# Programming web-based applications

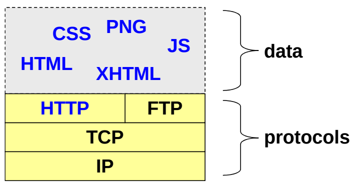
Antonio Lioy < lioy@polito.it >

***english version created and modified by***
Marco D. Aime < m.aime@polito.it >

***Politecnico di Torino***
***Dip. Automatica e Informatica***

## World Wide Web (WWW)

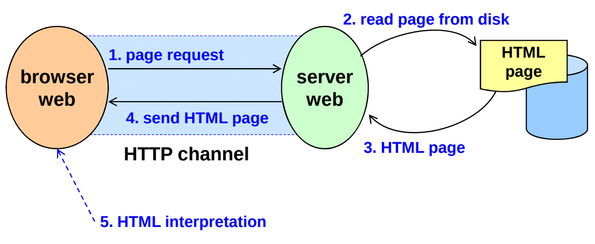
- **often simply abbreviated as "web"**
- **set of:**
  - communication protocols
  - data formats
- **built on top of TCP/IP channels**



## Protocols for the web

- **several existing protocols can be used (e.g. FTP)**
  - limitations / complexity since they were not designed for the web
- **a new application protocol has been defined:**
  - HTTP
- **the application protocol determines which functions are available (e.g. with FTP only GET and PUT of files)**

## The static web



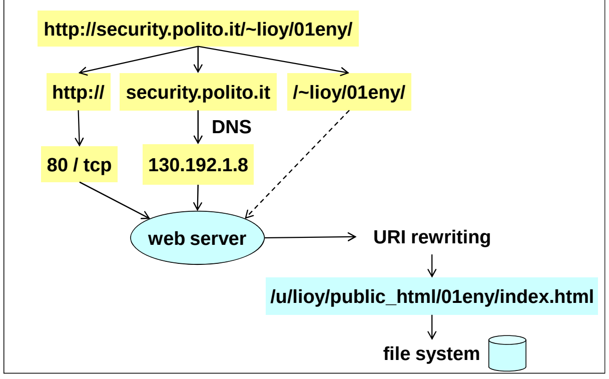
## Static web

- **the web page never changes its content**
- **… until the author does not change it explicitly**
- **the content of the 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 time it is requested
- **page implemented in HTML / CSS**

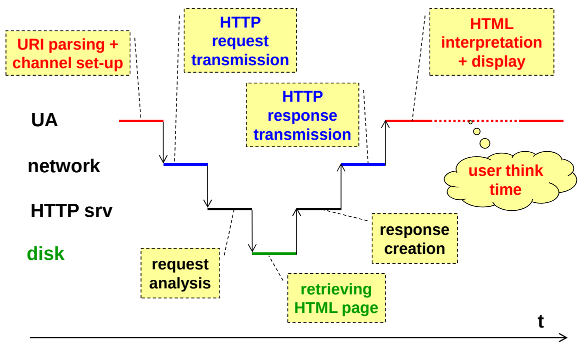
## Static web: pros and cons

- **every web page is associated with an HTML file**
- **(+) maximum efficiency (low load on CPU)**
- **(+) possibility to perform page caching:**
  - in RAM (by the server) or on disk (by client or proxy)
  - by the server
  - by the client
  - by proxies
- **(+) pages can be indexed by search engines**
- **(–) data is static**
- **(–) no adaptation to clients and their capabilities**

## Request of a static page



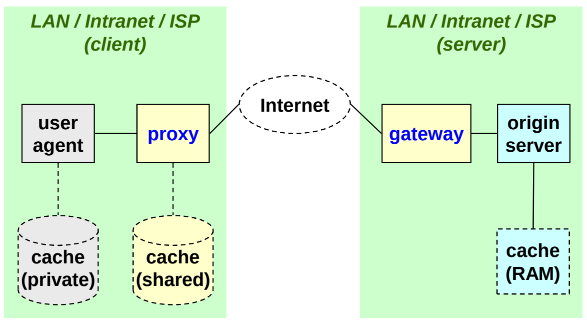
## Performance model for static web


URI parsing + channel set-up
HTTP request transmission
HTTP response transmission
HTML interpretation + display
UA
network
HTTP srv
disk
user think time
response creation
request analysis
retrieving HTML page
t

## Agent, server, proxy and gateway

- **User Agent = browser (but also spiders, robots, …)**
- **Origin Server = provider of the desired service**
- **intermediate elements may exist between UA and OS, acting as client and server at the same time:**
  - gateway
    - public interface for servers
    - e.g. for security or load balancing
  - (delegated) proxy
    - works on behalf of the client
    - forwards the request to the server or answers directly by using a cache
    - also for authentication

## Proxy and gateway – scheme



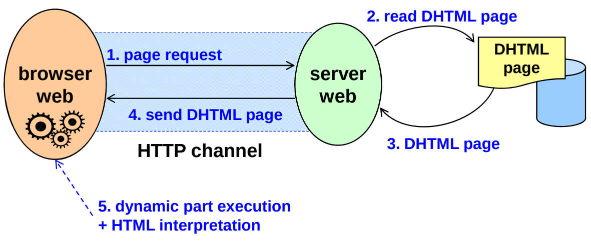
## Proxy

- **caches only static pages**
- **behaviour:**
  - transparent = does not modify the request (except mandatory parts)
  - non-transparent = re-writes the request (e.g. anonymiser)
- **UA configuration:**
  - explicit (requires intervention on the client)
  - implicit (requires intelligence in the network)
- **proxy hierarchies (e.g. POLITO, IT, EU) are possible**
- **often used by ISPs to improve clients' navigation speed**

1. **HTTP**
2. **HTML**
3. **CSS**

## Static web with dynamic pages

- **the client evaluates the dynamic content of the page (script, or Java applet, or Active-X control)**



## Static web with dynamic pages

- **pages change their content depending on user interaction**
  - e.g. context menu popping up when the mouse is positioned over a specific area
- **generically known as DHTML:**
  - HTML 4.0 or greater
  - CSS (Cascaded Style Sheet)
  - client side scripting languages
    - ECMA 262 (EcmaScript Edition 3) implemented as JavaScript or JScript
    - VBScript (only for IE 4.0 or greater)

## Dynamic pages: pros and cons

- **content presentation is variable**
- **(+) efficient (low CPU load on servers)**
- **(–) inefficient (medium-high CPU load on clients, especially for applets)**
- **(~) possibility to perform page caching**
- **(~) pages can be indexed by search engines (but only the static data …)**
- **(–) static data**
- **(–) functionality depends upon client capabilities (scripting languages, applet types, CPU, RAM, …)**

## Dynamic pages: applets

- **two types of applet:**
  - Java applet (requires a JVM in the browser)
  - ActiveX control (requires IE + Wintel)
- **problems:**
  - compatibility (which version of language / JVM ?)
  - load (require execution)
  - security (execution of a full program):
    - Java applet executes within a “sandbox”
    - activeX installs a DLL (!)

## Performance model for static web with dynamic pages

- **no difference w.r.t. static web for the network part and the server side**
- **increased computational and memory load on the client side:**
  - depends on the chosen technology
  - increasing load for
    - CSS
    - client-side scripts
    - Active-X controls
    - Java applets

## Client-side scripting

- **HTML is a page description language**
- **the only possible activity is following the links**
- **interactivity is added to HTML pages through some code to be interpreted at the client (by the browser):**
  - NS and SUN invented the LiveWire language, later renamed it JavaScript (but it’s not a subset of Java!)
  - MS invented VBScript (subset of VBA), and later JScript
  - JavaScript and JScript merged in ECMAScript:
    - ECMA-262 standard
    - popularly known as JavaScript (version >= 1.3)

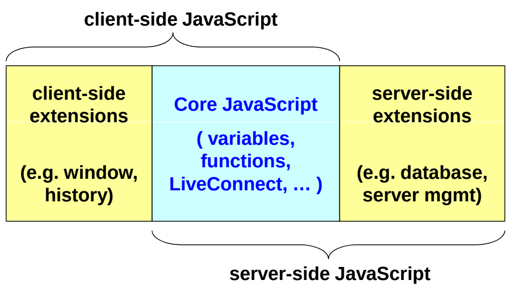
## Client-side scripting: what's for?

- **dynamically insert elements within HTML pages**
- **a function written in the scripting language of choice can be associated to some event triggered by page interaction:**
  - e.g. click on a figure
  - e.g. submission of a form
- **execute some code in reaction to an event**
  - validate data inserted in a form before submitting it to the server
    - it saves useless traffic on the network and simplifies the application logic on the server side

## JavaScript

- **interpreted language**
- **includes a limited set of commands required by client-side applications to:**
  - elaborate data inserted in the FORMs included in the HTML page
  - send commands to the browser (e.g. open/close windows)
  - execute some operations in reaction to an event triggered by a given user action (event handler)

## JavaScript core and extensions



## JavaScript and HTML pages

- to send JavaScript applications to the browser:
  - insert the JavaScript code inside the html page by using the tag <script>
  - import the code from an external file (with .js extension) by using <script src="..."">
  - specify a JavaScript expression as the value of an HTML attribute
  - insert a JavaScript expression as an event handler (DOM event handler) within specific HTML tags

## JavaScript: first example

```
<html>
<head></head>
<body>
  <script type="text/javascript">
    document.writeln("Ciao!")
  </script>
</body>
</html>
```

js1.html

## JavaScript: table of squares

```
<html>
  <head>
  <title>Table of squares</title>
  </head>
  <body>
  <h1>Table of squares</h1>
  <script type="text/javascript">
  <!--
  var i;
  for (i=1; i<20; i++) {
    document.writeln("<p>" + i + "^2 = " + i*i + "</p>");
  }
  // -->
  </script>
  </body>
</html>
```

## DOM event handler

- **you can associate JavaScript commands to events through 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, onblur, onsubmit, onreset, onchange, onload, onunload)
  - "JavaScript Code" is a sequence of JavaScript commands (often a function call)

## JS: second example

```
<html><head>
<title>Example: JS associated to onclick</title>
<script type="text/javascript">
function makeRed(x){
  obj = document.getElementById(x);
  obj.style.color="red";
}
</script></head><body>
<p id="id1" onclick="makeRed('id1')">
Click on this text to make it red!
</p>
</body></html>
```

js2.html

## JS: third example

- **when the same script is used for multiple pages, you may write it in an external file and link it in the HTML page**
- **the ".js" file must**
  - be a text file
  - have a name with max 8 characters
  - not contain the tag <script>

## JS: third example (2)

```
<html>
<head>
<script src="js3.js" type="text/javascript">
</script>
</head>
<body>
<p id="id1" onclick="makeRed('id1')">
Click on this text to make it red!</p>
</body>
</html>
```

js3.html

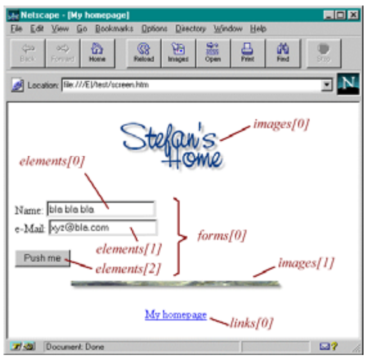
```
function makeRed(x) {
  obj = document.getElementById(x);
  obj.style.color="red"; }
```

js3.js

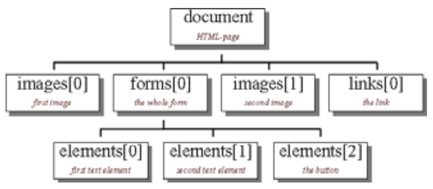
## DOM (Document Object Model)

- an “object-oriented view” of the HTML page
- provides a map of the web elements using an object-oriented metaphor
- DOM is a data structure not a language
- used in association with a client-side scripting (JavaScript, VBscript) to manipulate these data structures
- W3C tracks and tries to standardise the way the various scripting languages interact with the data structures at the basis of HTML
- DOM level 1:
  - www.w3.org/TR/1998/REC-DOM-Level-1-19981001

## DOM example



## DOM example: object hierarchy



```
<script ...>
  . . .
  name = document.forms[0].elements[0].value
  alert("Ciao " + name)
  . . .
</script>
```

## DOM: giving names to objects

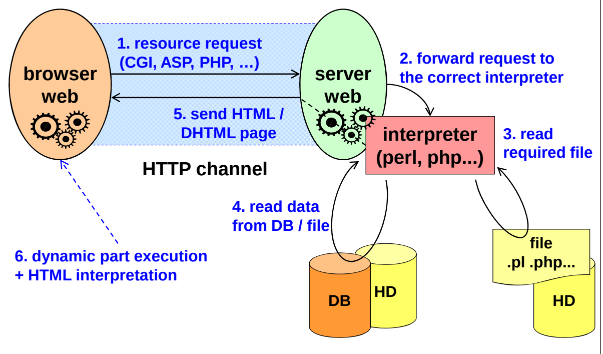
- **to simplify access to a given element (instead of using the hierarchical reference) you may assign a unique “name” to it:**
  - attribute “name” (available only for some tags)
  - attribute “id” (available for every tag)
- **example (“intro” is a reference to a specific instance of the tag <h1>):**

```
<html>
<body>
<h1 id="intro">Introduction</h1>
. . .
</body>
</html>
```

## DOM object hierarchy

```
window
  |
  +--parent, frames, self, top
  |
  +--location
  |
  +--history
  |
  +--document
       |
       +--forms
       |    |
       |  elements (text elements, textarea, checkbox, radio,
       |     password, select, button, submit, reset, ...)
       +--links
       |
       +--images
       |
       +--background
```

## The dynamic web


browser web
server web
1. resource request (CGI, ASP, PHP, ...)
2. forward request to the correct interpreter
interpreter (perl, php...)
3. read required file
file .pl .php...
HD
4. read data from DB / file
DB
HD
5. send HTML / DHTML page
HTTP channel
6. dynamic part execution + HTML interpretation

## Dynamic web

- **page is dynamically generated by the server**
- **its information content chages depending on:**
  - request sent by the client
  - content of a database
  - date/time of the request
- **techniques to implement the dynamic web:**
  - CGI
  - server-side scripting language (JavaScript/Jscript, VBScript, PHP, PerlScript, Python, ...)
  - SSI (Server Side Include)
  - servlet, JSP (Java Server Pages)

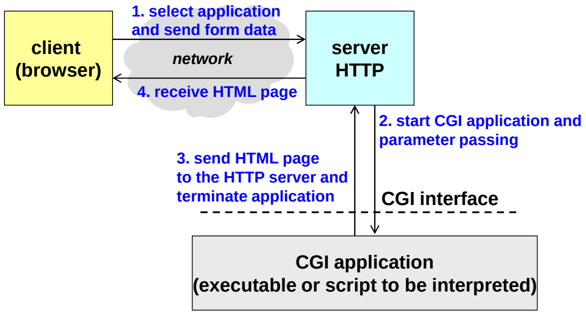
## Dynamic web: pros and cons

- **adaptation of pages to variable conditions:**
  - input provided by client
  - client capabilities
- **(+) maximum dynamicity for the data**
- **(+) optimal adaptability to clients and their capabilities**
- **(–) low efficiency (high CPU load)**
- **(–) impossible to perform page caching if the selection parameters are not inside the URL (e.g. are inside cookies)**
  - possible errors if caching is active
- **(–) pages cannot be indexed by search engines**

## CGI

- **Common Gateway Interface**
- **http://hoohoo.ncsa.uiuc.edu/cgi/interface.html**
- **RFC-3875**
- **the web server:**
  - starts the CGI application
  - passes possible parameters to it:
    - through stdin (POST, PUT methods)
    - through a modified URL (GET method)
  - receives result from stdout
  - result must be in web format (HTML/CSS/scripting client-side)

## CGI behavioural scheme


client (browser)
1. select application and send form data
network
4. receive HTML page
server HTTP
2. start CGI application and parameter passing
3. send HTML page to the HTTP server and terminate application
CGI interface
CGI application (executable or script to be interpreted)

## CGI: pros

- **general method**
- **available on every web server (IIS, Apache, …)**
- **application written in whatever way**
  - executable file (=more efficient)
  - interpreted script (=more flexible)

## CGI: cons

- **every call requires activating a process:**
  - high initialisation cost
  - high latency
  - creation / destruction of many processes
- **memory usage proportional to the number of processes active at the same time**
- **communication between the web server and the application is difficult (different memory spaces)**

## CGI: cons (II)

- **no mechanisms to share resources among CGI programs**
  - every access to a resource requires “opening” and “closing” the resource
  - session and transaction concepts do not exist
- **the graphic interface of the web application (i.e. the HTML tags) is embedded within the code**
- **paradigm not fit for applications with several concurrent users and requiring slow response times**

## CGI: possible improvements

- **use environment variables to communicate between the server and the application**
- **include one or more interpreters in the web server:**
  - (+) better activation speed
  - (+) better communication with the application
  - (+) lower memory occupation
  - (-) increased size of the server
- **pre-activation of the application (in N replicas) and inclusion in the server of a specific module to choose a free replica and communicate with it**
  - FastCGI

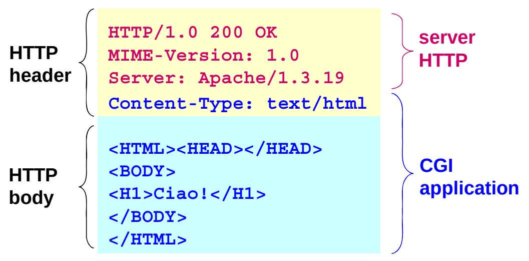
## Passing input parameters to CGI

- **three ways to transmit forms' data:**
  - standard input (when using POST or PUT)
  - modified URL (when using GET):
    - original CGI URL followed by '?' and by the list of data separated by '&'
    - the environment variable QUERY_STRING contains the part of the URL following '?'
  - command line (when using ISINDEX)
- **other information passed to the application through a set of environment variables (e.g. REMOTE_ADDR, HTTP_USER_AGENT)**

## Output generated by CGI

- **the application must return valid HTML**
  - use &agrave; &quot; &lt; &gt; …
- **the application must also return part of the HTTP headers; CGI/1.1 specifies the following headers:**
  - Content-Type:
    - MIME type of the response
  - Location:
    - if a URI, the server sends a redirect to the client
    - if a local document, the server sends it to the client
  - Status:
    - the server uses it as a status code in its header

## CGI: response generation



```
HTTP/1.0 200 OK
MIME-Version: 1.0
Server: Apache/1.3.19
Content-Type: text/html

<HTML><HEAD></HEAD>
<BODY>
<H1>Ciao!</H1>
</BODY>
</HTML>
```

## CGI example

- **http://security.polito.it/~lioy/cgi/cgiecho**
- **http://security.polito.it/~lioy/cgi.htm (look at the difference between GET and POST)**

## cgic

- **ANSI C library fro CGI programming**
- **http://www.boutell.com/cgic/**
- **extracts forms' data, correcting browsers' errors**
- **transparent treatment of GET and POST**
- **read form data or an uploaded file**
- **functions to set and read cookies**
- **correct treatment of CR and LF in text form**
- **extract forms' data (string, int, real, single and multiple choices), controlling ranges of numeric types**
- **load the CGI env. variables in not null strings**
- **compatible with every CGI server (U*ix, Win*)**

## Libwww

- **C library used to write HTTP+HTML clients**
- **also used to write robots**
- **http://www.w3.org/Library/**

## Server-side scripting

- **different technologies, all characterised by having, inside the page file, some scripting code merged with the template "HTML + client-side scripting"**
- **ASP (Microsoft)**
  - VBscript
  - JScript
  - implementation also for Apache (with PerlScript)
- **PHP (open source)**
  - developed for Apache
  - also for IIS
  - can be used both as a general scripting language and for CGI

## Server-side scripting (2)

- **JSP (Sun), hybrid technology**
  - the code is embedded in the HTML template (as for the other technologies for server-side scripting)
  - the code includes
    - scripting elements (as other server-side languages)
    - directives
    - actions (proprietary tags, XML & NS like)
  - the pages are translated into servlets by the web server

## SSI (Server Side Include)

- **introduce directives in the HTML code in the form of comments**
  - if SSI is ***not*** supported by the server web, directives are ignored
  - if SSI is supported, in the HTML page returned to the client, the directives are replaced by the text resulting from the elaboration
- **add new environment variables**
- **do not replace CGI/ASP/..., but introduce the possibility to add dynamicity to HTML pages performing simple operations**

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

## SSI (2)

- **in IIS, the HTML pages containing SSI directives must use the extension ".shtm" or ".shtml"**
- **you can configure web servers to elaborate the SSI directives also for pages with the extensions ".htm" or ".html"**
- **server web**
  - Apache supports SSI (and XSSI from version 1.2)
  - IIS supports only the directive #include of SSI
    - it must be inserted in the HTML part
    - cannot be produced by ASP code
    - in IIS, the other SSI functionalities can be provided with ASP objects

## SSI environment variables

- **DOCUMENT_NAME: the name of the current file**
- **DOCUMENT_URI: the virtual path to this document (e.g. /docs/tutorials/foo.shtml)**
- **QUERY_STRING_UNESCAPED: search string sent by the client, with every special shell character, preceded by '\'**
- **DATE_LOCAL: current date, local time zone; subject to the parameter `timefmt` of the command `config`**
- **DATE_GMT: similar to DATE_LOCAL, but relative to the Greenwich time**
- **LAST_MODIFIED: date of last modification of the current document; also subject to `timefmt`**

## SSI directives

- **#config: allows setting some parameters**
  - errmsg: message returned in case of error when parsing of the SSI directives
  - timefmt: date and time format; definition string like the one used by the Unix system function strftime( )
  - sizefmt: format for the file size
    - 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 environment variable (tag: var) passed as a parameter**

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

- **#exec: executes a shell command or a CGI script whose name is passed as parameter and returns the corresponding output; supported tags:**
  - cmd: shell command (Unix: /bin/sh, Win32: cmd.exe) identified by the string
  - cgi: CGI script identified by the string (virtual path); no output mangling but conversion from URI to <A>

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

## SSI directives (3)

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

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

- **#fsize: returns the size of a file whose name is passed as parameter; the format is configurable with sizefmt; supported tags:**
  - virtual: virtual path (no access to CGI scripts)
  - file: relative physical path starting from the current directory (no absolute paths, no use of '../')

```
<!--#fsize virtual="VIRT_PATH_NOME_FIL" -->
<!--#fsize file="REL_PATH_NOME_FILE" -->
```

## SSI directives (4)

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

```
<!--#include virtual="VIRT_PATH_NOM_FILE" -->
<!--#include file="REL_PATH_NOME_FILE" -->
```

  - attention! the included file cannot contain SSI directives

## SSI examples

- inserts local date and time in standard format:

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

- inserts local date and time in non-standard format:

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

- executes a system command (the text <DIR> in the output of the `dir` command can lead to wrong formatting by the browser)

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

## SSI examples (2)

- inserts a footer shared with other pages

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

- inserts the date of last modification of the current page; solution 1 (if you change the page name, you must update the directive)

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

- inserts the date of last modification of the current page; solution 2 (the same directive can be used for all pages)

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

## SSI examples (3)

- set an error message different from the standard one in case of problems when parsing the SSI directives

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

  - standard error message; the directive code is replaced by the following text

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

  - error message set with the directive

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

## Example of SSI/XSSI workflow

page before elaboration
(as stored on the 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 elaboration
(as sent to the client)

```
<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>
```

## Example of SSI/XSSI workflow (2)

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

```
Esempio di SSI
```

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

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

- **NOTE: files included with the directive include or the result of script execution (directive exec)**
  - can contain text and HTML
  - must comply with the HTML character encoding): e.g. quantità => quantit&agrave;
- **once included, they must comply with the requirements of HTML/CSS (TAG position, etc.)**

## Server-side or client-side?

- **server-side:**
  - (pro) higher security
  - (con) server overload
- **client-side:**
  - (pro) computation on the client
  - (con) client capabilities (functionality and performance)
  - (con) lower security (tampered with by the user)
- **in general:**
  - better server-side for security and functionality
  - better client-side to improve performance
  - often used together simultaneously

## Server-side vs. client-side

- sometimes they aren't equivalent
- example (content of prova.asp):

```
<%
var d=new Date();
var h=d.getHours();
var m=d.getMinutes();
Response.write(h + ":" + m);
%>
<script type="text/javascript">
var d=new Date();
var h=d.getHours();
var m=d.getMinutes();
document.write(h + ":" + m);
</script>
```