Using Dv::Ticket::Cgi to write cgi applications

Dirk Vermeir

July 26, 2003

Contents

1 Basic cgi facilities 1

2 User authentication using tickets 2

3 A simple example 3

4 The Dv::Ticket::Cgi authentication protocol in more detail 5

Abstract

A quick example-based guide on how to use the Dv::Ticket::Cgi class to write cgi applications. See also the doxygen documentation for the dvticket and dvcgi packages.

This text is also available in postscript (example.ps) and pdf (example.pdf) format.

1 Basic cgi facilities

The **dvcgi** package provides a class `Dv::Cgi::Cgi` that gives the application access to:

- The environment variables as defined by the cgi protocol, e.g. the `REMOTE_ADDR` variable. Access is through a `Dv::Util::Props` object (see the dvutil documentation). E.g. the following code accesses the dot address of the browser accessing your program.

```
Dv::Cgi::Cgi cgi(..);

std::string browser_address = cgi.env()["REMOTE_ADDR"];
```

- The form data as provided, e.g. by a form or a complex url.

```
Dv::Cgi::Cgi cgi(..);

if ( cgi.props().find("age") ) { // form data variable age is available
  int age( cgi.props()["age"] ); // automatic conversion from string
  ..
  }
```

- Cookies, again through a `Dv::Util::Props` object.

```
Dv::Cgi::Cgi cgi(..);

if ( cgi.cookies().find("ticket") ) { // cookie "ticket" is available
  int ticket_id( cgi.cookies()["ticket"] );
  ..
  }
```

In addition, one can set headers of the HTTP(S) reply using the `Dv::Cgi::HttpHeader` object that can be obtained by `cgi.header()`. The following code sets a cookie and then redirects the browser to a different url.

```
Dv::Cgi::Cgi cgi(..);
std::string url;
// set cookie with name "ticket" and value "123"
cgi.header().cookie("ticket",Dv::Util::tostring(123));
cgi.header().location(url);
```

Since `Dv::Cgi::Cgi` is derived from `std::ostream`, one can write to it using `operator«`. The following program prints "hello world" to the browser, followed by a listing of environment, form data and cookie variables.

```
Dv::Cgi::Cgi cgi(..);

cgi.header().content_type("text/plain");
cgi << "hello world" << std::endl
    << "env: " << std::endl << cgi.env() << std::endl
    << "form data: " << std::endl << cgi.props() << std::endl
    << "cookies: " << std::endl << cgi.cookies() << std::endl;
```

Warning
All header settings must be performed before writing output to the `cgi` object.

There are other facilities that are not discussed here, see the documentation for the dvcgi package.

2 User authentication using tickets

The class `Dv::Ticket::Cgi` (from the **dvticket** package) is derived from `Dv::Cgi::Cgi`. It adds automatic authentication using tickets.

A ticket is identified by a unique unsigned long which is associated to a user and a (browser) host for a limited period. The browser keeps the id of a ticket as a cookie, thus making it possible for a cgi program to obtain the ticket id from the browser and then get the ticket details (the user) from a ticket server.

One of the parameters of the constructor of `Dv::Ticket::Cgi` is the url of a so-called "login server". The constructor will operate as shown in the following simplified pseudo-code:

```
if (there is a "ticket" cookie) {
  obtain the ticket id from the cookie
  validate the ticket, using the id, with the ticket server
  }
else
  redirect to the login server url (a parameter of the constructor)
```

The login server will show a form where the user can fill in his name and password. These will be verified by the login server and, if ok, a new ticket will obtained from a ticket server. The login server will arrange for the id to be associated with a "ticket" cookie in the browser. A more detailed description can be found in another section .

The net result is that, when the constructor for the `Dv::Ticket::Cgi` object finishes, a valid ticket is available, as illustrated by the simple program below.

```
#include <dvticket/cgi.h>

int
main(int argc,char* argv[]) {
Dv::Ticket::Cgi cgi("test.cgi", "tinf2.vub.ac.be/~dvermeir/login.cgi", "en", true);
cgi.header().content_type("text/plain");

try {
  cgi << "You are " << cgi.user().name()
      << ", your category is " << cgi.user().category()
<< " and your id is " << cgi.user().id() << std::endl;
  }
catch (std::exception& e) {
  cgi << e.what() << std::endl;
  return 1;
  }
return 0;
}
```

3 A simple example

The example cgi program shows a form asking the user to fill in his age and nickname. It then echoes back the incremented age and other data.

The program uses an auxiliary function which substitutes references of the form `%{name}` in a file by the value associated with `name` in a `Dv::Util::Props` object.

```
// Output file fn on cgi, replacing %{name} in the file by out["name"]
bool
show_file(Dv::Cgi::Cgi& cgi, const Dv::Util::Props& out,
const std::string& fn) {
std::ifstream ifs(fn.c_str()); // open html template file
if (ifs) {
  std::ostringstream oss;
  out.substitute(ifs, oss); // replace e.g. %{age} by out["age"]
  cgi << oss.str() << std::endl;
  }
else {
  cgi->content_type("text/plain");
  cgi << fn << ": could not open\n\n" << out << std::endl;
  }
}
```

The main program refers to the login server at `tinf2.vub.ac.be/ dvermeir/login.cgi` to obtain a ticket, if one is not available through a cookie.

```
#include <dvticket/cgi.h>

// definition of show_file omitted

int
main(int argc,char* argv[]) {
Dv::Ticket::Cgi cgi("test.cgi", "tinf2.vub.ac.be/~dvermeir/login.cgi", "en", true);
try {
  Dv::Util::Props out; // used to substitute in html files
  out.add("username", cgi.user().name());
  out.add("category", cgi.user().category());
  out.add("info", cgi.user().info());
  out.add("uid", Dv::Util::tostring(cgi.user().id()));
  out.add("here", cgi.here() ); // the url of this program

  // if there is no form data variable "age", this must
  //   be the first time we are activated
  if (! cgi.props().find("age"))
    show_file(cgi, out, "test-cgi-a.html");
  else { // form has been filled
    try { // conversion to int age may throw exception
      int age = cgi.props()["age"];
      out["age"] = ++age; // increment
      out.add("nickname", cgi.props()["nickname"]);
      show_file(cgi, out, "test-cgi-b.html");
      }
    catch (std::exception& e) { // wrong input from user, show form again
      cgi->location(cgi.here());
      cgi << std::endl;
      }
    }
  }
catch (std::exception& e) {
  cgi->content_type("text/plain");
  cgi << e.what() << std::endl;
  return 1;
  }
return 0;
}
```

The second html template file is shown below.

```
 <html>
  <head>
   <title>TEST-CGI CLIENT</title>
   <style type="text/css">
         a:link { text-decoration:none }
         a:active { text-decoration:none }
         a:visited { text-decoration:none }
   </style>
  </head>
  <body bgcolor="#ffffff">
   <h1>TEST-CGI CLIENT</h1>
   <p>Welcome, %{category} %{username}.</p>
   <p>Your id is %{uid}.</p>
```

```
  <p>Associated info with your ticket is %{info}.</p>
  <p>Your age is %{age} and your nickname is %{nickname}.</p>
  <p><a href="%{here}">refresh</a></p>
 </body>
</html>
```

A skeleton **Makefile** used to compile and link the example is shown below (of course, using autotools is much simpler).

```
CPPFLAGS=-I /usr/local/include
LDFLAGS=-Wl,-rpath -Wl,/usr/lib/:/usr/local/lib/
LIBS= -L/usr/local/lib \
 -ldvticket -ldvmysql -ldvssl -ldvcgi -ldvnet -ldvxml -ldvutil \
 -lxmlwrapp -lxml2 -lxsltwrapp -lexslt -lxslt -lssl -lcrypto \
 -L/usr/lib/mysql -lmysqlclient
test.cgi: test-cgi.o
 g++ -o $@ $(LDFLAGS) $(LIBS) $<
```

A tarball with the source for a similar example is available in the file testcgi.tar.

4 The Dv::Ticket::Cgi authentication protocol in more detail

We describe what happens when the constructor of the `Dv::Ticket::Cgi` object does not find a "ticket" cookie.

We use `browser`, `login` and `client` to denote the host where the respective programs (browser, login server and the host where the `Dv::Ticket::Cgi` object is constructed) run (note that the `login` host also runs the ticket server).

1. `browser` contacts client without or with invalid or expired "ticket" cookie.
2. `client` redirects to `login` url.
3. `browser` contacts `login` url.
4. `login` program shows page to fill user name, password.
5. `browser` sends `username`, `password` to login program
6. `login` program obtains new ticket
7. `login` program redirects to `client?ticket=id` (note the "ticket" variable in the form data)
8. `browser` contacts `client`, with `ticket` variable in the form data.
9. `client` notices "ticket" variable in the form data, sets a "ticket" cookie with the same value and succeeds the `Dv::Ticket::Cgi` constructor.

The end result, upon a succesful login, will be that the browser has a cookie identifying a ticket for the `client` host. In such a case, the `client` simply validates the ticket id with the ticket server, thus obtaining e.g. information on the user.