



Programmazione WEB Avanzata per la Fisica Sperimentale

WEB



Strumenti per il trattamento
dei dati client-side al tempo di HTML5

```
1 <html>
2
3   <script>
4     function getInfo()
5     {
6       alert("Info will appear here") ;
7     }
8   </script>
9
10  <form>
11    <input type = "submit"
12      value = "Get info"
13      onClick= "getInfo()"/>
14  </form>
15
16 </html>
17
```

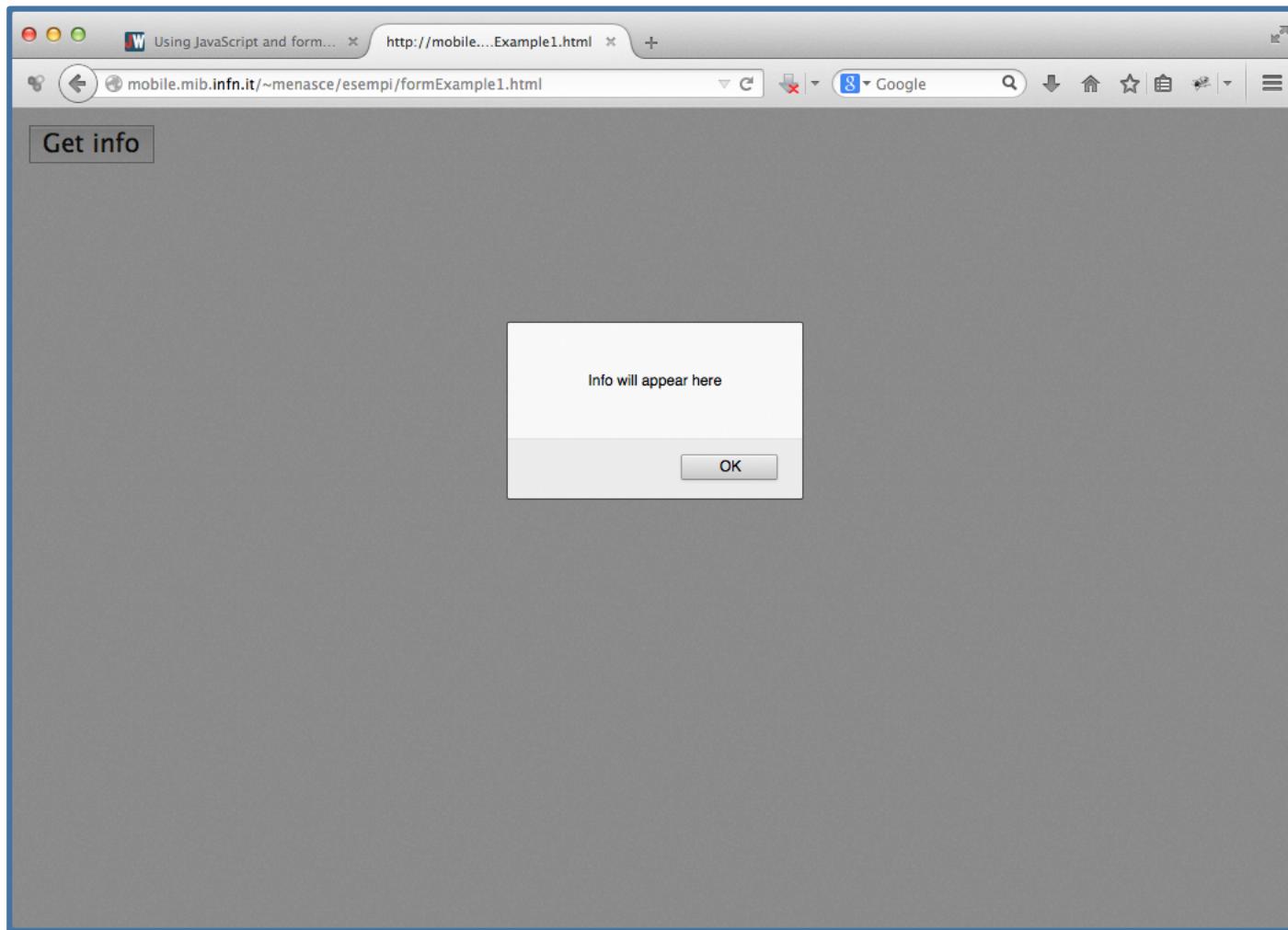
Una funzione in JavaScript attivata da un bottone del form (implementazione triviale di una pagina interattiva)

Il form espresso in termini di puro html

Vogliamo a questo punto approfondire le funzionalità, oltre a dare cenni di sintassi, di JavaScript, lo *scripting language* che permette di manipolare programmaticamente gli elementi di un Browser ed i suoi contenuti.

Il concetto fondamentale è che il comportamento di un Browser è modificabile, anche in modo interattivo, tramite opportune istruzioni caricate in documento HTML nella forma di codice Javascript, che viene eseguito dal browser stesso.

Cenni su JavaScript



File Edit Search Preferences Shell Macro Windows Help

/user/gr1/e831/menasce/public_html/esempi/formExample2.html - /user/gr1/e831/menasce/public_html/esempi/ L: 26 C: 0

```
1 <html>
2
3   <script>
4     function getInfo()
5     {
6
7
8
9
10    }
11
12
13
14
15
16  }
17 </script>
18
19 <form>
20   <input type="submit"
21     value="Get info"
22     onClick="getInfo()"/>
23 </form>
24
25 </html>
26
```

window è una referenza ad un particolare oggetto di Javascript, oggetto che rappresenta la finestra del browser.

Essendo un oggetto, è dotato di proprietà (valori) e di metodi (funzioni per la sua manipolazione).

In particolare una sua proprietà è la variabile **navigator**, a sua volta una referenza ad un altro oggetto che rappresenta il navigatore in cui essa compare, dotato di tutti gli attributi relativi dei quali vorremo fornire informazioni all'utente.

Ampliamo ora le funzionalità fornite dal bottone “Get info”

File Edit Search Preferences Shell Macro Windows Help

/user/gr1/e831/menasce/public_html/esempi/formExample2.html byte 661 of 661 L: 26 C: 0

```
1 <html>
2
3   <script>
4     function getInfo()
5     {
6       var info ;
7       var nav = window.navigator ;
8       info    = "Outer heigth: " + window.outerHeight + "\n";
9       info    += "Outer width: " + window.outerWidth + "\n";
10      info   += "Inner heighth: " + window.innerHeight + "\n";
11      info   += "Inner width: " + window.innerWidth + "\n";
12      info   += "Navigator: " + nav.appCodeName +
13          " V: " + nav.appVersion + "\n";
14      info   += "Platform: " + nav.platform + "\n";
15      alert("Info (local): \n\n" + info) ;
16    }
17  </script>
18
19  <form>
20    <input type="submit"
21          value="Get info"
22          onClick="getInfo()"/>
23  </form>
24
25</html>
26
```

outerHeight

innerHeight

outerWidth

Get info

Info (local):

Outer height: 436

Outer width: 611

Inner height: 390

Inner width: 611

Navigator: Mozilla V: 5.0 (Macintosh)

Platform: MacIntel

OK

File Edit Search Preferences Shell Macro Windows Help

formExample3.html *formExample4.html formExample2.html

/user/gr1/e831/menasce/public_html/esempi/formExample4.html byte 258 of 2453 L: 10 C: 8

```
30 <form id="sampleform" enctype="multipart/form-data" method="post" action="">
31   Name: <input type="text" name="Name" />
32   Email: <input type="text" name="Email" />
33   <p/>
34   <input type="checkbox" name="Subscribe" checked/>
35   Subscribe for Newsletter: <p> Format:
36   <input type="radio" name="Format" value="html" checked="checked"/>HTML
37   <input type="radio" name="Format" value="text" />Plain Txt
38   <p/> Type of subscription you want:
39   <select name="Type">
40     <option value="standard" >Standard - Free</option>
41     <option value="professional">Professional - Paid</option>
42   </select>
43   <p/>
44   Comments to the editor:<br/>
45   <textarea name="comments" rows="4" cols="50"></textarea>
46   <input type="image" alt="Submit button" src="ico/submit.gif" />
47   Upload file:<input type="file" name="file" />
48   Password: <input type="password" name="password" />
49   <input type="submit" value="Send" onClick="return validateForm()" />
50   <input type="reset" value="Reset" />
51   <input type="hidden" name="hidden" value="hidden value" />
52
53 </form>
54
55 </html>
```

Vediamo ora come costruire un form più complesso...

The screenshot shows a web browser displaying the HTML code above. The browser interface includes a title bar, menu bar, and address bar. The main content area shows the rendered HTML form. Several input fields and controls have been highlighted with red boxes, corresponding to the numbered lines in the code. The highlighted elements include:

- Line 31: The Name and Email input fields.
- Line 34: The checked checkbox for newsletter subscription.
- Line 36: The radio button for "HTML" which is selected (indicated by a checked state).
- Line 40: The option for "Standard - Free" in the dropdown menu, which is selected (indicated by a checked state).

File Edit Search Preferences Shell Macro Windows Help

formExample3.html *formExample4.html formExample2.html

/user/gr1/e831/menasce/public_html/esempi/formExample4.html 2453 bytes L: --- C: ---

```
21     "Subscribe: " + subscribe + "\n" +
22     "Format: " + format + "\n" +
23     "Type: " + type + "\n" +
24     "fileName: " + fileName + "\n" +
25     "Password: " + password + "\n" +
26     "Admin: " + admin ;
27 alert(s) ;
28 }
29 </script>
30 <form id="sampleform" enctype="multipart/form-data" method="post" action="">
31 Name: <input type="text" name="Name" />
32 Email: <input type="text" name="Email" />
33 <p/>
34 <input type="checkbox" name="checkbox" value="checkbox" checked="checked" /> Subscribe for Newsletter:
35 <input type="radio" name="radio" value="radio" checked="checked" /> Standard
36 <input type="radio" name="radio" value="radio" /> Professional
37 <p/> Type of subscription you want:
38 <select name="Type">
39   <option value="standard" selected="selected">Standard</option>
40   <option value="professional" >Professional</option>
41 </select>
42 <p/>
43 Comments to the editor:<br/>
44 <input type="text" name="comments" value="Comments to the editor:" />
45 <input type="image" alt="Submit button" src="ico/submit.gif" />
46 Upload file:<input type="file" name="file" value="Upload file: Browse... No file selected." />
47 Password: <input type="password" name="password" value="Password:" />
48 <input type="submit" value="Submit" /> <input type="reset" value="Reset form" />
```

http://mobile.mib.infn.it/~menasce/esempi/formExample4.html

Name: Email:

Subscribe for Newsletter:

Format: HTML Plain Txt

Type of subscription you want: Standard – Free

Comments to the editor:

SIGN UP NOW!

Upload file: Browse... No file selected.

Password: Submit Reset form

File Edit Search Preferences Shell Macro Windows Help

/user/gr1/e831/menasce/public_html/esempi/formExample4.html byte 1696 of 2392 L: 43 C: 30

```
1 <html>
2 <script>
3     function processForm(thisForm)
4     {
5         var name, email, subscribe, format, type, fileName, password, admin ;
6         var f = thisForm ;
7         for(var i=0; i < f.length; ++i)
8         {
9             if(f[i].name == 'Name' ) {name      = f[i].value;}
10            if(f[i].name == 'Email' ) {email     = f[i].value;}
11            if(f[i].name == 'Subscribe') {subscribe = f[i].value;}
12            if(f[i].name == 'Format' ) {format    = f[i].value;}
13            if(<input type="submit" name="Submit" value="Submit"
14                   onClick="processForm(this.form)">
15            if(f[i].name == 'Admin' ) {admin     = f[i].value;}
16        }
17
18        var s = "Name:      " + name      + "\n" +
19              "Email:      " + email     + "\n" +
20              "Subscribe: " + subscribe + "\n" +
21              "Format:    " + format    + "\n" +
22              "Type:      " + type      + "\n" +
23              "fileName:   " + fileName  + "\n" +
24              "Password:  " + password  + "\n" +
25              "Admin:     " + admin     ;
26        alert(s) ;
27    }
28 </script>
29 <form id="sampleform" enctype="multipart/form-data" method="post" action="">
30     Name: <input type="text" name="Name" />
31     Email: <input type="text" name="Email" />
32     <p>
33     <input type="checkbox" name="Subscribe" checked/>
34     Subscribe for Newsletter: <p> Format:
35     <input type="radio" name="Format" value="html" checked="checked"/>HTML
```

Laboratories

Perl and CGI Tuto...

Name: Dario

 SubscribFormat: F

Type of subscri

Comments to

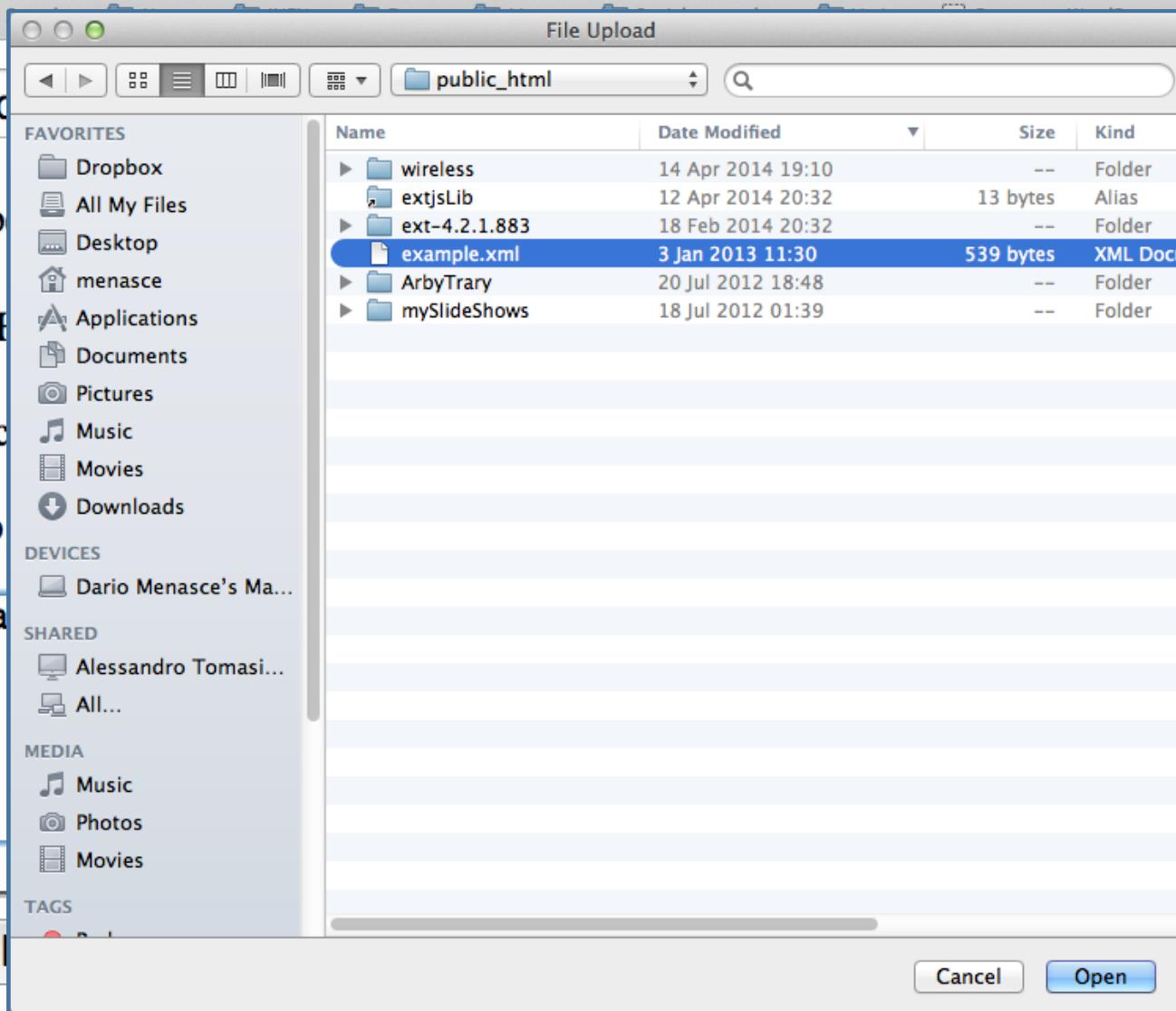
Bla bla a

Upload file:

Password:

Submit

Reset form



mobile.mib.infn.it/~menasce/esempi/formExample4.html

balonei

[Laboratories](#) [Search](#) [News](#) [INFN](#) [Docs](#) [Linux](#) [Social networks](#) [Varie](#) [Pages ← WordPr...](#) [Perl and CGI Tuto...](#)

Name: Dario Livio

Email: dario.menasce@mib.infn.it

 Subscribe for Newsletter:Format: HTML Plain Txt

Type of subscription you want: Standard – Free

Comments to the editor:

Bla bla and bla

Upload file: example.xml

Password: *****

Labs

Search

News

INFN

Docs

Linux

Social networks

Varie

Pages — WordPress

Perl and CGI Tuto...

>>>

Name: Dario Livio

Email: dario.menasce@mib.infn.it

 Subscribe for Newsletter:Format: HTML Plain Txt

Type of subscription you want: S

Comments to the editor:

Bla bla and bla

Name: Dario Livio
Email: dario.menasce@mib.infn.it
Subscribe: on
Format: text
Type: standard
fileName: example.xml
Password: myWord
Admin: master

OK

Upload file: example.xml

Password: *****

Submit

Reset form

File Edit Search Preferences Shell Macro Windows Help

*printAllEnv.html printAllEnv.pl

/user/gr1/e831/menasce/public_html/esempi/printAllEnv.html byte 225 of 225 L: 16 C: 0

```
1 <html>
2
3 <head>
4 </head>
5
6 <body>
7
8 <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/printAllEnv.pl"
9 method="GET">
10 <input type="submit" value="Get ENV parameters">
11 </form>
12
13 </body>
14
15 </html>
16
```

In questa parte del corso affronteremo le prime problematiche del CGI programming.

Introduciamo questo vasto argomento con un semplice esempio, un documento HTML che rappresenti un FORM di input dotato di un singolo bottone.

Premendo il bottone l'utente richiede l'attivazione di uno script ***server-side (remoto)*** il cui output viene ricevuto e opportunamente rappresentato dal browser (***client-side***).

Ciò che lo script restituisce al browser è l'insieme di tutte le variabili di ambiente disponibili nella ***shell***, creata come sotto-processo da ***httpd***, nell'ambito della quale lo script in questione viene eseguito.

Vediamo come configurare il server Apache affinchè sappia agire come tramite per l'esecuzione di programmi in modalità CGI (Common Gateway Interface).

L'idea di base è quella di far interagire il browser (per conto dell'utente) con il server, creando un canale di comunicazione bi-direzionale per la trasmissione di input al programma e di output al browser.

Poichè permettere l'esecuzione di programmi sul server da parte di client non identificati può essere assai pericoloso, occorre creare un'opportuna zona di sandboxing entro la quale permettere questo tipo di azioni.

Il problema della sicurezza è ovviamente molto più vasto, ma non abbiamo tempo di trattarlo in profondità.

```
#  
# ScriptAlias: This controls which directories contain server scripts.  
# ScriptAliases are essentially the same as Aliases, except that  
# documents in the target directory are treated as applications and  
# run by the server when requested rather than as documents sent to the  
# client. The same rules about trailing "/" apply to ScriptAlias  
# directives as to Alias.  
#  
ScriptAlias /cgi-bin/ "/home/guest/public_html/cgi-bin/"
```

```
[root@webber23 cgi-bin]# systemctl start httpd  
[root@webber23 cgi-bin]# systemctl stop firewalld  
[root@webber23 cgi-bin]# setenforce 0  
[root@webber23 cgi-bin]#
```

```
[guest@webber23 public_html]$ pwd  
/home/guest/public_html  
[guest@webber23 public_html]$ mkdir cgi-bin  
[guest@webber23 public_html]$ mkdir cgi-bin/script  
[guest@webber23 public_html]$ mkdir cgi-bin/script/esempi  
[guest@webber23 public_html]$
```

File Edit Search Preferences Shell Macro Windows Help

*printAllEnv.html printAllEnv.pl

/user/gr1/e831/menasce/public_html/esempi/printAllEnv.html byte 225 of 225 L: 16 C: 0

```
1 <html>
2
3   <head>
4   </head>
5
6   <body>
7
8     <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/printAllEnv.pl"
9       method="GET">
10    <input type="submit" value="Get ENV parameters">
11  </form>
12
13 </body>
14
15 </html>
16
```

Come si presenta questa pagina una volta invocata nel browser?

http://mobile.mib.infn.it/~menasce/esempi/printAllEnv.html

Get ENV parameters

```
[guest@webber23 esempi]$ pwd
/home/guest/public_html/cgi-bin/scripts/esempi
[guest@webber23 esempi]$ chmod +x printAllEnv.pl
```

File Edit Search Preferences Shell Macro Windows Help

*printAllEnv.html printAllEnv.pl

/user/gr1/e831/menasce/public_html/esempi/printAllEnv.html byte 225 of 225 L: 16 C: 0

```
1 <html>
2
3 <head>
4 </head>
5
6 <body>
7
8 <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/printAllEnv.pl"
9 method="GET">
10 <input type="submit" value="Get ENV parameters">
11 </form>
12
```

Abbiamo visto che la pagina viene invocata mediante il seguente URL:

<http://mobile.mib.infn.it/~menasce/esempi/printAllEnv.html>

Occorre prestare attenzione a quello che NON è un dettaglio: la **action** associata al **form** specifica necessariamente lo URL **completo** con il nome del programma da eseguire, ossia:

<http://mobile.mib.infn.it/cgi-bin/scripts/esempi/printAllEnv.pl>

Non possiamo specificare il solo nome, ossia **action="printAllEnv.pl"** perché, in quel caso, lo URL verrebbe composto in modo automatico assumendo che si tratti di un indirizzo relativo alla pagina corrente, col risultato che lo URL trasmesso diverrebbe:

<http://mobile.mib.infn.it/~menasce/esempi/printAllEnv.pl>

Ma questo non corrisponderebbe alla richiesta di **esecuzione** dello script **printAllEnv.pl** bensì di **trasferimento** al browser del suo source, che non corrisponde a ciò che vogliamo!

File Edit Search Preferences Shell Macro Windows Help

*printAllEnv.html temp.lis printAllEnv.pl /user/gr1/e831/menasce/public_html/esempi/ L: 33 C: 0

```
1 #!/usr/bin/perl
2
3 print <<EOB ;
4 Content-type: text/html
5
6 <style type="text/css">
7 #stile
8 {
9   align      : center;
10  color     : blue  ;
11  font-weight: bold  ;
12 }
13 #chiave
14 {
15   align      : left;
16  color     : red   ;
17  font-weight: bold  ;
18 }
19 </style>
20
21 <table>
22 <tr id="stile">
23   <td> Key </td><td> Value </td>
24 </tr>
25 EOB
26
27 foreach $key (sort keys %ENV)
28 {
29   print(" <tr> <td id='chiave'> $key </td><td> $ENV{$key}</td> </tr>\n") ;
30 }
31
32 print("</table>\n") ;
33
```

Vediamo ora il codice dello script la cui esecuzione abbiamo richiesto:

L'output prodotto dallo script consta di puro HTML, con uno *http-header* rivolto al *browser* per indicare il tipo di contenuto trasmesso (in formato html) consistente in una tabella: vedremo nella slide seguente come interpretare l'output prodotto da questo script.

Si completa poi la tabella con le celle che devono contenere, per ogni elemento della hash **%ENV**, il nome di una variabile d'ambiente ed il suo corrispondente valore (**%ENV** contiene le variabili d'ambiente nel quale lo script viene eseguito). **%ENV** è una variabile implicita di **perl**, sempre disponibile e popolata al momento dell'esecuzione

L'output dello script

Key	Value
DOCUMENT_ROOT	/var/www/html
GATEWAY_INTERFACE	CGI/1.1
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	en-US,en;q=0.5
HTTP_CONNECTION	keep-alive

```

<style type="text/css">
#stile
{
  align      : center;
  color      : blue   ;
  font-weight: bold   ;
}
#chiave
{
  align      : left;
  color      : red    ;
  font-weight: bold   ;
}
</style>

<table>
<tr id="stile"> <td> Key </td><td> Value </td> </tr>
<tr > <td id='chiave'> DOCUMENT_ROOT </td><td> /var/www/html </td> </tr>
<tr > <td id='chiave'> GATEWAY_INTERFACE </td><td> CGI/1.1 </td> </tr>
<tr > <td id='chiave'> ..... </td><td> ..... </td> </tr>
</table>

```

Key	Value
DOCUMENT_ROOT	/var/www/html
GATEWAY_INTERFACE	CGI/1.1
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	en-US,en;q=0.5
HTTP_CONNECTION	keep-alive
HTTP_COOKIE	_utma=201308338.470728258.1379605070.1402900427.1402927277.94; _utmz=201308338.1379605070.1.1.utmcsr=(direct) utmccn=(direct) utmcmd=(none); __utma=211720397.872847529.1384881328.1384881328.1384881328.1
HTTP_DNT	1
HTTP_HOST	mobile.mib.infn.it
HTTP_REFERER	http://mobile.mib.infn.it/~menasce/esempi/printAllEnv.html
HTTP_USER_AGENT	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:30.0) Gecko/20100101 Firefox/30.0
PATH	/sbin:/usr/sbin:/bin:/usr/bin
QUERY_STRING	
REMOTE_ADDR	212.189.204.8
REMOTE_PORT	49930
REQUEST_METHOD	GET
REQUEST_URI	/cgi-bin/scripts/esempi/printAllEnv.pl
SCRIPT_FILENAME	/var/www/cgi-bin/scripts/esempi/printAllEnv.pl
SCRIPT_NAME	/cgi-bin/scripts/esempi/printAllEnv.pl
SERVER_ADDR	212.189.204.213
SERVER_ADMIN	root@localhost
SERVER_NAME	mobile.mib.infn.it
SERVER_PORT	80
SERVER_PROTOCOL	HTTP/1.1
SERVER_SIGNATURE	Apache/2.2.15 (Red Hat) Server at mobile.mib.infn.it Port 80

I files di logging



```
[root@webber23 cgi-bin]# tail /var/log/httpd/error_log
[Mon Oct 12 11:02:03.681504 2015] [core:notice] [pid 2504] AH00094: Command line: '/usr/sbin/htt
pd -D FOREGROUND'
[Mon Oct 12 11:02:26.397517 2015] [core:error] [pid 2509] (13)Permission denied: [client 212.189
.204.58:55146] AH00035: access to /~guest/sendAudio.html denied (filesystem path '/home/guest')
because search permissions are missing on a component of the path
[Mon Oct 12 11:03:24.752647 2015] [core:error] [pid 2506] (13)Permission denied: [client 212.189
.204.58:55151] AH00035: access to /~guest/sendAudio.html denied (filesystem path '/home/guest')
because search permissions are missing on a component of the path
[Mon Oct 12 11:35:10.384889 2015] [mpm_prefork:notice] [pid 2504] AH00170: caught SIGWINCH, shut
ting down gracefully
[Mon Oct 12 11:35:20.909147 2015] [core:notice] [pid 2616] SELinux policy enabled; httpd running
as context system_u:system_r:httpd_t:s0
[Mon Oct 12 11:35:20.910685 2015] [suexec:notice] [pid 2616] AH01232: suEXEC mechanism enabled (
wrapper: /usr/sbin/suexec)
[Mon Oct 12 11:35:20.944450 2015] [auth_digest:notice] [pid 2616] AH01757: generating secret for
digest authentication ...
[Mon Oct 12 11:35:20.945459 2015] [lbmethod_heartbeat:notice] [pid 2616] AH02282: No slotmem fro
m mod_heartmonitor
[Mon Oct 12 11:35:20.962552 2015] [mpm_prefork:notice] [pid 2616] AH00163: Apache/2.4.6 (Scienti
fic Linux) OpenSSL/1.0.1e-fips mod_fcgid/2.3.9 configured -- resuming normal operations
[Mon Oct 12 11:35:20.962580 2015] [core:notice] [pid 2616] AH00094: Command line: '/usr/sbin/htt
pd -D FOREGROUND'
[root@webber23 cgi-bin]#
```

I files di logging



```
[root@hal9000 ~]# tail /var/log/httpd/access_log
121.42.14.70 - - [12/Oct/2015:11:41:20 +0200] "GET / HTTP/1.1" 200 15601 "-" "Java/1.7.0_79"
121.42.14.70 - - [12/Oct/2015:11:41:20 +0200] "GET / HTTP/1.1" 200 15601 "-" "Java/1.7.0_79"
crawl-66-249-64-156.googlebot.com - - [12/Oct/2015:11:43:36 +0200] "GET /~menasce/TriDAS/html/de/d60/extjs-good
_2source_2widgets_2layout_2BorderLayout_8js-source.html HTTP/1.1" 200 65792 "-" "Mozilla/5.0 (iPhone; CPU iPhon
e OS 8_3 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12F70 Safari/600.1.4 (compat
ible; Googlebot/2.1; +http://www.google.com/bot.html)"
crawl-66-249-64-156.googlebot.com - - [12/Oct/2015:11:43:36 +0200] "GET /~menasce/TriDAS/html/de/d60/extjs-good
_2source_2widgets_2layout_2BorderLayout_8js-source.html HTTP/1.1" 200 65792 "-" "Mozilla/5.0 (iPhone; CPU iPhon
e OS 8_3 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12F70 Safari/600.1.4 (compat
ible; Googlebot/2.1; +http://www.google.com/bot.html)"
spider-5-255-253-11.yandex.com - - [12/Oct/2015:11:44:38 +0200] "GET /~malvezzi/MilanoGroupMeeting/img024.GIF H
TTP/1.1" 304 - "-" "Mozilla/5.0 (compatible; YandexImages/3.0; +http://yandex.com/bots)"
spider-5-255-253-11.yandex.com - - [12/Oct/2015:11:44:38 +0200] "GET /~malvezzi/MilanoGroupMeeting/img024.GIF H
TTP/1.1" 304 - "-" "Mozilla/5.0 (compatible; YandexImages/3.0; +http://yandex.com/bots)"
crawl-66-249-64-146.googlebot.com - - [12/Oct/2015:11:49:38 +0200] "GET /~menasce/TriDAS/html/doxygen.css HTTP/
1.1" 200 8077 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
crawl-66-249-64-146.googlebot.com - - [12/Oct/2015:11:49:38 +0200] "GET /~menasce/TriDAS/html/doxygen.css HTTP/
1.1" 200 8077 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
crawl-66-249-64-151.googlebot.com - - [12/Oct/2015:11:49:39 +0200] "GET /~menasce/TriDAS/html/tabs.css HTTP/1.1
" 200 1758 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
crawl-66-249-64-151.googlebot.com - - [12/Oct/2015:11:49:39 +0200] "GET /~menasce/TriDAS/html/tabs.css HTTP/1.1
" 200 1758 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
[root@hal9000 ~]#
```



```
1 <html>
2
3   <head>
4   </head>
5
6   <body>
7
8     <center>
9       <table border="yes" cellpadding="2px" cellspacing="2px">
10      <tr>
11        <td> India </td>
12        <td> Santo Domingo </td>
13      </tr>
14      <tr>
15        <td><a href="#"> 340" width="340"></a></td>
16        <td><a href="#"> 340" width="340"></a></td>
17      <tr>
18        </tr>
19        <td> Chicago </td>
20        <td> Fermilab </td>
21      </tr>
22      <tr>
23        <td><a href="#"> 340" width="340"></a></td>
24        <td><a href="#"> 340" width="340"></a></td>
25      </tr>
26    </table>
27  </center>
28
29 </body>
30
31 </html>
32
33
34
```

Vediamo ora il caso in cui la pagina web debba rappresentare delle semplici fotografie, sempre in modalità statica (senza interattività di sorta)

http://mobile..../pictures.html

mobile.mib.infn.it/~menasce/esempi/pictures.html

Google

India



Santo Domingo



Chicago



E se volessimo poi trasformare le immagini in **hyperlinks**, in modo che cliccandovi sopra si potessero ricevere le foto in formato originale?

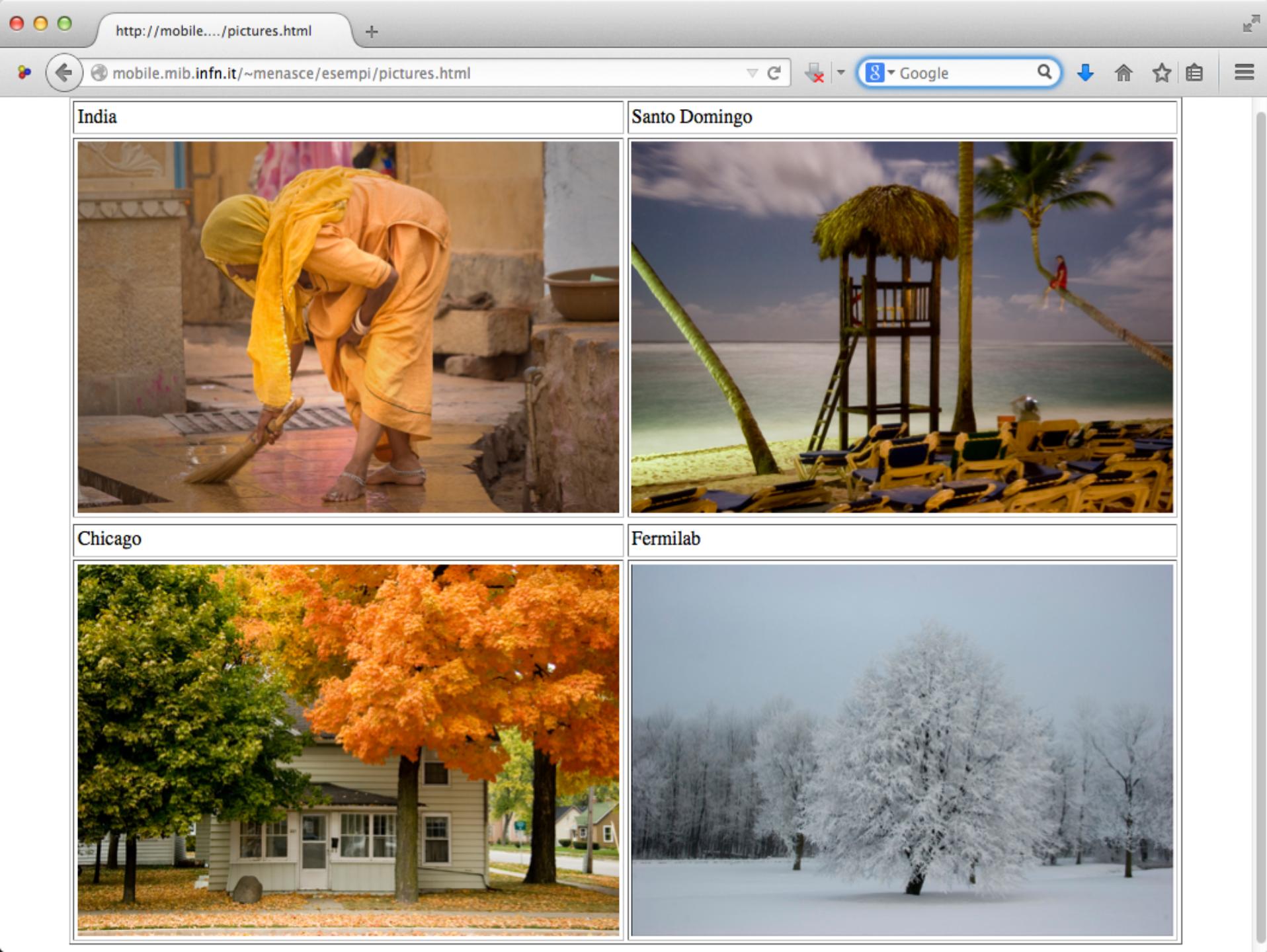


File Edit Search Preferences Shell Macro Windows Help

sendAudio.html pictures.html /user/gr1/e831/menasce/public_html/esempi/

/user/gr1/e831/menasce/public_html/esempi/pictures.html byte 662 of 678 L: 34 C: 0

```
1 <html>
2
3   <head>
4   </head>
5
6   <body>
7
8     <center>
9       <table border="yes" cellpadding="2px" cellspacing="2px">
10      <tr>
11        <td> India </td>
12        <td> Santo Domingo </td>
13      </tr>
14      <tr>
15        <td><a href="images/A.jpg"></a></td>
16        <td><a href="images/B.jpg"></a></td>
17      </tr>
18    </table>
19    <tr>
20      <td> Chicago </td>
21      <td> Fermilab </td>
22    </tr>
23    <tr>
24      <td><a href="images/C.jpg"></a></td>
25      <td><a href="images/D.jpg"></a></td>
26    </tr>
27  </table>
28</center>
29 </body>
30
31 </html>
32
33
34
```



India



Santo Domingo



Chicago



Fermilab





File Edit Search Preferences Shell Macro Windows

sendAudio.html

sendAudio.pl

```
<html>
<head>
</head>
```

```
<body>
```

```
<form>
<input>
</form>
```

```
</body>
```

```
</html>
```

File Edit S

sendAudio.html

!/usr/b

\$audioFi

print("C

open(FILE

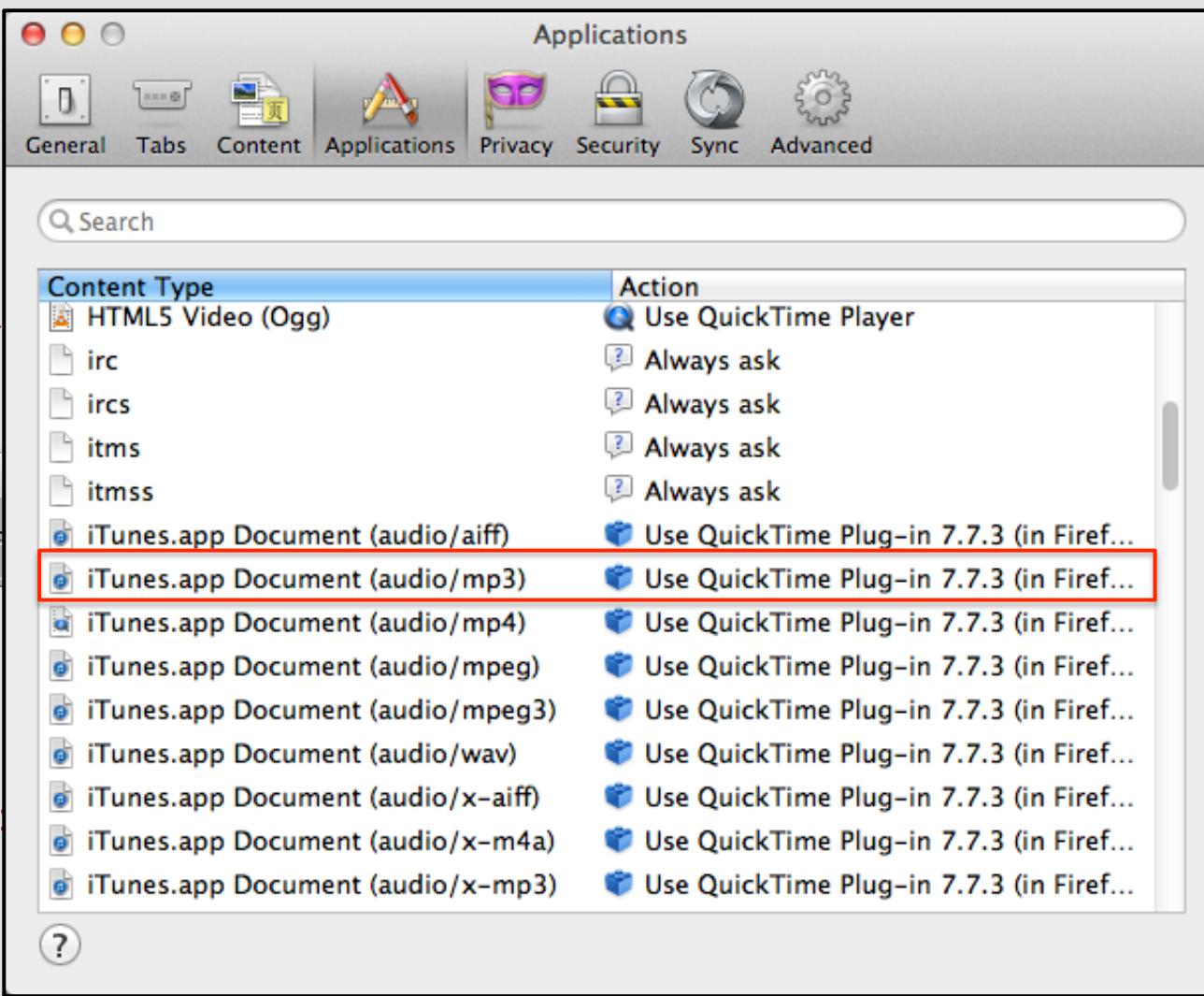
while(re

{

print("

}

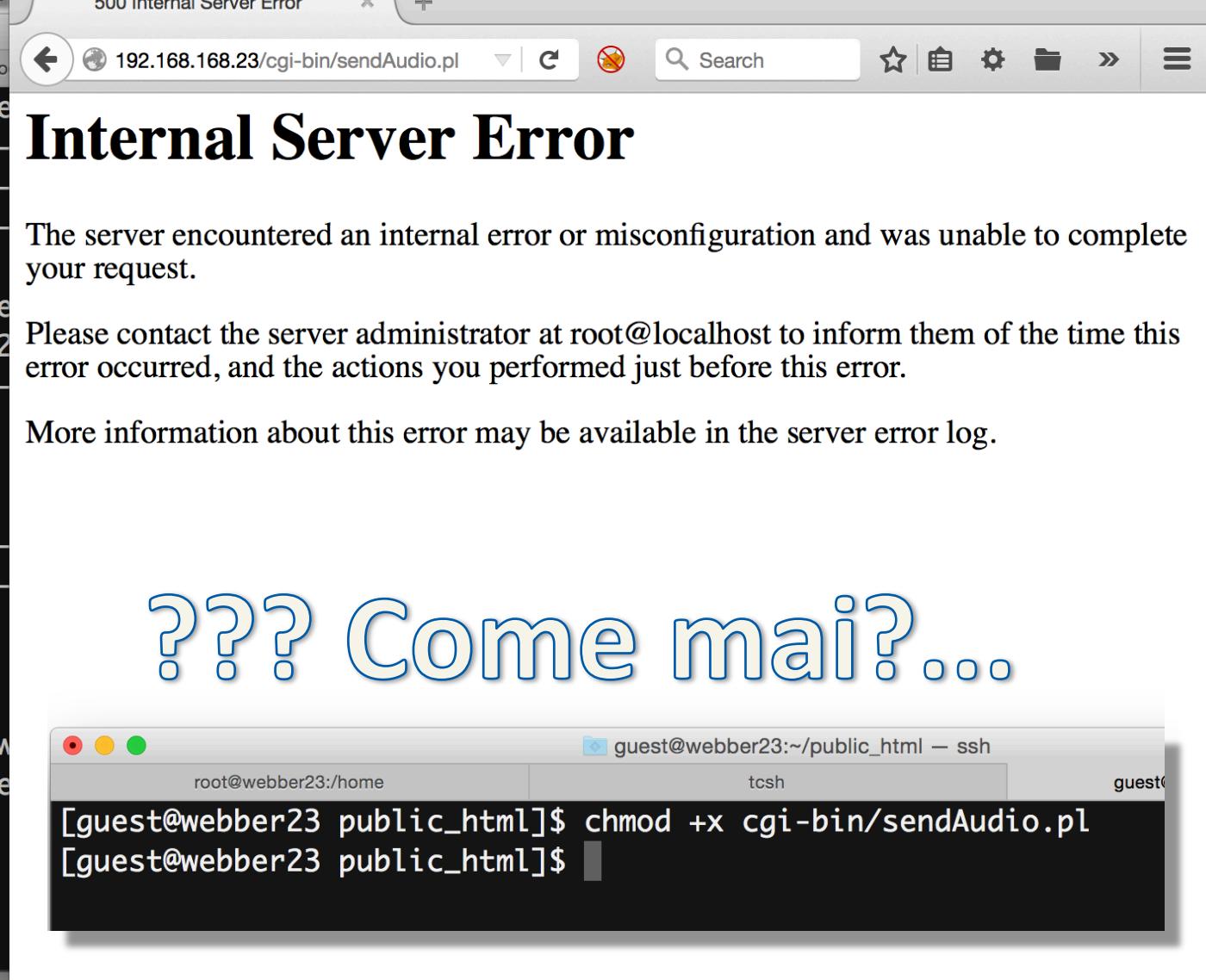
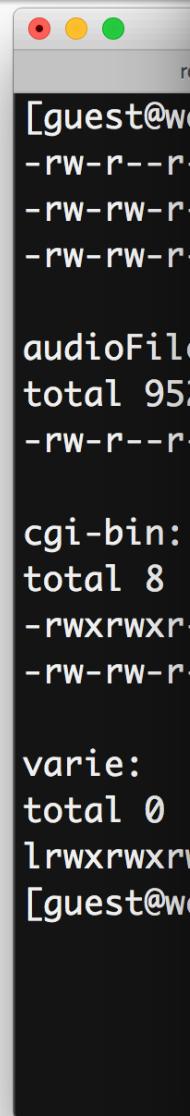
close(FILE) ;



= "GET">

i dati trasmessi
il browser è
invocare un
dati, il brano
gestendo
n lo stream.

\r\n\r\n");



Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.

Please contact the server administrator at root@localhost to inform them of the time this error occurred, and the actions you performed just before this error.

More information about this error may be available in the server error log.

??? Come mai?...

```
guest@webber23:~/public_html — ssh
root@webber23:/home          tcsh
[guest@webber23 public_html]$ chmod +x cgi-bin/sendAudio.pl
[guest@webber23 public_html]$
```

sendAudio.pl

sendAudio.pl 192.168.168.23/cgi-bin/sendAudio.pl Search

E se invece di farlo riprodurre direttamente dal browser volessimo permettere all'utente che lo richiede di salvarne una copia sul proprio Hard Disk?

0:06 1:01

NB: accendete temporaneamente l'audio per ascoltare...



sendAudio.html - /home/guest/public_html/

File Edit Search Preferences Shell Macro Windows Help

sendAudio.html sendAudio.pl

```
<html>

<head>
</head>

<body>

<form action="http://192.168.168.23/cgi-bin/sendAudio.pl" method="GET">
<input type="Submit" value="Play Bill Evans"/>
</form>

</body>

</html>
```

sendAudio.pl - /home/guest/public_html/cgi-bin/

File Edit Search Preferences Shell Macro Windows Help

sendAudio.html sendAudio.pl

```
#!/usr/bin/perl

$audioFile = "../audioFiles/billEvans.mp3" ;

print("Content-type: application/octet-stream; name=\"$audioFile\"\r\n\r\n");
print("Content-disposition: attachment; name=\"$audioFile\"\r\n\r\n");

open(FILE, "<$audioFile" ) or die "Cannot open $audioFile";

while(read(FILE, $buffer, 100) )
{
    print("$buffer") ;
}

close(FILE) ;
```

Mozilla Firefox

http://192.168.1.../sendAudio.html

192.168.168.23/~guest/sendAudio.html

Play Bill Evans

Opening sendAudio.pl

You have chosen to open:
 **sendAudio.pl**
which is: Binary File (951 KB)
from: <http://192.168.168.23>

Would you like to save this file?

Cancel Save File

File Edit Search Preferences Shell Macro Windows Help

pictures.html sendAudio.pl sendVideo.html sendVideo.pl /user/gr1/e831/menasce/public_html/esempi/ L: 20 C: 0

/user/gr1/e831/menasce/public_html/esempi/sendVideo.html byte 213 of 227

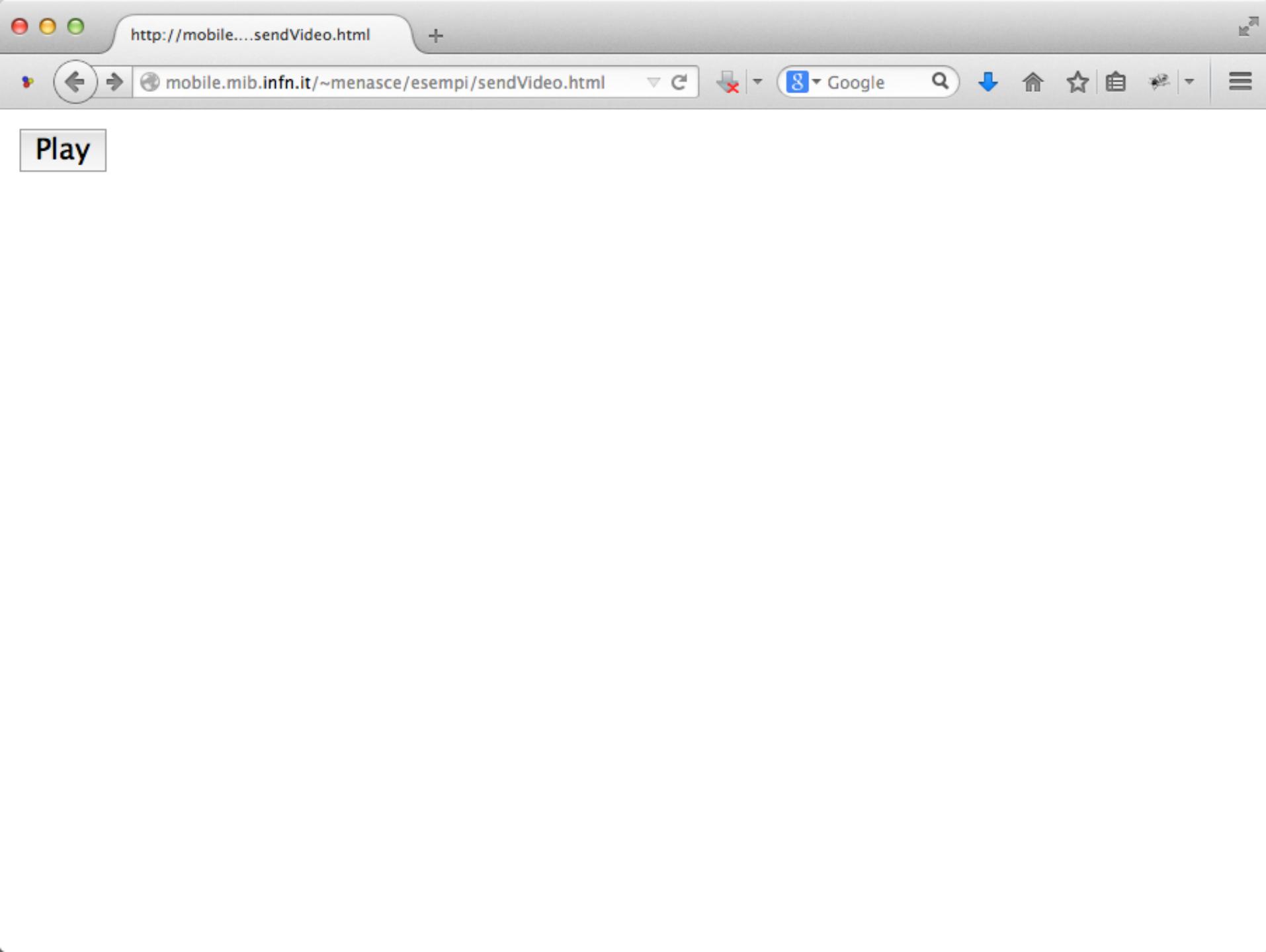
```
1 <html>
2
3   <head>
4     </head>
5
6   <body>
7     <form action="mailto:gr1@e831.menasce.it">
8       <input type="file" name="fileToUpload" />
9     </form>
10
11   </body>
12
13 </html>
14
15
16
17
18
```

Di nuovo, ciò che conta è l'opportuna direttiva al browser, nella forma di un MIME-TYPE.

Questo varia a seconda del tipo di file: possono gestiti tantissimi formati.

- `video/avi` : Covers most Windows-compatible formats including .avi and .divx^[15]
- `video/example`: example in documentation, Defined in [RFC 4735](#)
- `video/mpeg`: [MPEG-1](#) video with multiplexed audio; Defined in [RFC 2045](#) and [RFC 2046](#)
- `video/mp4`: [MP4](#) video; Defined in [RFC 4337](#)
- `video/ogg`: [Ogg Theora](#) or other video (with audio); Defined in [RFC 5334](#)
- `video/quicktime`: [QuickTime](#) video; Registered^[16]
- `video/webm`: [WebM](#) Matroska-based open media format
- `video/x-matroska`: [Matroska](#) open media format
- `video/x-ms-wmv`: [Windows Media Video](#); Documented in [Microsoft KB 288102](#)
- `video/x-flv`: [Flash video \(FLV files\)](#)

```
1 #!/usr/bin/perl
2
3 $fileName = "test.flv"
4
5 print("Content-Type:video/quicktime; name=\"$fileName\"\r\n\r\n");
6 # print("Content-Disposition: attachment; filename=\"$fileName\"\r\n\r\n");
7
8 open(FILE, "<$fileName");
9 while(read(FILE, $buffer, 100) )
10 {
11   print("$buffer");
12 }
13
```



Play

http://mobile....i/sendVideo.pl

+



mobile.mib.infn.it/cgi-bin/scripts/esempi/sendVideo.pl





```
1 <html>
2
3   <head>
4   </head>
5
6   <body>
7     GET method
8     <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl"
9       method="GET">
10    First Name: <input type="text" name="firstName">
11    <br>
12    Last Name : <input type="text" name="lastName" >
13    <input type="submit" value="Submit">
14  </form>
15
16
17  Analizziamo ora i due differenti metodi per fornire un input
18  ad un programma eseguito in modalità CGI (in questo caso
19  uno script perl). Iniziamo con il metodo GET, che abbiamo già
20  utilizzato.
21
22
23
24
25  </body>
26
27 </html>
28
```

Analizziamo ora i due differenti metodi per fornire un input ad un programma eseguito in modalità CGI (in questo caso uno script perl). Iniziamo con il metodo GET, che abbiamo già utilizzato.

three_1.html - /user/gr1/e831/menasce/public_html/esempi/

File Edit Search Preferences Shell Macro Windows Help

three_1.html three_1.pl

/user/gr1/e831/menasce/public_html/esempi/three_1.html byte 624 of 624 L: 28 C: 0

```
1 <html>
2
3   <head>
4   </head>
5
6   <body>
7     GET method
8     <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl"
9       method="GET">
10    First Name: <input type="text" name="firstName">
11    <br>
12    Last Name : <input type="text" name="lastName" >
13    <input type="submit" value="Submit">
14  </form>
15  <hr/>
16  POST method
17  <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl"
18    method="POST">
19    First Name: <input type="text" name="firstName">
20    <br>
21    Last Name : <input type="text" name="lastName" >
22    <input type="submit" value="Submit">
23  </form>
24
25  </body>
26
27 </html>
```

Nella stessa pagina, però, predisponiamo un secondo FORM, in tutto e per tutto identico al primo ma che utilizza invece il metodo POST.

Entrambi i FORM sono associati alla medesima ACTION, ossia l'esecuzione di un unico script in Perl: scopo dello script è quello di ricevere nome e cognome digitati in input dall'utente e manipolarli in qualche modo. Vediamo come è implementato lo script:

three_1.pl - /user/gr1/e831/menasce/public_html/esempi/

File Edit Search Preferences Shell Macro Windows Help

three_1.html three_1.pl /user/gr1/e831/menasce/public_html/esempi/three_1.pl byte 431 of 431 L: 24 C: 0

```
1 #!/usr/bin/perl
2
3 print <<EOB ;
4 Content-type: text/html
5
6
7 <pre>
8 REQUEST_METHOD: $ENV{'REQUEST_METHOD'}
9 EOB
10
11 Come sempre, la prima cosa che lo script deve produrre è uno header che
12 informi il client del tipo di dati che gli verrà fornito (in questo esempio, html).
13
14
15 Subito dopo lo script fornisce il valore della variabile di ambiente
16 REQUEST_METHOD (GET o POST a seconda del metodo scelto dall'utente)
17
18
19
20
21
22
23
24|
```

three_1.pl – /user/gr1/e831/menasce/public_html/esempi/

File Edit Search Preferences Shell Macro Windows Help

three_1.html three_1.pl /user/gr1/e831/menasce/public_html/esempi/three_1.pl byte 431 of 431 L: 24 C: 0

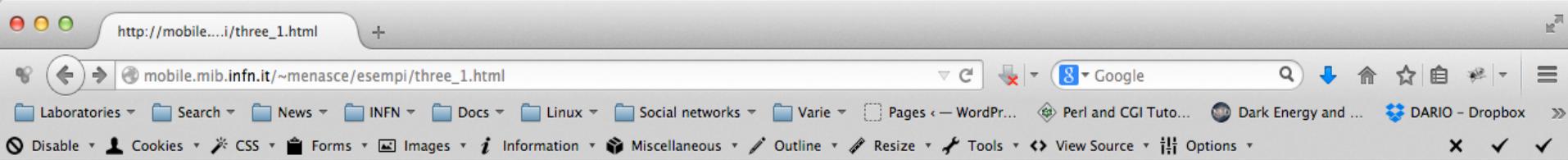
```
1 #!/usr/bin/perl
2
3 print <<EOB ;
4 Content-type: text/html
5
6
7 <pre>
8 REQUEST_METHOD: $ENV{'REQUEST_METHOD'}
9 EOB
10
11 if      ($ENV{'REQUEST_METHOD'} eq 'GET')
12 {
13     print("QUERY_STRING : $ENV{'QUERY_STRING'}\n" );
14 }
15
16
17
18
19
20
21
22
23
24
```

Nel caso in cui il metodo di trasferimento dei dati di input dal client al server fosse di tipo **GET**, questi verrebbero impacchettati in un'unica stringa e trasferiti al programma come valore di una variabile di ambiente chiamata **QUERY_STRING**.

La **QUERY_STRING** ha una limitazione di lunghezza di 2083 caratteri (ma questo numero varia nelle implementazioni dei browser). Inoltre non tutti i caratteri ANSI possono farne parte (occorre usare alcuni accorgimenti...)

```
1 #!/usr/bin/perl
2
3 print <<EOB ;
4 Content-type: text/html
5
6
7 <pre>
8 REQUEST_METHOD: $ENV{'REQUEST_METHOD'}
9 EOB
10
11 if      ($ENV{'REQUEST_METHOD'} eq 'GET')
12 {
13 print ("QUERY_STRING : $ENV{'QUERY_STRING'}\n" );
14 }
15 elsif   ($ENV{'REQUEST_METHOD'} eq 'POST')
16 {
17 read (STDIN, $buffer,  $ENV{'CONTENT_LENGTH'}) ;
18 print ("CONTENT_LENGTH: $ENV{'CONTENT_LENGTH'}\n") ;
19 print ("buffer        : $buffer\n" )
20 }
21 print("</pre>") ;
22
23
24 I Se invece il metodo di trasferimento dei dati di input dal client al server fosse
```

di tipo **POST**, questi verrebbero impacchettati in un'altra stringa e trasferiti al programma come STDIN al programma. Alla variabile di ambiente **CONTENT_LENGTH** viene assegnata la lunghezza dello stream di input in modo che la funzione perl read possa sapere quanti bytes leggere.



GET method

First Name:

Last Name :

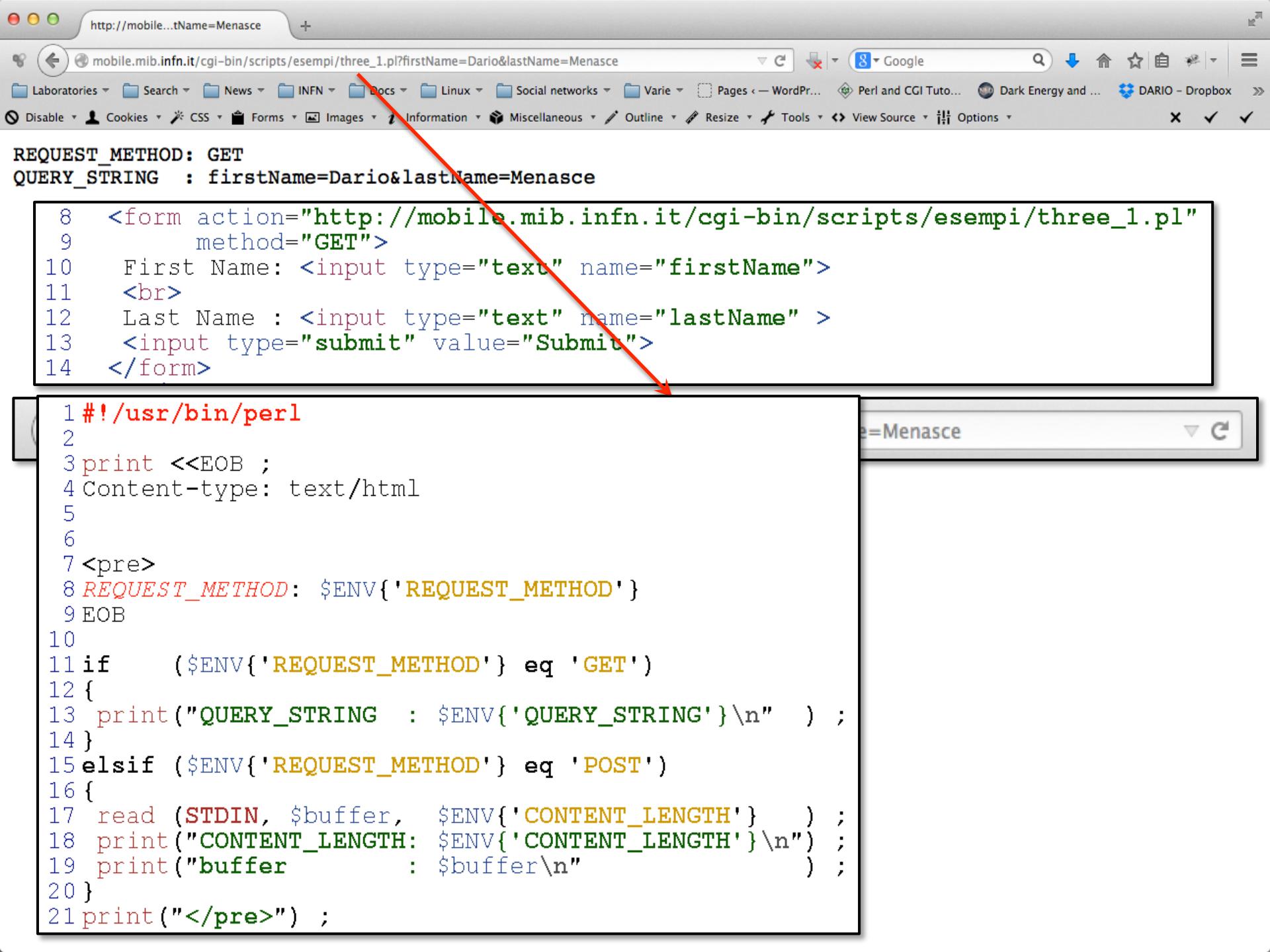
POST method

First Name:

Last Name :

Vediamo allora questa pagina web in azione:

mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl?firstName=Dario&lastName=Menasce

A screenshot of a web browser window. The address bar shows the URL "http://mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl?firstName=Dario&lastName=Menasce". The page content displays the form code from the previous block, with a red diagonal line crossing it out. Below the crossed-out code, the browser's status bar shows the query string "firstName=Dario&lastName=Menasce".

REQUEST_METHOD: GET
QUERY_STRING : firstName=Dario&lastName=Menasce

```
8 <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl"
9     method="GET">
10    First Name: <input type="text" name="firstName">
11    <br>
12    Last Name : <input type="text" name="lastName" >
13    <input type="submit" value="Submit">
14 </form>
```

```
1#!/usr/bin/perl
2
3print <<EOB ;
4Content-type: text/html
5
6
7<pre>
8REQUEST_METHOD: $ENV{'REQUEST_METHOD'}
9EOB
10
11if      ($ENV{'REQUEST_METHOD'} eq 'GET')
12{
13 print("QUERY_STRING : $ENV{'QUERY_STRING'}\n" );
14 }
15elsif  ($ENV{'REQUEST_METHOD'} eq 'POST')
16{
17 read (STDIN, $buffer, $ENV{'CONTENT_LENGTH'}) ;
18 print("CONTENT_LENGTH: $ENV{'CONTENT_LENGTH'}\n");
19 print("buffer          : $buffer\n");
20 }
21print("</pre>" );
```

e=Menasce

GET method

First Name:

Last Name :

POST method

First Name:

Last Name :

REQUEST_METHOD: POST
CONTENT_LENGTH: 34
buffer : firstName=Francesco&lastName=Prelz

mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl

```
17 <form action="http://mobile.mib.infn.it/cgi-bin/scripts/esempi/three_1.pl"
18     method="POST">
19     First Name: <input type="text" name="firstName">
20     <br>
21     Last Name : <input type="text" name="lastName" >
22     <input type="submit" value="Submit">
23 </form>
```

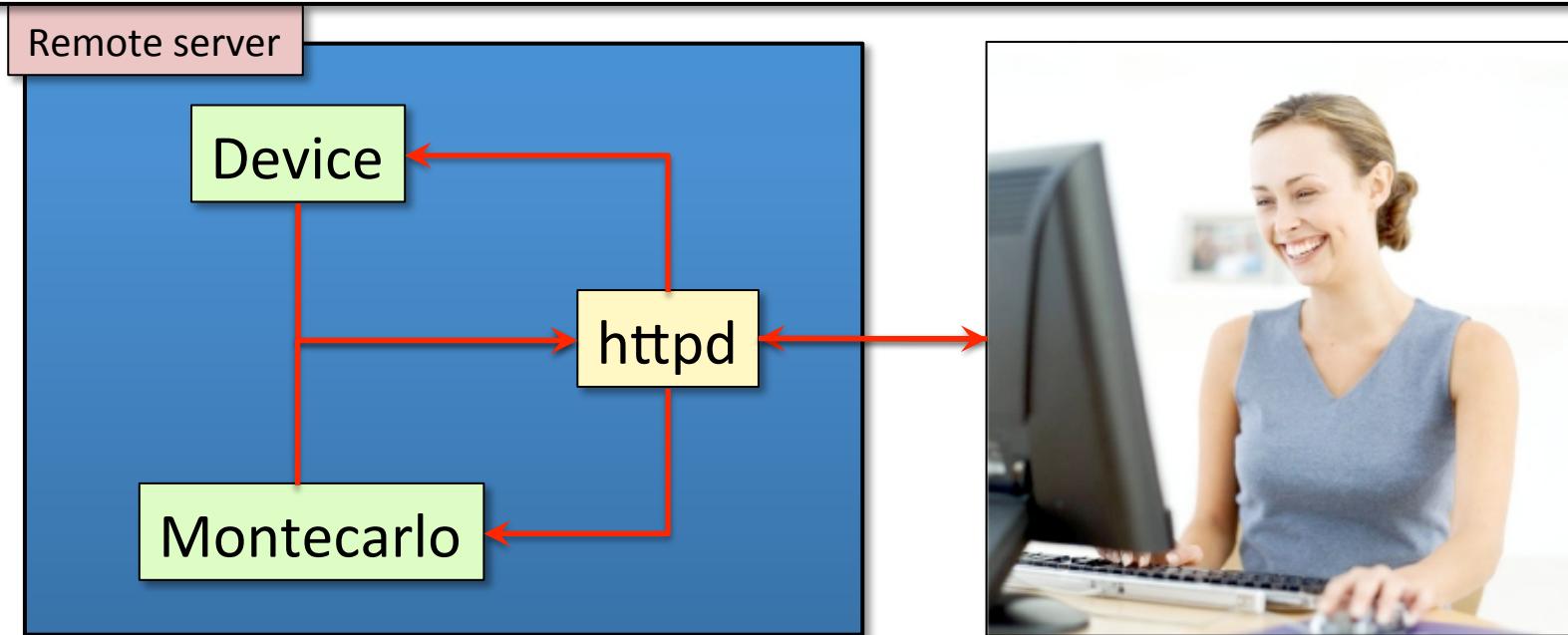
```
11 if      ($ENV{'REQUEST_METHOD'} eq 'GET')
12 {
13 print("QUERY_STRING  : $ENV{'QUERY_STRING'}\n"  );
14 }
15 elsif   ($ENV{'REQUEST_METHOD'} eq 'POST')
16 {
17 read (STDIN, $buffer,  $ENV{'CONTENT_LENGTH'}) ;
18 print("CONTENT_LENGTH: $ENV{'CONTENT_LENGTH'}\n");
19 print("buffer        : $buffer\n" );
20 }
```

Abbiamo quindi visto come utilizzare il CGI per invocare l'esecuzione di un programma remoto il cui output venga spedito indietro al browser per essere rappresentato nella *viewport* (dove stanno i controlli per l'utente).

Gli esempi visti, mediante l'uso di un FORM, prevedono però un comportamento statico: ogni nuovo output viene rappresentato in una nuova finestra (o tab) aperti dal browser ad hoc (oppure rimpiazzano il contenuto precedente).

Esaminiamo ora i possibili meccanismi per rendere invece *dinamica* una pagina web, capace cioè di fornire contenuti che vengono usati dal browser per modificare solo una parte della pagina web di partenza.

Supponiamo allora di aver bisogno di rendere interattiva una pagina Web, nel senso che il suo contenuto, o solo una parte di esso, cambi in funzione di azioni intraprese dall'utente e che i nuovi contenuti siano generati da un programma ad-hoc che agisca dal lato server (ad esempio un simulatore **MonteCarlo** o uno strumento, un **device**, capaci di generare dati numerici, alfanumerici ma anche audio e video).



Ciò che rende possibile questo meccanismo è il protocollo CGI, mediante il quale il *daemon server* *httpd* genera gli opportuni processi capaci di comunicare con i *device* o i programmi

Il file di configurazione

httpd.conf (modified) - /etc/httpd/conf/

File Edit Search Preferences Shell Macro Windows Help

/etc/httpd/conf/httpd.conf 34437 bytes L: 576 C: 0

```

#
# ScriptAlias: This controls which directories contain server scripts.
# ScriptAliases are essentially the same as Aliases, except that
# documents in the realmname directory are treated as applications and
# run by the server when requested rather than as documents sent to the client.
# The same rules about trailing "/" apply to ScriptAlias directives as to
# Alias.
#
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"

#
# "/var/www/cgi-bin" should be changed to whatever your
# CGI directory exists, if you have that config
#
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None FollowSymLinks
    Order allow,deny
    Allow from all
</Directory>

#
# Redirect allows you to tell clients about

```

Nel file di configurazione, [httpd.conf](#), dovremo specificare una key-word, che, usata nello URL di chiamata, indichi al server che ciò che si richiede nello URL non è un documento ma l'**esecuzione** di un programma.

La keyword punta, inoltre, al luogo in cui sono contenuti i programmi eseguibili da httpd.

```
[root@mobile conf]# ll /var/www/cgi-bin/
total 0
lrwxrwxrwx. 1 root root 35 Apr  9 10:06 scripts -> /user/gr1/e831/menasce/public_html/
[root@mobile conf]#
```

Il file di configurazione

Ma come funziona? Se uno user digita il seguente URL:

<http://mobile.mib.infn.it/cgi-bin/scripts/job.pl>

Il daemon http che lo riceve individua la keyword **cgi-bin** e, usando il re-indirizzamento definito nel file di configurazione, cerca un chiamato **job.pl** nella directory

/user/gr1/e831/menasce/public_html/job.pl

```
ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"

#
# "/var/www/cgi-bin" should be changed to
# CGI directory exists, if you have that
#
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None FollowSymLinks
    Order allow,deny
    Allow from all
</Directory>
```

Se l'eseguibile (od uno script) viene trovato in quella area e le condizioni di accesso definite qui di fianco risultano soddisfatte, il programma viene eseguito come un sottoprocesso di **httpd** stesso, e quindi con i privilegi da esso posseduti (**user apache, group apache** e tutti i privilegi definiti in questo file di configurazione per il *server daemon*)

```
[root@mobile conf]# ll /var/www/cgi-bin/
total 0
lrwxrwxrwx. 1 root root 35 Apr  9 10:06 scripts -> /user/gr1/e831/menasce/public_html/
[root@mobile conf]#
```

Un esempio di CGI

Ma come comunica il programma `job.pl` con la pagina web dalla quale l'utente ne ha richiesto l'esecuzione? Anzitutto vediamo cosa `job.pl` sia supposto fare e poi come si fa a lanciarne l'esecuzione da una pagina web.

Supponiamo di aver la necessità di costruire un'interfaccia web ad un *file system* (ad esempio ad una directory) e di voler identificare tutti i *file* ivi posizionati che contengano una certa stringa (selezionabile dall'utente tramite un apposito *form* sulla pagina stessa).

Supponiamo poi di voler rappresentare sulla pagina web i nomi dei file trovati, con i numeri di riga delle occorrenze della stringa richiesta trovata nel file (rappresentati in verde) e uno hyperlink al file medesimo, in modo da poterlo invocare direttamente con un click.

Una pagina il cui aspetto sia sostanzialmente il seguente:

mobile.mib.infn.it/~menasce/esempi/finder.html

Google

Search string: Submit Query

Search results

I campi della viewport

finder.html - /user/gr1/e831/menasce/public_html/esempi/

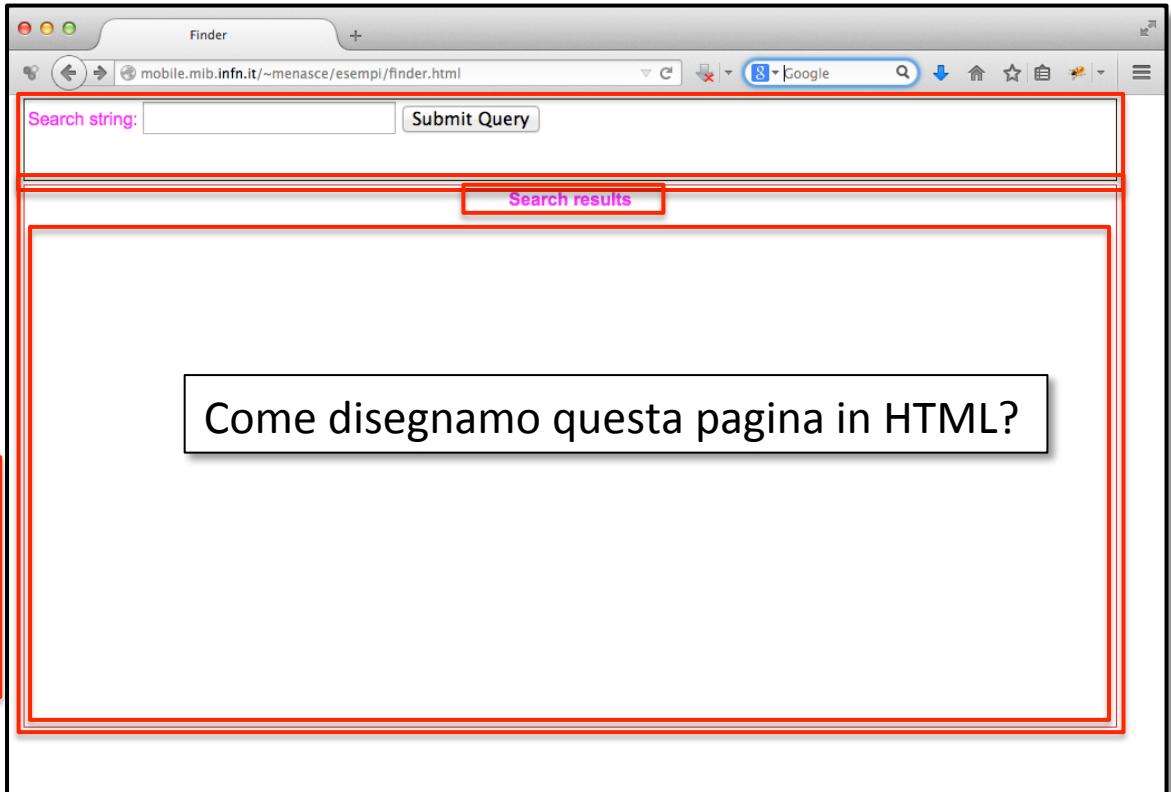
File Edit Search Preferences Shell Macro Windows Help

job.pl finder.html L: 110 C: 55

```

1 <html>
2   <head>
3     <title>Finder</title>
102  </head>
103
104  <body>
105    <div id="formArea">
106
107
108
109
110
111
112    </div>
113    <div id="outputArea">
114      <div id="outputAreaHead">
115        Search results
116      </div>
117      <div id="outputText">
118        </div>
119      </div>
120    </body>
121
122 </html>
123

```



Come disegniamo questa pagina in HTML?

I corrispondenti CSS elements



```

1 <html>
2   <head>
3     <title>Finder</title>
4     <style type="text/css">
5       #formArea {
6         position : absolute ;
7         display  : block ;
8         top      : 2px ;
9         left     : 10px ;
10        width   : 730px ;
11        height   : 50px ;
12        position : fixed ;
13        border   : 1px solid #231 ;
14        padding  : 2px ;
15        text-align: left ;
16        font-family: arial ;
17        color    : #f0f ;
18      }
19      #outputArea {
20        position : absolute ;
21        display  : block ;
22        top      : 60px ;
23        left     : 10px ;
24        width   : 730px ;
25        height   : 360px ;
26        position : fixed ;
27        border   : 1px solid #e33 ;
28        padding  : 2px ;
29        text-align: left ;
30        font-family: arial ;
31        color    : #00f ;
32      }
33      #outputAreaHead{
34        text-align : center ;
35        font-weight: bold ;
36        font-family: arial ;
37        color     : #f3f ;
38      }
39   </style>

```



Search string: Submit Query

Search results

```

105  <div id="formArea">
106
107
108
109
110
111
112  </div>
113  <div id="outputArea">
114    <div id="outputAreaHead">
115      Search results
116    </div>
117    <div id="outputText">
118    </div>
119  </div>

```

Il form di input

```

104 <body>
105   <div id="formArea">
106     <form>
107       Search string: <input id="searchString"
108         type="text"
109         name="searchString">
110       <input type="submit" onclick="callServer.launch()"/>
111     </form>
112   </div>
113   <div id="outputArea">
114     <div id="outputAreaHead">
115       Search results
116     </div>
117     <div id="outputText">
118     </div>
119   </div>
120 </body>

```

Search string: Submit Query

Il punto di partenza della azione è la dichiarativa **onclick**. Al premere del bottone, l'utente invoca il metodo **launch** della classe **CallServer** (di cui **callServer** è una istanza concreta)

Definiamo le chiamate a opportune funzioni in JavaScript

```

41 <script>
42   function CallServer()
43   {
44     this.xhr_object;
45     this.server_response,
46
47     this.createXMLHttpRequest = XMLHttpRequest;
48     this.sendDataToServer = sendXMLHttpRequest;
49     this.displayAnswer = displayAnswer;
50     this.launch = launch;
51   }

```

```

104 <body>
105   <div id="formArea">
106     <form>
107       Search string: <input id="searchString"
108           type="text"
109           name="searchString">
110       <input type="submit" onclick="callServer.launch()" />
111     </form>
112   </div>
113   <div id="outputArea">
114     <div id="outputAreaHead">
115       Search results
116     </div>
117     <div id="outputText">
118     </div>
119   </div>
120 </body>

```

Search string: Submit Query

Il blocco **<script>**, che appare nello **<head>** dell'**<html>** viene eseguito subito, prima che la pagina venga caricata in memoria e resa graficamente dal *rendering engine* del browser

```

<script>
...
var callServer = new CallServer();
callServer.createXMLHttpRequest();
</script>

```

In questo blocco viene istanziata una versione concreta di un oggetto della classe **CallServer** e, subito dopo, viene creata una richiesta **XMLHTTP** (che verrà popolata in seguito ed attivata poi dal metodo **launch**, che contatterà il server quando l'utente premerà il tasto **Submit**

```

53 function createXMLHttpRequest()
54 {
55   this.xhr_object = null;
56
57   if(window.XMLHttpRequest) ←
58   {
59     this.xhr_object = new XMLHttpRequest();
60   }
61   else if(window.ActiveXObject)
62   {
63     this.xhr_object = new ActiveXObject("Microsoft.XMLHTTP");
64   }
65   else
66   {
67     alert("Your browser doesn't provide XMLHttpRequest functionality");
68     return;
69   }
70 }
```

Qui si tiene conto delle significative differenze tra i browser nel gestire gli oggetti capaci di trasmettere richieste Ajax

```

106 <div id="formArea">
107   <form>
108     Search string: <input id="searchString"
109       type="text"
110       name="searchString">
111     <input type="submit" onclick="callServer.launch()"/>
112   </form>
113 </div>

```

Search string:

```

92 function launch ()
93 {
94   var ss = document.getElementById("searchString").value;
95   this.sendDataToServer("search="+ss);
96   this.displayAnswer();
97 }

```

```

72 function sendDataToServer (dataToSend)
73 {
74   var xhr_object = this.xhr_object;
75   xhr_object.open("POST",
76                 "http://mobile.mib.infn.it/cgi-bin/scripts/esempi/job.pl",
77                 false);
78   xhr_object.setRequestHeader("Content-type",
79                             "application/x-www-form-urlencoded");
80   xhr_object.send(dataToSend);
81   if(xhr_object.readyState == 4)
82   {
83     this.server_response = xhr_object.responseText;
84   }
85 }

```

Torniamo al bottone submit
e seguiamo il percorso della
richiesta al server:

Apriamo il canale di
comunicazione col server

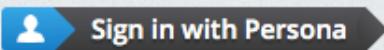
Trasferiamo il membro della
classe in una variabile locale

Dov'è la documentazione disponibile
per la classe **XMLHttpRequest**, di cui
xhr_object è un'istanza concreta?

XMLHttpRequest – Web AP... +

Laboratories Search News INFN Docs Linux Social networks Varie Pages — WordPress Incorrect rendering

Help us improve the quality of our code samples by answering this brief survey: <http://bit.ly/sample-demo>

 mozilla

 ZONES WEB PLATFORM TOOLS DEMOS CONNECT

MDN > Web technology for developers > Web API Interfaces > XMLHttpRequest

LANGUAGES EDIT 

XMLHttpRequest

by 85 contributors:  Show all...

XMLHttpRequest is a [JavaScript](#) object that was designed by Microsoft and adopted by Mozilla, Apple, and Google. It's now being [standardized in the W3C](#). It provides an easy way to retrieve data from a URL without having to do a full page refresh. A Web page can update just a part of the page without disrupting what the user is doing. XMLHttpRequest is used heavily in [AJAX](#) programming.

Despite its name, XMLHttpRequest can be used to retrieve any type of data, not just XML, and it supports protocols other than [HTTP](#) (including [file](#) and [ftp](#)).

To create an instance of XMLHttpRequest, simply do this:

```
var myRequest = new XMLHttpRequest();
```

IN THIS ARTICLE

[Method overview](#)

[Properties](#)

[Non-standard properties](#)

[Constructor](#)

[XMLHttpRequest\(\)](#)

[Methods](#)

[abort\(\)](#)

[getAllResponseHeaders\(\)](#)

[getResponseHeader\(\)](#)



Laboratories

Search

News

INFN

Docs

Linux

Social networks

Varie

Pages ← WordPr...

Incorrect renderi...



For details about how to use XMLHttpRequest, see [Using XMLHttpRequest](#).

Method overview

```

XMLHttpRequest (JSObject objParameters);

void abort();

DOMString getAllResponseHeaders();

DOMString? getResponseHeader(DOMString header);

void open(DOMString method, DOMString url, optional boolean async, optional
DOMString? user, optional DOMString? password);

void overrideMimeType(DOMString mime);

void send();
void send(ArrayBuffer data);
void send(ArrayBufferView data);
void send(Blob data);
void send(Document data);
void send(DOMString? data);
void send(FormData data);

void setRequestHeader(DOMString header, DOMString value);

```

Non-standard methods

```
[noscript] void init(in nsIPrincipal principal, in nsIScriptContext scriptContext,
in nsPIDOMWindow ownerWindow);
```

IN THIS ARTICLE

[Method overview](#)

[Properties](#)

[Non-standard properties](#)

[Constructor](#)

[XMLHttpRequest\(\)](#)

[Methods](#)

[abort\(\)](#)

[getAllResponseHeaders\(\)](#)

[getResponseHeader\(\)](#)

[open\(\)](#)

[overrideMimeType\(\)](#)

[send\(\)](#)

[setRequestHeader\(\)](#)

[Non-standard methods](#)

[init\(\)](#)

[openRequest\(\)](#)

[Requires Gecko 1.9
\(Firefox 3\)](#)
[sendAsBinary\(\)](#)

[Notes](#)

[Events](#)

open()

Initializes a request. This method is to be used from JavaScript code; to initialize a request from native code, use `openRequest()` instead.

Note: Calling this method for an already active request (one for which `open()` or `openRequest()` has already been called) is the equivalent of calling `abort()`.

```
void open(  
          DOMString method,
          DOMString url,
          optional boolean async,
          optional DOMString user,
          optional DOMString password
        );
```

Parameters

method

The HTTP method to use, such as "GET", "POST", "PUT", "DELETE", etc. Ignored for non-HTTP(S) URLs.

url

The URL to send the request to.

async

An optional boolean parameter, defaulting to `true`, indicating whether or not to perform the operation asynchronously. If this value is `false`, the `send()` method does not return until the response is received. If `true`, notification of a completed transaction is provided using event listeners. This *must* be true if the

IN THIS ARTICLE

[Method overview](#)

[Properties](#)

[Non-standard properties](#)

[Constructor](#)

Scegliamo il metodo di comunicazione (**GET o POST**).
Di questo parametro discuteremo in seguito

Specifichiamo lo URL dove ci sarà lo script in attesa

Indichiamo con `false` che ci aspettiamo un comportamento sincrono (il metodo `open` ritorna solo quando lo script ha terminato). Questo sistema in genere non va usato perché blocca (congela) il browser potenzialmente per tempi anche lunghi.

[init\(\)](#)

[openRequest\(\)](#)

[Requires Gecko 1.9
\(Firefox 3\)](#)
[sendAsBinary\(\)](#)

[Notes](#)

[Events](#)

```

72     function sendDataToServer (dataToSend)
73     {
74         var xhr_object = this.xhr_object;
75         xhr_object.open("POST",
76                         "http://mobile.mib.infn.it/cgi-bin/scripts/esempi/job.pl",
77                         false);
78         xhr_object.setRequestHeader("Content-type",
79                                     "application/x-www-form-urlencoded");
80         xhr_object.send(dataToSend);
81         if(xhr_object.readyState == 4)
82         {
83             this.server_response = xhr_object.responseText;
84         }
85     }

```

Poiché abbiamo richiesto una transazione sincrona, dobbiamo attendere la risposta!...

```

92     function launch ()
93     {
94         var ss = document.getElementById("searchString").value;
95         this.sendDataToServer("search="+ss);
96         this.displayAnswer();
97     }

```

```

87     function displayAnswer ()
88     {
89         document.getElementById("outputText").innerHTML = this.server_response;
90     }

```

```

114     <div id="outputArea">
115         <div id="outputAreaHead">
116             Search results
117         </div>
118         <div id="outputText">
119             </div>
120         </div>

```

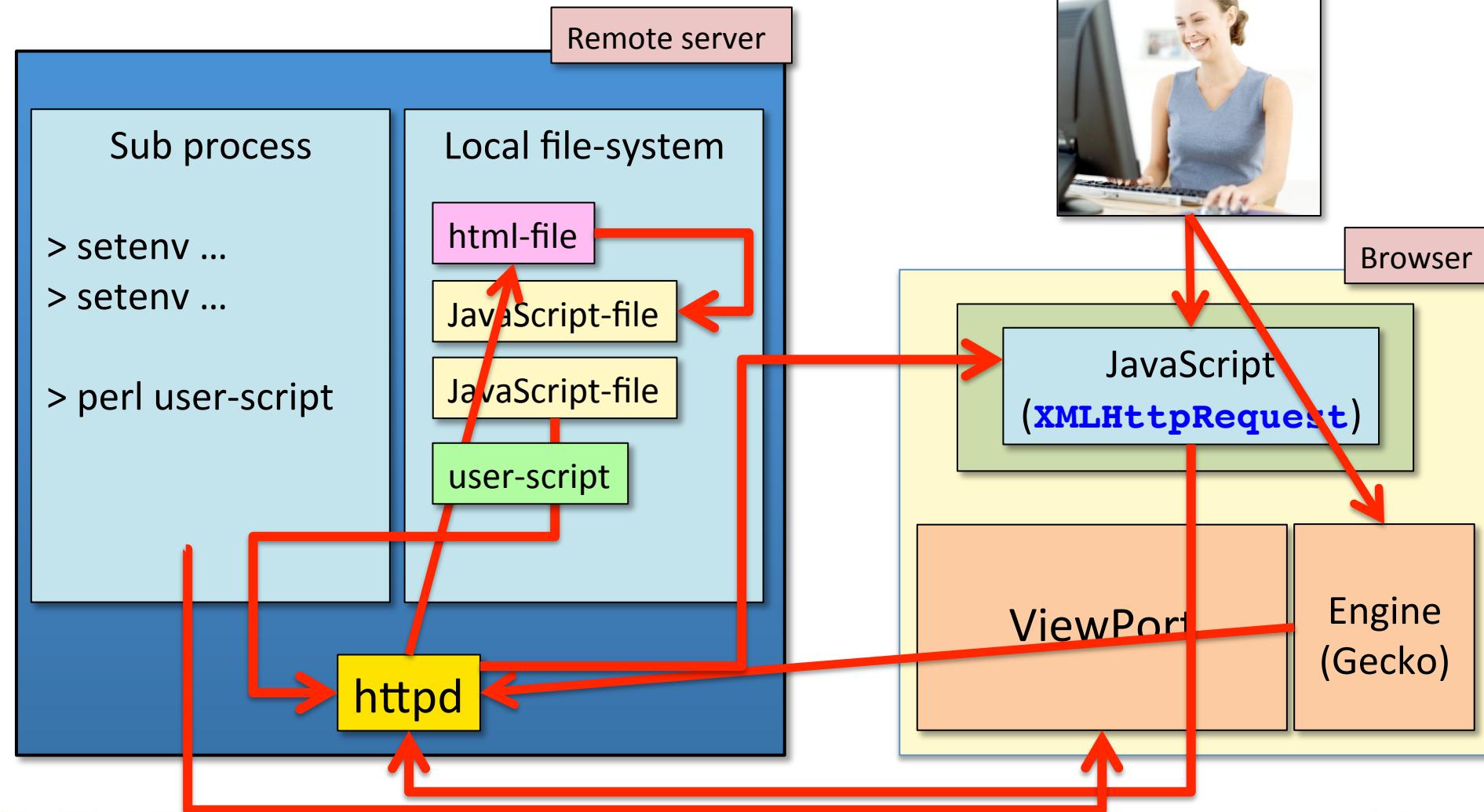
Dobbiamo ora descrivere il programma che esegua effettivamente la ricerca della stringa richiesta dall'utente e produca il risultato da esporre sulla pagina web. Si tratta nel nostro caso di uno *script* codificato in *Perl*, un popolare *scripting language*, molto adatto ad affrontare problemi di questo tipo.

NB: Il codice, eseguito per noi da Apache, può essere sia un programma compilato (in qualsiasi linguaggio) che un qualsiasi script, da shell a Python, a Perl.

Esaminiamo anzitutto il canale di comunicazione dello script nei confronti di Apache:

- L'input (la *search string*) gli verrà fornito tramite l'uso di un certo numero di variabili di ambiente, dal nome standard, che Apache definirà nella shell del sotto-processo che creerà (mediante fork) e nel quale verrà eseguito lo script.
- L'output dello script (STDOUT e STDERR) verrà recuperato ed assorbito da Apache e convogliato verso il client (JavaScript) che ne farà l'uso più opportuno.

Parte la richiesta per una pagina web contenente codice in JavaScript per l'esecuzione di un programma remoto:



Vediamo ora come è fatto il programma eseguibile, lato server, che, una volta attivato con la richiesta **XMLHttpRequest**, fornisca l'output richiesto.

- Si tratta, nell'esempio che si propone in questo corso, di uno script in **Perl**, ma un programma in qualunque altro linguaggio di programmazione potrebbe essere adatto allo scopo, sia compilato che interpretato.
- Per Perl lo è particolarmente poiché il massimo della sua evoluzione storica è avvenuto in concomitanza con lo sviluppo del WWW ed è quindi stato dotato di una ricca serie di moduli adatti a questo ambiente.

httpd

ViewPort

Engine
(Gecko)

```
1 #!/usr/bin/perl
```

```
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6
7
8
9
```

Lo shebang: dichiarativa al program loader per comunicare il path assoluto necessario a localizzare l'interpreter capace di eseguire il codice contenuto nelle righe seguenti

che è un sottoprocesso generato da Apache mediante un fork. (Dettagli in seguito).

```
10 Esegue la ricerca della stringa richiesta e produce il risultato: qui ci va il grosso delle
11 istruzioni di questo script
12
```

```
13
14     32 =====
15     33 sub header
16     34 {
17     35     print ("Content-type: text/html\n\n") ;
18     36 }
19     37
20     38 =====
```

```
21 Il preciso formato di questo header è definito dal protocollo dei MIME-TYPES: per
22 informazioni su questo ed altri dettagli relativi al CGI Programming si veda:
23
24     http://httpd.apache.org/docs/current/howto/cgi.html
25
```

```
26 Lasciamo per ora in sospeso l'analisi della funzione decodeQS, e procediamo ad
27 esaminare il programma principale
28
```

```
1 #!/usr/bin/perl  
2  
3             &header () ;  
4 my %formdata = &decodeQS() ;  
5
```

6 Se dovessimo cercare tutte le occorrenze di una stringa (ad es. **doc**) nei file
7 contenuti in una certa directory e ci potessimo accontentare di una semplice
8 sessione interattiva (**ssh**), eseguiremmo un comando di questo tipo:
9

```
10  
11 > find . -name \* -exec egrep -n doc {} \; -print
```

```
12  
13 [menasce@mobile esempi]$ find . -name \* -exec egrep -n doc {} \; -print  
14 9: This document refers to a  
15 ./two_1.html
```

```
16  
17 92: document.getElementById("outputText").innerHTML = t;  
18 97: var ss = document.getElementById("searchString").value;  
19 ./finder.html  
20 36: var divMenu = document.getElementById("menu") ;  
21 ./one_4.html  
22 36: var divMenu = document.getElementById("menu") ;  
23 ./one_5.html
```

24
25 Come sappiamo, la formattazione di questo tipo di output lascia molto a
26 desiderare come leggibilità. Inoltre non possiamo fare un click su un file e
27 accedere immediatamente al listato, non possiamo '*navigare*' nell'output.
28

File Edit Search Preferences Shell Macro Windows Help

/user/gr1/e831/menasce/public_html/esempi/ L: 35 C: 16

```
1 #!/usr/bin/perl
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6
7
8
9 my $cmd      = "find . -name \"*"
10          " -exec egrep -n $searchString {} \";
11          " -print | ";
```

Prepariamo una variabile, una stringa di caratteri, col comando che vorremo eseguire con la **searchString** espressa a sua volta come stringa (vederemo dopo come passarne il valore dall'utente al codice).

File Edit Search Preferences Shell Macro Windows Help

/user/gr1/e831/menasce/public_html/esempi/ L: 35 C: 16

```
1 #!/usr/bin/perl
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6
7
8
9 my $cmd      = "find . -name \\\/* "
10          " -exec egrep -n $searchString {} \\;" :
11          " -print | " ;
12
13
14
15
16 open(PIPE,"$cmd") ;
17
18
19
20
21
22
23
24
25
26
27
28
```

Tramite il comando open, apriamo una **pipe**, una conduttura che permette la comunicazione tra due processi. In questo caso permettiamo all'output del processo **find** di essere ricevuto dal processo corrente (**perl**) per essere ulteriormente processato.

```
1 #!/usr/bin/perl
2 [menasce@mobile esempi]$ find . -name \* -exec egrep -n doc {} \; -print
3 9: This document refers to a\n
4 ./two_1.html \n
5 92:     document.getElementById("outputText").innerHTML = t;\n
6 97:     var ss = document.getElementById("searchString").value;\n
7 ./finder.html \n
8 36:     var divMenu = document.getElementById("menu") ;\n
9 ./one_4.html \n
10 36:     var divMenu = document.getElementById("menu") ;\n
11 ./one_5.html \n
12
13
14
15
16 open(PIPE,"$cmd") ;
17
18
19 while(<PIPE>)
20 {
21
22
23
24
25
26
27 }
28 }
```

Gli operatori `<` e `>` servono per catturare ogni linea di output prodotta da **find** e trasferirla poi nella variabile di default di **perl**, `$_` (che, essendo sempre disponibile, non è necessario evidenziare in maniera esplicita come, ad esempio, `$_ = <pipe>`)

X job.pl - /user/gr1/e831/menasce/public_html/esempi/

File Edit Search Preferences Shell Macro Windows Help

/user/gr1/e831/menasce/public_html/esempi/job.pl byte 798 of 1811 L: 35 C: 16

```
1 #!/usr/bin/perl
2 [menasce@mobile esempi]$ find . -name \* -exec egrep -n doc {} \; -print
3 9: This document refers to a\n
4 ./two_1.html \n
5 92:     document.getElementById("outputText").innerHTML = t;\n
6 97:     var ss = document.getElementById("searchString").value;\n
7 ./finder.html \n
8 36:     var divMenu = document.getElementById("menu") ;\n
9 ./one_4.html \n
10 36:     var divMenu = document.getElementById("menu") ;\n
11 ./one_5.html \n
12
13
14
15
16 open(PIPE,"$cmd") ;
17
18
19 while(<PIPE>)
20 {
21     chomp() ; Eliminiamo i caratteri \n da ogni riga
22
23
24
25
26
27 }
28
```

Vogliamo poi trasmettere questo output al browser affinchè lo rappresenti graficamente, con gli opportuni hyperlinks ai corrispondenti file e con la stringa di **match** colorata.

Eliminiamo i caratteri \n da ogni riga

Search string: doc

Submit Query

Search results

9: This document refers to a

./two_1.html

92: document.getElementById("outputText").innerHTML = t;

97: var ss = document.getElementById("searchString").value;

./finder.html

36: var divMenu = document.getElementById("menu") ;

./one_4.html

```
[menasce@mobile esempi]$ find . -name \* -exec egrep -n doc {} \; -print
9: This document refers to a
./two_1.html
92: document.getElementById("outputText").innerHTML = t;
97: var ss = document.getElementById("searchString").value;
./finder.html
36: var divMenu = document.getElementById("menu") ;
./one_4.html
36: var divMenu = document.getElementById("menu") ;
./one_5.html
```

```
1 #!/usr/bin/perl
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6 my $URL          = "http://mobile.mib.infn.it/~menasce/esempi" ;
7
8 my $searchString = $formdata{'search'} ;
9 my $cmd           = "find . -name \\\\"*"
10                "-exec egrep -n $searchString {} \\;" :
11                " -print |" ;
12
13 open(PIPE,"$cmd") ;
14
15
16 while(<PIPE>)
17 {
18
19 if( !(m/\w+/)) {next ;}
20 s|^(\d+)|<font color='green'><b>$1</b></font>| ;
21
22
23 }
24 }
25
26
27
28
29 #####
```

9:

This document refers to a

9:

This document refers to a

```
1 #!/usr/bin/perl
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6 my $URL          = "http://mobile.mib.infn.it/~menasce/esempi" ;
7
8 my $searchString = $formdata{'search'} ;
9 my $cmd           = "find . -name \\\* "
10                "-exec egrep -n $searchString {} \\;" :
11                " -print | " ;
12
13 open(PIPE,"$cmd") ;
14
15
16 while(<PIPE>)
17 {
18     chomp() ;
19
20     s|^(\d+):|<font color='green'><b>$1</b></font>| ;
21     s|$searchString|<font color='red'>$searchString</font>| ;
22
23
24 }
25
26 Abbiamo accennato all'esistenza della variabile di default, $_: questo stesso
27 pezzo di codice potrebbe essere scritto così, anche se meno sinteticamente:
28
29 #####
```

```
1 #!/usr/bin/perl
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6 my $URL          = "http://mobile.mib.infn.it/~menasce/esempi" ;
7
8 my $searchString = $formdata{'search'} ;
9 my $cmd           = "find . -name \\\\"*"
10                "-exec egrep -n $searchString {} \\;" :
11                " -print |" ;
12
13 open(PIPE,"$cmd") ;
14
15
16 while($x = <PIPE>)
17 {
18     chomp($x) ;
19
20     $x =~ s|^(\d+):|<font color='green'><b>$1</b></font>| ;
21     $x =~ s|$searchString|<font color='red'>$searchString</font>| ;
22
23
24 }
25
26 O anche:
27
28
29 #####
```

```

1 #!/usr/bin/perl
2
3             &header () ;
4 my %formdata = &decodeQS() ;
5
6 my $URL          = "http://mobile.mib.infn.it/~menasce/esempi" ;
7
8 my $searchString = $formdata{'search'} ;
9 my $cmd           = "find . -name \\\* "
10                "-exec egrep -I -n $searchString {} \\;" ;
11                " -print |"
12
13 open(PIPE,"$cmd") ;
14
15 print("<pre>\n");

```

Nel caso dovessero esserci righe vuote prodotte da **find**, le eliminiamo tramite un'opportuna *regular-expression*

Spediamo l'output allo STDOUT, che, nel caso presente, è rediretto al canale di comunicazione istituito col browser cliente (un *socket* di rete bidirezionale aperto col server)

L'input ad uno script CGI

Avevamo lasciato in sospeso la descrizione del meccanismo usato per comunicare allo script, a sua volta attivato dal server, i parametri necessari alla sua esecuzione (si tratta, fondamentalmente nel nostro esempio, del valore della *search string*, **div**)

Esistono due modi per invocare uno script col protocollo CGI e passargli dei parametri in ingresso, il metodo GET ed il metodo POST. Vediamo prima come usare il metodo GET:

<http://mobile.mib.infn.it/cgi-bin/scripts/esempi/job.pl?searchString=doc&anotherPar=thisOne>

Protocollo	URL tradizionale con la richiesta del server nel file di configurazione	Nome del programma richiesto dal server nel file di configurazione	La QUERY_STRING , un insieme di coppie chiave/valore separate dal simbolo &
------------	---	--	--

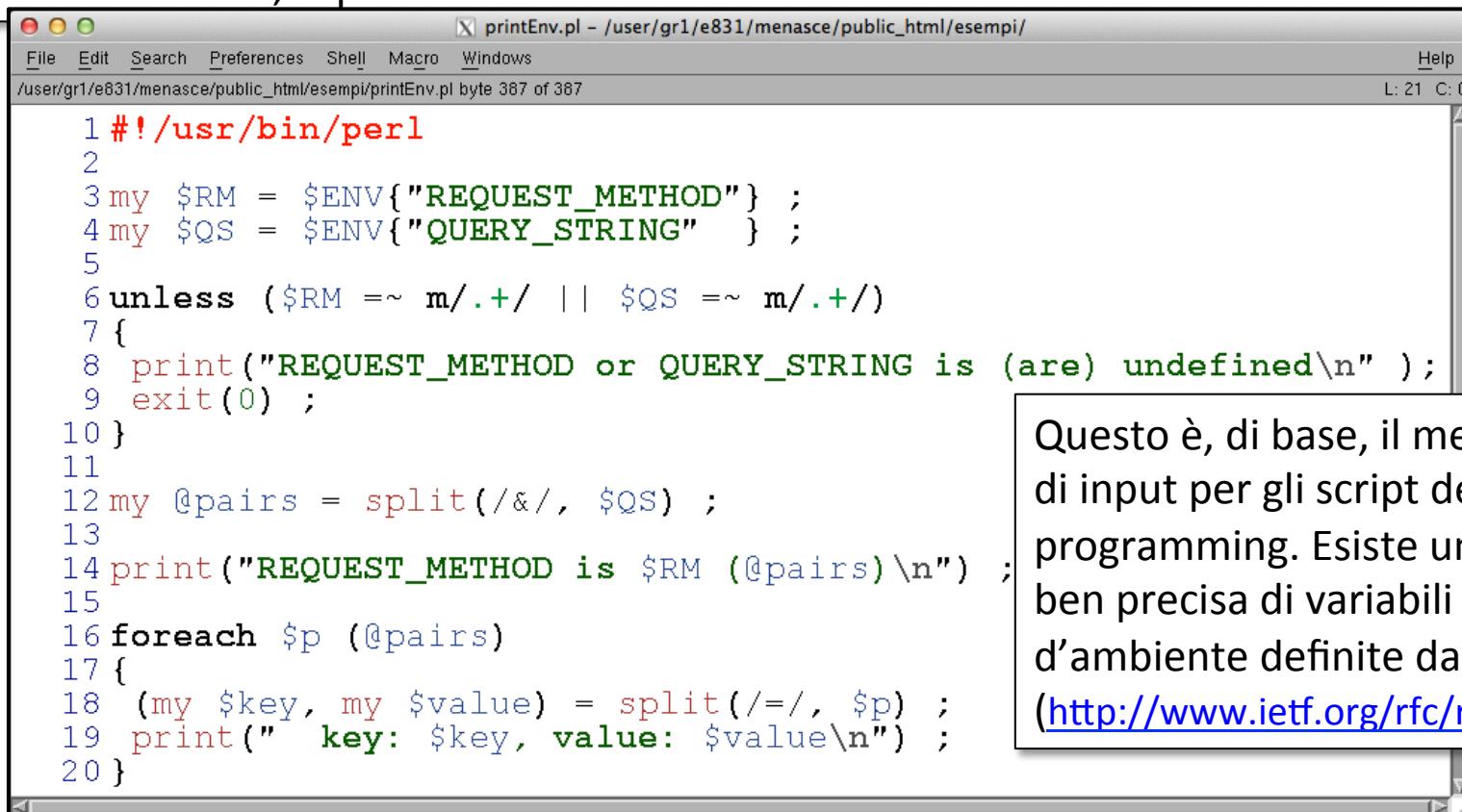
Bene, quando si spedisce questo **URL** al server indicato, questi lo intercetta come un programma da eseguire (grazie alla presenza della keyword **cgi-bin**) . A quel punto il *daemon* di Apache sul server istanzia un sotto-processo, (tramite **fork**), definisce alcune variabili di ambiente e tenta di eseguire il programma richiesto nell'ambito di quello stesso sotto-processo. In particolare vengono definiti:

- `setenv REQUEST_METHOD GET`
- `setenv QUERY_STRING 'searchString=doc&anotherPar=thisOne'`

Supponiamo di definire queste due variabili in una shell interattiva

- `setenv REQUEST_METHOD GET`
- `setenv QUERY_STRING 'searchString=doc&anotherPar=thisOne'`

e supponiamo di eseguire uno script perl che stampi il valore della prima variabile e le due coppie chiave valore, separate tre loro:



```

printEnv.pl - /user/gr1/e831/menasce/public_html/esempi/
File Edit Search Preferences Shell Macro Windows Help
L: 21 C: 0
/usr/gr1/e831/menasce/public_html/esempi/printEnv.pl byte 387 of 387
1 #!/usr/bin/perl
2
3 my $RM = $ENV{"REQUEST_METHOD"} ;
4 my $QS = $ENV{"QUERY_STRING"} ;
5
6 unless ($RM =~ m/.+/? || $QS =~ m/.+/?)
7 {
8   print("REQUEST_METHOD or QUERY_STRING is (are) undefined\n");
9   exit(0) ;
10 }
11
12 my @pairs = split(/&/, $QS) ;
13
14 print("REQUEST_METHOD is $RM (@pairs)\n") ;
15
16 foreach $p (@pairs)
17 {
18   (my $key, my $value) = split(/=/, $p) ;
19   print("  key: $key, value: $value\n") ;
20 }

```

Questo è, di base, il meccanismo di input per gli script del CGI programming. Esiste una serie ben precisa di variabili d'ambiente definite da Apache (<http://www.ietf.org/rfc/rfc3875>)

/user/gr1/e831/menasce/public_html/esempi/job.pl byte 1586 of 1811

```
39 sub decodeQS
40 {
41     my $buffer ; 
42     my $name ; 
43     my $pair ; 
44     my $value ; 
45
46     my @getpairs ; 
47     my @pairs ; 
48
49     if      ($ENV{'REQUEST_METHOD'} eq 'GET')
50     {
51         @pairs = split(/&/,      $ENV{'QUERY_STRING'} );
52     }
53     elsif   ($ENV{'REQUEST_METHOD'} eq 'POST')
54     {
55         read (STDIN, $buffer,  $ENV{'CONTENT_LENGTH'});
56         @pairs = split(/&/, $buffer);
57     }
58     else
59     {
60         print <<EOB ;
61
62 FATAL: parseForm() in the decodeForm.pm package issued an
63       'Unknown Request Method' error.
64       This is because the REQUEST_METHOD was not set to
65       GET nor to POST for the current shell.
66
```

Avevamo quindi lasciato in sospeso la descrizione del meccanismo usato per comunicare allo script attivato dal server i parametri necessari alla sua esecuzione (fondamentalmente, nel nostro esempio, il valore della *search string*, **doc**)

searchString=doc&anotherPar=thisOne

```
57 }
58 else
59 {
60     print <<EOB ;
61
62 FATAL: parseForm() in the decodeForm.pm package issued an
63       'Unknown Request Method' error.
64       This is because the REQUEST_METHOD was not set to
65       GET nor to POST for the current shell.
66
67 EOB
68 }
69
70 foreach $pair (@pairs)
71 {
72     ($name, $value) = split (/=/, $pair);
73     $name =~ tr/+/_;
74     $name =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
75     $value =~ tr/+/_;
76     $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
77
78     $value =~ s/<!--(.|\n)*-->///g;
79
80     $formdata{$name} = $value;
81 }
82 return %formdata ;
83 }
84 }
```

Ed ecco finalmente il risultato di questo esercizio:

Laboratories

Search

News

INFN

Docs

Linux

Social networks

Varie

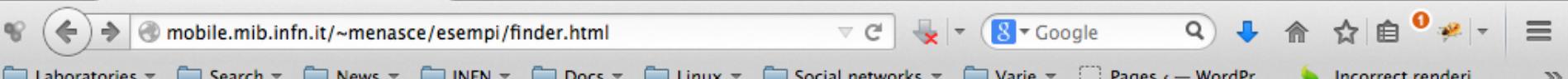
Pages ← WordPr...

Incorrect renderi...

Search string: doc

Submit Query

Search results



Search string: doc

Submit Query

Search results

89: document.getElementById("outputText").innerHTML = this.server_response;

94: var ss = document.getElementById("searchString").value;

[finder.html](#)

36: var divMenu = document.getElementById("menu") ;

[one_4.html](#)

36: var divMenu = document.getElementById("menu") ;

[one_5.html](#)

Visione d'insieme...

File Edit Search Preferences Shell Macro Windows

L: --- C: ---

```
1 <html>
2   <head>
3     <title>Finder</title>
4     <style type="text/css">
5       #formArea      {
6         position      : absolute          ;
7         display       : block            ;
8         top          : 2px              ;
9         left          : 10px             ;
10        width         : 730px            ;
11        height        : 50px             ;
12        position      : fixed           ;
13        border        : 1px solid #231  ;
14        padding       : 2px              ;
15        text-align    : left             ;
16        font-family   : arial            ;
17        color         : #f0f              ;
18      }
19      #outputArea    {
20        position      : absolute          ;
21        display       : block            ;
22        top          : 60px             ;
23        left          : 10px             ;
24        width         : 730px            ;
25        height        : 360px            ;
26        position      : fixed           ;
27        border        : 1px solid #e33  ;
28        padding       : 2px              ;
```

```
29         text-align : left ;
30         font-family : arial ;
31         color       : #00f ;
32     }
33 #outputAreaHead{
34         text-align : center ;
35         font-weight: bold ;
36         font-family : arial ;
37         color       : #f3f ;
38     }
39 </style>
40
41 <script>
42     function CallServer()
43     {
44         this.xhr_object;
45         this.server_response;
46
47         this.createXMLHttpRequest = createXMLHttpRequest;
48         this.sendDataToServer    = sendDataToServer;
49         this.displayAnswer       = displayAnswer;
50         this.launch              = launch;
51     }
52
53     function createXMLHttpRequest()
54     {
55         this.xhr_object = null;
56     }
```

```
57     if(window.XMLHttpRequest)
58     {
59         this.xhr_object = new XMLHttpRequest();
60     }
61 else if(window.ActiveXObject)
62 {
63     this.xhr_object = new ActiveXObject("Microsoft.XMLHTTP");
64 }
65 else
66 {
67     alert("Your browser doesn't support XMLHttpRequest");
68     return;
69 }
70 }
71
72 function sendDataToServer (dataToSend)
73 {
74     var xhr_object = this.xhr_object;
75     xhr_object.open("POST",
76                     "http://mobile.mib.infn.it/" +
77                     "cgi-bin/scripts/esempi/job.pl",
78                     false);
79     xhr_object.setRequestHeader("Content-type",
80                               "application/" +
81                               "x-www-form-urlencoded");
82     xhr_object.send(dataToSend);
83     if(xhr_object.readyState == 4)
84 {
```

```
85      this.server_response = xhr_object.responseText;
86  }
87 }
88
89 function displayAnswer ()
90 {
91     var t = this.server_response
92     document.getElementById("outputText").innerHTML = t;
93 }
94
95 function launch ()
96 {
97     var ss = document.getElementById("searchString").value;
98     this.sendDataToServer("search="+ss);
99     this.displayAnswer();
100 }
101
102 var callServer = new CallServer();
103 callServer.createXMLHttpRequest();
104 </script>
105
106 </head>
107
108 <body>
109 <div id="formArea">
110   <form>
111     Search string: <input id="searchString"
112                               type="text"
```

```
100      }
101
102      var callServer = new CallServer();
103      callServer.createXMLHttpRequest();
104  </script>
105
106 </head>
107
108 <body>
109  <div id="formArea">
110    <form>
111      Search string: <input id="searchString"
112                                type="text"
113                                name="searchString">
114      <input type="submit" onclick="callServer.launch()" />
115    </form>
116  </div>
117  <div id="outputArea">
118    <div id="outputAreaHead">
119      Search results
120    </div>
121    <div id="outputText">
122    </div>
123  </div>
124 </body>
125
126</html>
127
```


Fine seconda parte