followed by '?' and the list of parameters separated by '&'
    - the environment variable QUERY_STRING contains part of the URL after '?'
  - command line (when ISINDEX is used)
- **other information passed in to the application by means of environment variables**

General CGI environment variables

- **SERVER_SOFTWARE**
  - name and version of the server answering the request (and running the gateway)
  - format: name/version
- **SERVER_NAME**
  - DNS name, alias of the server or IP address
  - as it would apper in a self-referencing URL
- **GATEWAY_INTERFACE**
  - specific version of the CGI to which this server complies
  - format: CGI/revision (e.g. CGI/1.1)

Request-specific CGI variables (I)

- **SERVER_PROTOCOL**
  - name and revision of the application protocol the request came in with
  - format: protocol/revision (e.g. HTTP/1.0)
- **SERVER_PORT**
  - port number of the server to which the request was sent
- **REQUEST_METHOD**
  - the method with which the request was made
  - for HTTP: "GET", "HEAD", "POST", …

Request-specific CGI variables (II)

- **PATH_INFO**
  - if at the end of the URI of the CGI there still exists extra path information (e.g. part of path), the client stores it in this variable
- **PATH_TRANSLATED**
  - part of the path after the URI of CGI viewed as the CGI would be the root of the server, that is a sort of virtual-to-physical mapping (but it does not necessarily exist)
- **example, for the CGI http://prova.cgi.it/cgi/x:**
  - if started with URI = http://prova.cgi.it/cgi/x/aaa.txt
  - PATH_INFO = /aaa.txt
  - PATH_TRANSLATED = http://prova.cgi.it/aaa.txt

Request-specific CGI variables (III)

- **SCRIPT_NAME**
  - virtual path and name of the application being executed, used for the self-referencing URLs
- **QUERY_STRING**
  - information which follows the '?' in the URL which referenced this application

Request-specific CGI variables (IV)

- **REMOTE_HOST**
  - the hostname making the request
  - if the server does not have this information, it should set only REMOTE_ADDR
- **REMOTE_ADDR**
  - IP address of the host making the request

Request-specific CGI variables (V)

- **AUTH_TYPE (*)**
  - authentication method used to validate the user
- **REMOTE_USER (*)**
  - username with which the user has authenticated as
- **(*) available only if:**
  - the server supports client authentication
  - the URL is protected
- **REMOTE_IDENT**
  - if the server requires identification with IDENT, then this variable contains the remote username
  - use of this variable should be limitted to logging only

IDENT

- **IDENTification protocol (RFC-1413)**
- **the server that receives a TCP connection from a client:**
  - connects to the port tcp/113 of the client
  - sends the string “client-port : server-port”
  - receives a response
    - USERID : operating-system : username
    - ERROR : …
- **remark:**
  - on the client must be started an IDENT server
  - protocol rarely used in practice

Request-specific CGI variables (VI)

- **CONTENT_TYPE**
  - MIME type of data passed in on standard input (cases PUT and POST)
- **CONTENT_LENGTH**
  - length in bytes of the above-mentioned content, as given by the client

CGI var. contained in HTTP header

- **HTTP header lines the client sends are placed in the environment as variables**
  - with the prefix HTTP_ followed by the header name
  - characters ‘-’ in the header are changed to ‘_’
- **the server may exclude any headers which it has already processed (e.g. Authorization, Content-type, Content-length)**
- **the server may exclude one or more headers, if including them would exceed any system environment limits**

CGI var. relative to HTTP header

Two examples:

- **HTTP_ACCEPT**
  - MIME types which the client will accept
  - the application understands what it can generate as response
  - each item in the list is separated by comma
  - format: type/subtype, type/subtype
- **HTTP_USER_AGENT**
  - the browser the client is using to send the request
  - format: software/version library/version

Reading environment variables in C

- **getenv**
- **receives in input the name of the environment variable as string**
- **returns the pointer to the string containing the value**
- **... or NULL if the variable is not defined**

```
#include <stdlib.h>
char *getenv (const char *varname);
```

Example: read env. var. in C

Write a program that:

- **takes as input on the command line the name of an environment variable**
- **returns its value**
- **signals error in case the variable does not exist**

`printenv.c`

List of environment variables in C

- **parameter "envp" passed in to the function main**
  - supported by MS-VC++ and gcc … but it is not ANSI
- **envp is an array of pointers to strings**
  - each string contains the pair:
    - NOME_VARIABILE=VALORE_VARIABILE
  - the last element of the array has the value NULL
    - required, given that there is no parameter that reports the number of strings

```
int main (int argc, char *argv[], char *envp[]);
```

Example: list env. var. in C

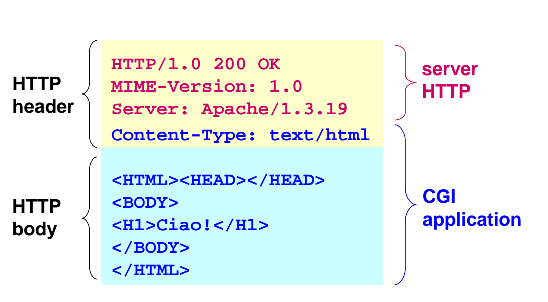
Write a program that prints on standard output all the environment variables defined and their respective values.

`prallenv.c`

Output generated by the CGI

- **the application must return a valid HTML page**
  - use à " < > …
- **the application must return also part of the HTTP headers; CGI/1.1 specifies the next headers:**
  - Content-Type:
    - MIME type of the response
  - Location:
    - if it is an URI, server sends a redirect to the client
    - if it is a document, server sends it to the client
  - Status:
    - the server uses it as status code in its header

CGI: generation in the response



Example: visual. CGI env. var. in C

Write a CGI application in C that:

- **prints on the standard output all the environment variables defined together with the relative values**
- **format the output as requested by the CGI interface**
  - write part of the header HTTP on the standard output
  - write the data in HTML format on the standard output

To use the program:

- **copy the executable file in the appropriate directory of the web server (CGI-enabled)**
- **copy the program from the browser, by selecting the appropriate URL (htmlenv, htmlenv.cgi o htmlenv.exe)**

`cgiecho.c`

Use of CGI together with form

Basic structure of form

- NAME: symbolic name used to refer the form
- ACTION: URL relative to a CGI, PHP, ASP script or to any type of processing on the server
- METHOD: GET or POST

```
<FORM NAME=F1 METHOD=GET ACTION=http://... >
  <INPUT  ...>
  <SELECT ...>
  ...
  <INPUT TYPE=SUBMIT ...>
  <INPUT TYPE=RESET ...>
</FORM>
```

Input elements

- tag INPUT
- TYPE: text, password, checkbox, radio, image, file, hidden, submit, reset
- NAME: symbolic name
  - to pass in the data to the server (CGI interface)
  - to get access to the element from JavaScript or VBScript source code
- VALUE: initial content of the field

```
<INPUT TYPE=... NAME=... VALUE=...>
```

Selection with single choice

- tag SELECT to group the various options
- tag OPTION for the single options, if necessary with a default choice (SELECTED)

```
<SELECT NAME=...>
  <OPTION> ...
  <OPTION> ...
  <OPTION SELECTED> ...
</SELECT>
```

Example of form (file upload)

```
<FORM
  action=/cgi-bin/acquisisci
  ENCTYPE="multipart/form-data"
  METHOD=POST>

  <INPUT TYPE=FILE NAME=upload>

  <BR><BR>

  <INPUT TYPE=SUBMIT VALUE=Submit>

</FORM>
```

Use of script to validate a form

```
<FORM
  NAME=sample
  METHOD=POST
  ACTION=...
  onSubmit="return validateForm()">
Name:
  <INPUT TYPE=TEXT NAME=name SIZE=30><BR>
Age:
  <INPUT TYPE=TEXT NAME=age SIZE=3><BR>
Date of birth:
  <INPUT TYPE=TEXT NAME=birthdate SIZE=10><BR>
  <INPUT TYPE=SUBMIT>
  <INPUT TYPE=RESET>
</FORM>
```

Validation script

```
<SCRIPT LANGUAGE="JavaScript">
function validateForm()
{
  formObj = document.sample;
  if (formObj.nome.value == "") {
    alert("You have not inserted the name!");
    return false;
  }
  else if (formObj.eta.value == "") {
    alert("You have not inserted the age!");
    return false;
  } else if ...  return false;
}
</SCRIPT>
```

Pass of form parameters - CGI

- independently of the method used, the string containing the parameters of the query is the form:

  nome_par1=val_par1&nome_par2=val_par2&…

- separator of parameters: &
- spaces replaced by: +
- special characters, non US-ASCII or with a particular meaning (/ ? …) replaced with %xx (where 'xx' is the hexadecimal number that represents the code)

Example: sending of a form

```
<FORM
  NAME=sample
  METHOD=GET
  ACTION=/cgi-bin/acquisisci>
First name and last name:
  <INPUT TYPE=TEXT NAME=lastname SIZE=30><BR>
Number of children:
  <INPUT TYPE=TEXT NAME=children SIZE=3><BR>
Date of birth:
  <INPUT TYPE=TEXT NAME=birth SIZE=10><BR>
  <INPUT TYPE=SUBMIT>
  <INPUT TYPE=RESET>
</FORM>
```

Example: sending of a form (2)

- If the above form would have been filled in by Mr. Marco Noè, born in 30/10/74, father of 3 children …
- then it would be sent the next string:

```
cognome=Marco+No%E8&figli=3&nascita=30%2F10%2F74
```

Attention at empty fields !

- excepting the tag SELECT …
- … all the other fields of a form are allowed not to transmit an input
- … and in one case (TYPE=CHECK) can also be absent the variable regarding the field (if it is OFF)
- the CGI applications must know how to treat all the cases

Example CGI GET: the form

```
<FORM METHOD="GET" ACTION="/cgi/insaula">
<TABLE BORDER="0">
<TR>
  <TD>Classroom Number:</TD>
  <TD><INPUT TYPE="text" SIZE="8" NAME="num"></TD>
</TR>
<TR>
  <TD>Location:</TD>
  <TD><INPUT TYPE="text" SIZE="15" NAME="location"></TD>
</TR>
<TR>
  <TD>
    <INPUT TYPE="submit" VALUE="Send">
    <INPUT TYPE="reset" VALUE="Cancel">
  </TD>
</TR>
</TABLE>
</FORM>
```

Example CGI GET: URI, HTTP, env

URI
```
http://localhost/cgi/insaula?num=12A&sede=Sede+Centrale
```

header HTTP
```
GET /cgi/insaula?num=12A&sede=Sede+Centrale HTTP/1.1
Accept: image/gif, image/x-xbitmap,image/jpeg,*/*
Referrer: http://localhost/pad/formaula.html
Accept-Language: it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)
Host: 192.168.235.10
Connection: Keep-Alive
```

QUERY_STRING
```
num=12A&sede=Sede+Centrale
```

Example CGI POST: the form

```
<FORM METHOD="POST" ACTION="/cgi/insaula">
<TABLE BORDER="0">
<TR>
  <TD>Classroom Number:</TD>
  <TD><INPUT TYPE="text" SIZE="8" NAME="num"></TD>
</TR>
<TR>
  <TD>Location:</TD>
  <TD><INPUT TYPE="text" SIZE="15" NAME="txtSede"></TD>
</TR>
<TR>
  <TD>
  <INPUT TYPE="submit" VALUE="Send">
  <INPUT TYPE="reset" VALUE="Cancel">
  </TD>
</TR>
</TABLE>
</FORM>
```

Example CGI POST: URI, HTTP, env

URI
```
http://localhost/cgi/insaula
```

header HTTP
```
POST /cgi/insaula HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Referrer: http://localhost/pad/formaula.html
Accept-Language: it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;MSIE 6.0;Windows NT 5.0)
Host: 192.168.235.10
Connection: Keep-Alive
Content-Length: 26

num=12A&sede=Sede+Centrale
```

QUERY_STRING

Form: better GET or POST?

- **GET:**
  - allows to cache the response page
  - allows to bookmark the page
  - leave trace of the parameter values in the server log (problem of privacy and/or security)
  - some servers limit the length of the query string to 256 characters if it is contained in the URI
- **POST:**
  - does not allow caching and bookmarking
  - does not leave trace in log
  - does not impose limits to the query string

cgic

- **ANSI C library to program CGI**
- **http://www.boutell.com/cgic/**
- **extracts form data, correcting browser errors**
- **supports transparently GET and POST**
- **reads data from the form or an uploaded file**
- **functions to set and read the cookies**
- **supports correctly CR and LF in text forms**
- **extracts form data (string, int, real, single / multiple choices), controlling the limits of numeric fields**
- **loads the CGI env. var. in non null strings**
- **compatible with any CGI server (U*ix, Win*)**

Libwww

- **C library to develop client HTTP+HTML**
- **used also to implement robot**
- **http://www.w3.org/Library/**

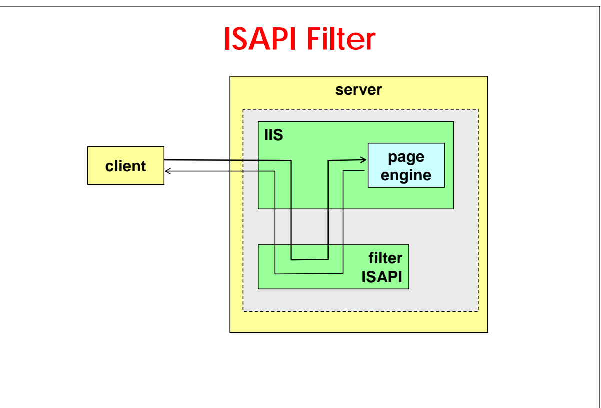
ISAPI

- **Internet Server API**
- **MS proprietary mechanism to create dynamic pages with IIS:**
  - each ISAPI application is a DLL
  - … loaded in memory at the first request
  - … remains in memory to respond to other requests
  - in the same memory space of IIS (bidirectional communication by means of specific objects shared between IIS and the ISAPI application)
  - can be removed from memory only by the system administrator
- **the ISAPI application must be thread-safe**

ISAPI Filter

- **a custom DLL**
- **runs in the same memory space of the IIS web server**
- **can make pre-processing of the request**
- **can make post-processing of the response**
- **examples of possible features:**
  - forwarding of the request to balance the load among various servers
  - additional security / log features
  - formatting of the response depending on the client capability

ISAPI Filter



Configuration of ISAPI filters

- **in base of extension of the URL**
- **use MMC to manage a virtual directory in IIS**
- **in the configuration of applications associate:**
  - extensions
  - applications
  - commands HTTP that are accepted
- **possibile also to associate specific web pages for various application errors**

ASP

- **Active Server Pages**
- **is an ISAPI filter (about 300 KB) associated by default to the extensions .ASP**
- **allows to insert in an HTML page :**
  - server-side scripts in various interpreted languages (default: Vbscript; possible also JScript)
  - IIS variables
  - interaction with built-in ASP objects

ASP Objects

- **Application**
  - global variables of the application, for all users
  - events Application_OnStart and Application_OnEnd
- **ASPError**
  - description (read-only) of possible errors
- **ObjectContext**
  - management of transactions (internal to the script)
  - methods SetAbort and SetComplete
  - events OnTransactionAbort and OnTransactionCommit

ASP Objects

- **Request**
  - request parameters from the client (form data, parameters of the URL)
  - cookie and client certificates
  - collection ServerVariables contains fields of the header HTTP of the request, environment variables
- **Response**
  - control of the HTTP response
  - sending of cookie, redirection to another URL, buffering

ASP Objects

- **Server**
  - management of server parameters (execution timeout of the scripts, encoding of pages, mapping from virtual directory to real directory)
  - creation of objects with the method CreateObject (e.g. ADO)
- **Session**
  - session management with help of cookies
  - session parameters (code page, location, timeout)

Client-side scripting

- **HTML is a description language of pages**
- **the only possible interactivity is to follow the links**
- **add-on of interactivity to the HTML page by means of code to be interpreted on the client side (in the browser):**
  - NS and SUN invent the language LiveWire, further called JavaScript (but it is not a subset of Java!)
  - MS invents VBScript (subset of VBA), then JScript
  - agreement to merge JavaScript e JScript in ECMAScript:
    - standard ECMA-262
    - globally referred as JavaScript (version 1.3)

Client-side scripting: what is useful for ?

- **insert elements in the HTML page dynamically**
- **a function implemented with JavaScript/VBscript can be associated to an event resulting from the interaction with the page**
  - e.g. the click on a figure
  - e.g. on the sending of a form
- **execute code following an event**
  - validation of data introduced in a form before sending them to the server
    - unnecessary traffic on the network is spared and the server-side application logic is simplified

Insertion of client-side code

- **use of SCRIPT tag with parameters:**
  - LANGUAGE=… (JavaScript, JavaScript1.3, VBScript, …)
  - SRC=file (if the script is contained in an external file)
- **use of comments to hide the code from browsers that cannot process/support it**

```
<script src="file" language = "linguaggio">
<!--
… codice client-side …
script_comment -->
</script>
```

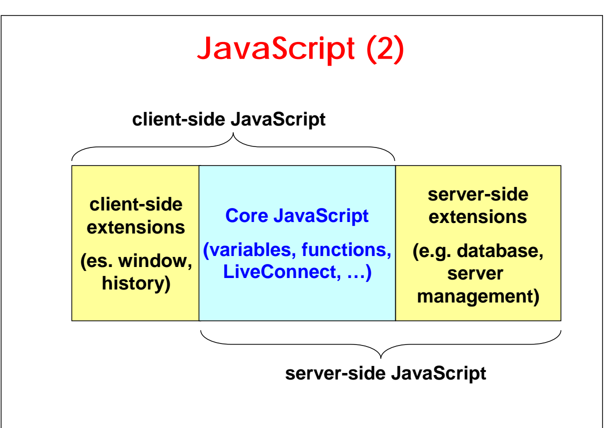
Comments & JavaScript

- **JavaScript supports comments in C and C++ style**
  - ignored text between: // and newline, /* and */
- **JS supports the HTML comment <!– as single-line comment**
- **JS does not support the closing sequence -->**
  - necessary to permit the JS comment: // -->
  - a browser that does not support JS ignores everything among the first line of the JS program <!– and the last line of the program // -->
  - a browser that supports JS ignores only the first line (<!--) and the last line (// -->) but not the lines in between

JavaScript

- **is an interpreted language**
- **contains a restricted set of commands that serve to the client side applications to:**
  - process data inserted in the FORM included in the HTML page
  - send comands to the browser (e.g. open/close windows)
  - execute certain operations following an event caused by a certain user action (event handling)

JavaScript (2)



Versions

- **connection among versions of JavaScript and browser versions:**
  - JavaScript 1.0 = Navigator 2.0 / IE 3.0
  - JavaScript 1.1 – Navigator 3.0
  - JavaScript 1.2 – Navigator 4.0-4.05 / IE 4
  - JavaScript 1.3 – Navigator 4.06-4.8 / IE 5

Pay attention at version number

- **NS 2.0**
  - executes code in <SCRIPT LANGUAGE="JavaScript">
  - ignores code in <SCRIPT LANGUAGE="JavaScript1.1"> and <SCRIPT LANGUAGE="JavaScript1.2">
- **NS 3.0:**
  - executes code in <SCRIPT LANGUAGE="JavaScript"> and <SCRIPT LANGUAGE="JavaScript1.1">
  - ignores code in <SCRIPT LANGUAGE="JavaScript1.2">

Pay attention at version number

- **NS 4.0**
  - executes code in <SCRIPT LANGUAGE="JavaScript">, <SCRIPT LANGUAGE="JavaScript1.1">, and <SCRIPT LANGUAGE="JavaScript1.2">
- **when the SRC attribute is used, the LANGUAGE attribute is ignored by Netscape Navigator**

JavaScript and HTML pages

- **transmission of JavaScript applications to the browser:**
  - insert the JavaScript code in the html page using the tag SCRIPT
  - write the code in a script file (with the extension .js)
  - specify a JavaScript expression as value of an HTML attribute
  - as event handlers in certain HTML tags
  - further details: JavaScript Guide http://developer.netscape.com/docs/manuals/communicator/jsguide4/getstart.htm#1009491

JavaScript: a first example

```
<html>
<head></head>
<body>
  <script language="JavaScript">
    document.writeln("Ciao!")
  </script>
</body>
</html>
```

js1.html

JavaScript: Square table

```
<HTML>
    <HEAD>
    <title>Square table</title>
    </HEAD>
    <BODY>
    <h1>Square table</h1>
    <p>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
    var i;
    for (i=1; i<20; i++) {
        document.writeln(i+ "^2 = " + i*i + "<br>");
    }
    // -->
    </SCRIPT>
    </BODY>
</HTML>
```

Event handler

- **it is possible to associate JavaScript commands to events with an "event handler"**
- **syntax:**

  <TAG . . . eventHandler = "JavaScript_code">
- **where:**
  - "TAG" is a generic HTML tag
  - "eventHandler" is the name of the event handler (e.g. onClick, onFocus, onSubmit, onChange, onLoad, onUnload,)
  - "JavaScript Code" is a sequence of commands JavaScript

JS: a second example

```
<html><head>
<script language="JavaScript">
function setColor(){
headerObj = document.getElementById("headerID");
headerObj.style.color="red";
}
</script></head><body>
<h1 id="headerID" onclick="setColor()">
click here!
</h1>
</body></html>
```

js2.html

JS: a third example

- **when the same script serves for several pages, it can be written in an external file and call it in the HTML page**
- **the file ".js" must**
  - be a text file
  - have a name of maximum 8 characters long
  - not to contain the tag <script>

JS: a third example (2)

```
<html>
<head>
<script src="js3.js" language="JavaScript">
</script>
</head>
<body>
<h1 id="headerID" onclick="setColor()">
click here!</h1>
</body>
</html>
```

js3.html

```
function setColor() {
headerObj = document.getElementById("headerID");
headerObj.style.color="blue" }
```

js3.js

JS: considerations

- **the semicolon is mandatory only if more than one instruction is on the same line**
- **comments: as for C++**
- **old browsers:**

```
...
<script language="JavaScript">
<!--
    ...
//-->
</script>
```

Server-side scripting

- **different technologies, but are characterized by having scripting code mixed with template HTML + client-side scripting in the page file**
- **ASP (Microsoft)**
  - VBscript
  - JScript
  - implementation also for Apache (with PerlScript)
- **PHP (open source)**
  - developed for Apache
  - exists also for IIS
  - can be used either as general scripting language or as CGI

Server-side scripting (2)

- **JSP (Sun), hybrid technology**
  - the code is included in the HTML template (as for other server-side scripting technologies)
  - the code is composed of
    - scripting elements (as other server-side languages)
    - directives
    - actions (like proprietary XML & NS tags)
  - the pages are translated in servlet by the server Web

SSI (Server Side Include)

- **introduces directives in the HTML code as comments**

```
<!--#command tag1=value1 tag2=value2 ... -->
```

  - if SSI ***is not*** supported by the web server, the directives are ignored
  - if SSI is supported, the directives are replaced with the text resulted from the processing in the HTML page returned to the client
- **adds new environment variables**
- **does not replace CGI, but introduces the possibility to add dynamic features to the HTML pages by making simple operations**

SSI (2)

- **the pages that contain SSI directives must have ".shtm" or ".shtml" extensions**
- **it is possible to configure appropriately the web servers so that to process the SSI directives also for the pages with ".htm" or ".html" extensions**
- **server web**
  - Apache supports SSI (and XSSI from version 1.2)
  - IIS supports only SSI
    - in ASP pages the SSI directives must be inserted in the part of HTML template; cannot be generated by ASP code

SSI environment variables

- **DOCUMENT_NAME: the name of the current file**
- **DOCUMENT_URI: the virtual path to this document (as /docs/tutorials/foo.shtml)**
- **QUERY_STRING_UNESCAPED: version of the query string the client is sending, with all the special characters of the shell, preceded by '\'**
- **DATE_LOCAL: current date, local time zone; depends on the param. `timefmt` in the command `config`**
- **DATE_GMT: analogous to DATE_LOCAL, but relative to the Greenwich meridian**
- **LAST_MODIFIED: the last modification date of the current document; as above it depends on `timefmt`**

SSI directives

- **#config: allows to set some parameters**
  - errmsg: message to return in case of error encountered at parsing of the SSI directives
  - timefmt: the format of date and hour; definition string similar to the Unix system function strftime( )
  - sizefmt: the format of the file dimension
    - bytes: expressed in bytes
    - abbrev: abbreviated format (KB or MB)

```
<!--#config errmsg="ERROR_MSG" -->
<!--#config timefmt="FORM_STRING" -->
<!--#config sizefmt="bytes" -->
```

SSI directives (2)

- **#echo: returns the content of the environment variable (tag: var) passed in as parameter**

```
<!--#echo var="NAME_VARIABLE_ENV" -->
```

- **#exec: executes a shell command or a CGI script whose name is passed in as parameter and returns the relative output, tag that are supported:**
  - cmd: shell command (Unix: /bin/sh, Win32: cmd.exe) indicated by the string
  - cgi: CGI script indicated by the string (virtual path); no proc. unless the convers. from URL to tag <A>

```
<!--#exec cmd="PATH_SHELL_SCRIPT" -->
<!--#exec cgi="VIRT_PATH_CGI_SCRIPT" -->
```

SSI directives (3)

- **#flastmod: returns date and hour of the last modification of a file (tag: file) whose name is passed in as parameter**

```
<!--#flastmod file="NAME_FILE" -->
```

- **#fsize: returns the dimension of a file whose name is passed in as parameter; the format can be modified by sizefmt; supported tags:**
  - virtual: virtual path (no access to CGI script)
  - file: relative physical path starting from the current directory (no absolute paths, no use of '../')

```
<!--#fsize virtual="VIRT_PATH_NAME_FIL" -->
<!--#fsize file="REL_PATH_NAME_FILE" -->
```

SSI directives (4)

- **#include: inserts the content of a file in the page returned to the client; the file name is passed in as parameter; supported tags:**
  - virtual: virtual path (no access to CGI script)
  - file: relative physical path starting from the current directory (no absolute paths, no use of '../')

```
<!--#include virtual="VIRT_PATH_NAM_FIL" -->
<!--#include file="REL_PATH_NAME_FILE" -->
```

SSI examples

- **returns the local date and hour**

```
<!--#echo var="DATE_LOCAL" -->
```

- **returns the local date and hour formatted in a different way**

```
<!--#config timefmt="%A %B %d, %Y" -->
<!--#echo var="DATE_LOCAL" -->
```

- **executes a hypothetical script that returns the number of accesses to the page**

```
<!--#exec virtual="/cgi-bin/counter" -->
```

SSI examples

- **inserts local date and hour in standard form:**

```
<!--#echo var="DATE_LOCAL" -->
```

- **inserts local date and hour in non standard form:**

```
<!--#config timefmt="%A %B %d, %Y" -->
<!--#echo var="DATE_LOCAL" -->
```

- **executes a system command (the text <DIR> contained in the output of the command `dir` can cause an erroneous formatting by the browser)**

```
<!--#exec cmd="ls" -->
<!--#exec cmd="dir" -->
```

SSI examples (2)

- **inserts a page footnote common to other pages**

```
<!--#include file="footer.html" -->
```

- **inserts the date of the last modification to the current page; solution 1 (if the page name is changed, then it must modify the directive)**

```
<!--#config timefmt="%A %B %d, %Y" -->
<!--#flastmod file="tesine.html" -->
```

- **inserts the date of the last modification to the current page; solution 2 (the directive can be included as it is in all the pages)**

```
<!--#config timefmt="%D" -->
<!--#echo var="LAST_MODIFIED" -->
```

SSI examples (3)

- **set an error message different from the standard one in case of problems encountered at the parsing of the directives**

```
<!--#config errmsg="[New error message!]" -->
```

  - standard error message; the code of the directive is replaced with next text

```
[an error occurred while processing this directive]
```

  - error message set with the directive

```
[New error message!]
```

XSSI (eXtended Server Side Include)

- **introduced in the version 1.2 of Apache**
- **adds**
  - the possibility to set the values of variables
  - a conditional construct of type if-then-else
- **modifies**
  - the directive #include that includes now the functionalities of #exec
- **permits to carry out more complex tasks (for example to recognize the browser type)**

XSSI directives

- **#include: inserts the content of a file in the page returned to the client and also the result of a script; the file name is passed in as parameter; supported tags:**
  - virtual: virtual path (no access to CGI script)
  - file: relative physical path starting from the current directory (no absolute paths, no use of '../')

```
<!--#include virtual="VIRT_PATH_NAM_FI" -->
<!--#include file="REL_PATH_NAME_FILE" -->
```

XSSI directives (2)

- **`#set`: defines a variable and sets its value the file name is passed in as parameter (tag `var`)**

```
<!--#set var="VAR_name" value="String or a #" -->
```

- **`#if, #elif, #else, #endif`: control instructions of the conditional flow; can contain HTML code or other SSI/XSSI directives**

```
<!--#if expr="EXPRESS_CONDIT" -->
<!--#elif expr="EXPRESS_CONDIT" -->
<!--#else -->
<!--#endif -->
```

XSSI examples

- **executes a hypothetical script that returns the number of accesses to the page**

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

- **recognition of the type of browser**

```
<!--#if expr="${HTTP_USER_AGENT} = /MSIE/" -->
  <B>You are using some kind of MSIE
Browser</B><BR>
 <!--#else -->
  <B>You are not using an MSIE Browser</B><BR>
<!--#endif -->
```

XSSI examples (2)

- **recognition of the browser type and setting of the CSS stylesheet**

```
<!--#set var="VAR_css" value="msie" -->
   <!--#if expr="($HTTP_USER_AGENT=/Mozilla/)
              && ($HTTP_USER_AGENT !=/compatible/)" -->
   <!--#set var="VAR_css" value="nav" -->
   <!--#elif expr="($HTTP_USER_AGENT=/Opera/)" -->
   <!--#set var="VAR_css" value="opera" -->
   <!--#endif -->
   <LINK REL="stylesheet" type="text/css"
    href="/css/<!--#echo var='VAR_css' -->.css">
   <!--#if expr="($HTTP_USER_AGENT=/Mac/)" -->
   <LINK REL="stylesheet" type="text/css"
    href="/css/mac.css">
   <!--#endif -->
```

SSI/XSSI in action

page before processing (as memorized on server)

```
<HTML><HEAD><TITLE>
<!--#include virtual="title.inc" -->
</TITLE></HEAD><BODY>
...
<FONT face=sans-serif size=-2>
<BR>Maintained by: <!--#include virtual="author.inc" -->
<BR>Last modified: <!--#echo var="LAST_MODIFIED" -->
</FONT>
```

page after processing (as sent to the browser)

```
<HTML><HEAD><TITLE>
Esempio di SSI
</TITLE></HEAD><BODY>
...
<FONT face=sans-serif size=-2>
<BR>Maintained by: <B>Antonio Lioy</B>
<BR>Last modified: Thursday, 21-Feb-2002 18:53:28 MET
</FONT>
```

SSI/XSSI in action(2)

- **content of the file `title.inc`**

```
Esempio di SSI
```

- **conten of the file `author.inc`**

```
<B>Antonio Lioy</B>
```

- **REMARK: the file included with the include directive or the result of execution of the script (directive exec)**
  - can contain text and HTML
  - must respect the encoding of characters specified by HTML): e.g. quantità => quantità
- **once included must respect the HTML/CSS requirements (position of TAG, ecc.)**

SSI and XSSI: references

- **http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes .html**
- **http://httpd.apache.org/docs/howto/ssi.html**
- **http://www.bignosebird.com/ssi.shtml**
- **http://www.georgedillon.com/web/ssi.shtml**
- **http://www.georgedillon.com/web/ssivar.shtml**
- **http://msdn.microsoft.com/library/ default.asp?url=/library/en-us/dnproweb/html/ includingcontentsofdiskfile.asp**