



C++ Multiplatform and Object Orientation

Part 5: Web/CGI software development using C++ and VBMcgi



By: Sergio Barbosa Villas-Boas (sbVB)

<http://www.vbmcgi.org/> (official site)

<http://sourceforge.net/projects/vbmcgi> (recommended site for download)

<http://tech.groups.yahoo.com/group/vbmcgi/> (discussion group)

vbmcgi@yahoogroups.com (discussion list)

Version 8.1, of September 04th, 2003

Copyright © 1992~2003 by sbVB

Author's email: sbvb@sbvb.com.br

Table of Contents

Table of Contents.....	2
1) Introduction	4
1.1 Browsing the web.....	6
1.2 CGI.....	8
1.3 Multiply table.....	10
1.4 CGI programs and the environment.....	11
1.5 History and future of VBMcgi.....	15
1.6 Installing VBMcgi	15
1.6.1 Windows, Visual C++ 6.0	16
1.6.2 Unix, Gnu C++ compiler	20
2) Using VBMcgi.....	22
2.1 Main characteristics and advantages of using VBMcgi.....	22
2.2 Hello VBMcgi	22
2.3 Decoding HTML form	23
2.4 Multiply table with parameter	23
2.5 String parameters.....	24
2.6 “3-tier” web software architecture	26
2.7 VBMcgi golden rule and tier isolation	28
2.8 String change feature.....	28
2.9 formDecode adds variables.....	30
2.10 Complete form reference	31
2.10.1 textbox	31
2.10.2 textarea	31
2.10.3 checkbox (caixa de checagem)	32
2.10.4 radio button	33
2.10.5 drop down	34
2.11 Call function feature.....	35
2.11.1 Call function feature passing parameters from C++	38
2.11.2 Call function feature passing parameters from HTML	40


2.12	Cookies.....	41
2.12.1	Configuring the browser to see the cookies	43
2.12.2	Persistent cookies	44
2.12.3	The cookie's domain	44
2.12.4	Login to web using cookies	44
2.13	gif or jpeg output (non HTML data in CGI programs)	47
2.14	Redirection.....	49
3)	References.....	52

1) Introduction

Web can be understood as the popular name of a package of technologies that include the items below. It's also known as www or World Wide Web.

- HTTP (Hyper Text Transfer Protocol)
- URL (Universal Resource Locator)
- HTML (Hyper Text Markup Language)
- CGI (Common Gateway Interface)

By using HTTP, one can easily develop client-server software systems, because both client and server software are already developed, they are taken for granted. So, only by writing HTML code and CGI programs, a web system is developed. Some of the popular software for HTTP client are Internet Explorer, Netscape Navigator and Opera. Some of the popular software for HTTP server are Apache, CERN and Microsoft's IIS.

VBMcgi is a free library for web/CGI software development. The latest version for download, tutorial and related information can be copied from the VBMcgi web page at <http://www.vbmcgi.org>. One of the main distinctive features of VBMcgi is that it allows isolation of the presentation tier and the business rules tier. That is, isolation of the webdesigner and the webmaster. Further isolation from the database can be added thus achieving the widely proved 3-tier software architecture. VBMcgi stands for "Villas-Boas and Martins library for cgi development". Villas-Boas is also known as sbVB . VB stands for Villas-Boas, *not* for Visual Basic.

For short, CGI is the interface that allows HTML pages to execute programs in the HTTP server, and direct the result to the HTTP client. Here is one example: a user wants to search the web on the keyword "banana". If he chooses the google's web site (www.google.com), he will type in "banana" in a form of a web page of google's web system, and by using the CGI interface the keyword "banana" will be passed as a parameter to a program somewhere (related to a google's HTTP server), that will process the keyword and return to the HTTP client a list of links for web pages related to it.

Those who invented the CGI interface¹ did that in a very clever way. The CGI interface connects HTML data to a computer program without requirement of a specific computer language. This computer program can be either in binary (compiled) form, or be a script (text files) that are to be interpreted by some software installed in the server. Some of the popular script interpreters used for CGI are: Perl, PHP, ASP, JSP, Cold Fusion. The CGI interface does not forbid that someone invent a new script interpreter in the future. Still, there's no

¹ European Laboratory for Particle Physics. <http://www.cern.ch/>

problem to use more than one type of script at the same time, as far as all script interpreters are properly installed in the HTTP server. CGI programs in script and binary form can also be mixed up without any problem.

If you use C++ for CGI development, it's obvious that you're looking for a nice library that handles most of the repetitive tasks of decoding forms, sending headers, etc. There are many C/C++ libraries for CGI development. This document presents the VBMcgi library. This library is open source and LGPL licensed (no restrictions, can be used for commercial applications). Still, this library is multiplatform, tested in HTTP servers in unix and Windows. CGI programs can run equally in Windows and unix, but recompilation is required.

The main feature that VBMcgi has that distinguishes it from other libraries is its golden rule, shown below.

To isolate the work of webdesigner (tier 1, for presentation) and webmaster (tier 2, for implementation of business rules)

Figure 1: VBMcgi golden rule

VBMcgi allows and leads one to use the widely proved 3-tier software architecture, as will be shown later. For the moment, the important thing to keep in mind is that any commercial HTML design tool (such as FrontPage, DreamWeaver, GoLive) can and should be used. Due to VBMcgi's golden rule, there's no C++ code mixed up with HTML code. One important consequence of this is that the webdesigner does not have to know about C++, OO or anything like that. Only the webmaster needs to know about that. Complex HTML code (something that HTML authoring tool understands, but a human being has trouble in understanding) can be used without any problem at all. If the webmaster and webdesigner are the same person, it is of great value to isolate these works, because this brings good maintainability (easiness to handle small requirement changes). The golden rule is implemented from 2 features of VBMcgi shown below (to be explained in detail latter).

- string change feature
- call function feature

To test the CGI programs, one has to know how to compile and place the executables in a directory that the HTTP server has been mapped to. For Windows users, one can get a free copy of the apache web server, or use Microsoft's IIS, and do all the work in one computer not linked to the Internet. To develop like this is sometimes convenient, because it may be not easy to be connected to Internet all time. The CGI programs need the HTTP server running to be tested. The HTTP browser refers it with <http://localhost>.

To help the newbies, there's a section with introduction about web.

1.1 Browsing the web

Let's have a quick revision of what really happens when someone browses the web. Suppose the web user wants to see a certain page. He should know of its URL (Universal Resource Locator).

Suppose the URL is "http://www.google.com". By typing in the given URL in the proper field of the web browser (web client), if everything is operating normally, the page will soon appear.

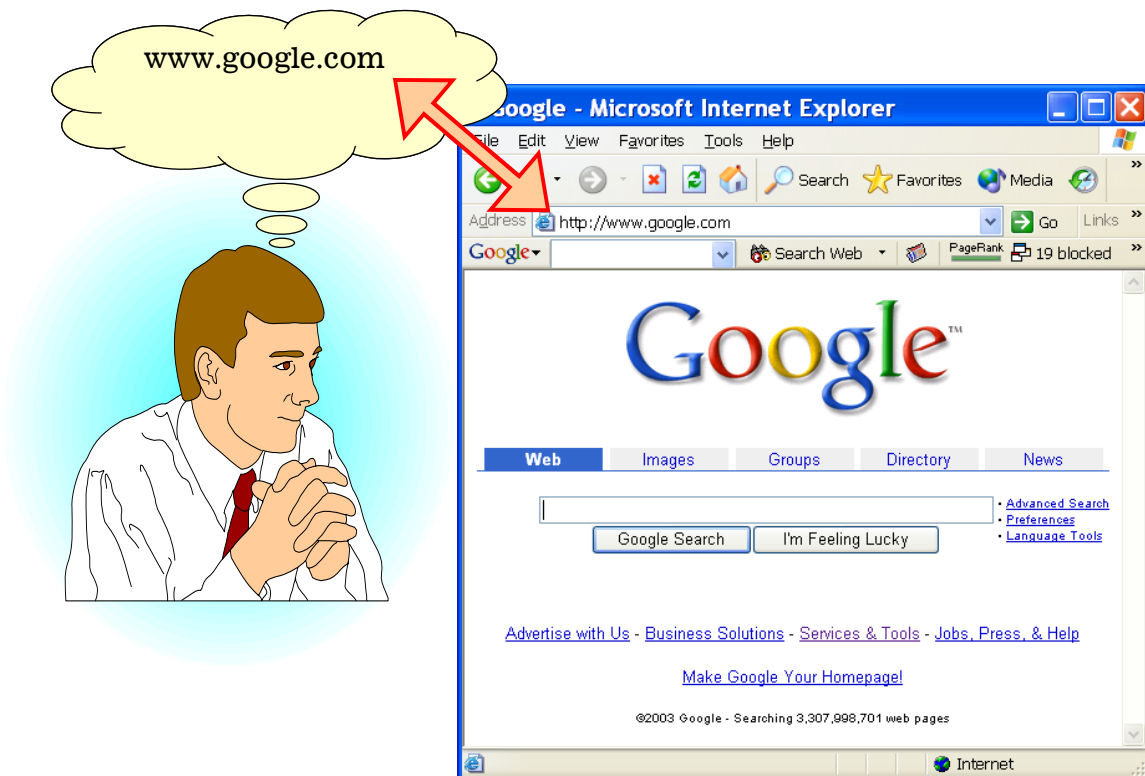


Figure 2: The web user types in the URL he wants to see, and gets the result

From what he sees, the web user clicks in links, and that commands new requests to the web server. Suppose the client clicks to something that requests "file.html" from the server. That will make the server to copy the "file.html" to the client, and the user will see its contents.

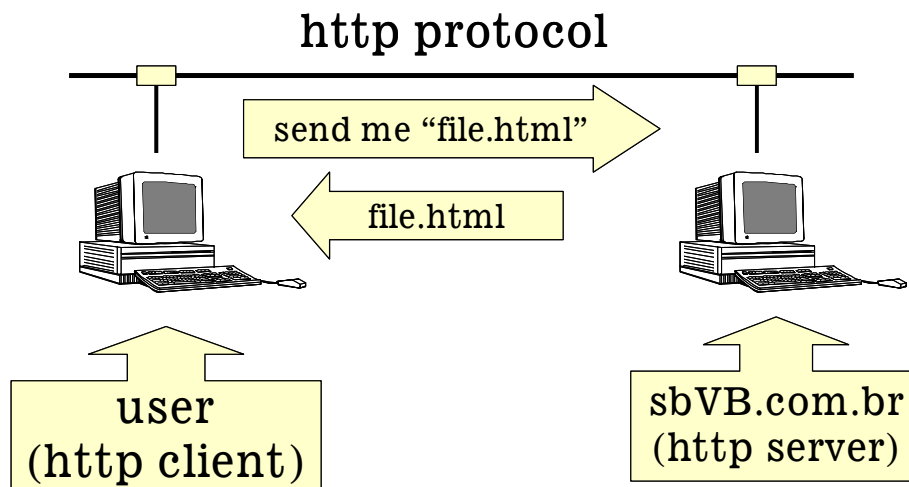


Figure 3: The client requests data, and the server sends it.

It's important to recall that all this browsing happens *without* need to specify what software is the HTTP server and what software is the HTTP client, or even what operating system the client or the server is running. The only needed standardization is the TCP/IP and web.

To create a web page is very simple. Just use a text editor and type in something with in the HTML standard, and save it say as a hello.html, like shown below.

```
<!-- hello.html -->
<HTML><body>
Some text in HTML
</body></HTML>
```

The result is shown below².

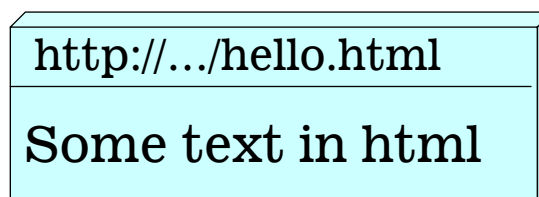


Figure 4: A simple HTML file seen in the web client.

To create a link (or anchor) is very simple. There's a HTML tag that represent the anchor, or "a" for short. In the example below, the files a.html and b.html are linked one to the other.

```
<!-- a.html -->
<HTML><body>
Click <a href="b.html">here</a> for more details.
</body></HTML>
```

```
<!-- b.html -->
<HTML><body>
More details. <p>
```

² This box is a pictorial representation of the web client.

```
Click <a href="a.html">here</a> to go back.
</body></HTML>
```



Figure 5: Two simple HTML documents with links one to the other.

1.2 CGI

The CGI allows that a HTML page call a program in the HTTP server (can be either a binary executable or a script). The CGI interface does not impose any particular computer language to be used for the program in the server. Some of the languages used are: C/C++, Delphi, Perl, ASP, PHP, JSP (Java) and Cold Fusion. This document is a tutorial that uses C++ and the library VBMcgi.

Let's see how can one do a hello_cgi program in C++. Using a C++ compiler, one can build a binary executable from the source file. For simplicity, the extension of the binary executable that are intended to be used as CGI programs will have the extension *.cgi³.

In VBMcgi web page, the extension *.chtml is an executable file the same way the *.cgi extension is.

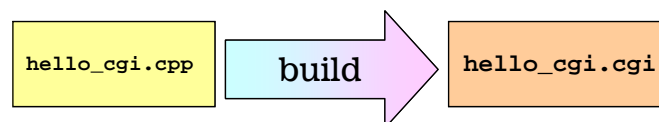


Figure 6: To build is to use the compiler to create a binary executable from the C++ source files

³ In unix, any file extension can be used to be an executable. In Windows, most compilers produce binary executables with the extension *.exe. But these files can be renamed to *.cgi, and the web server will run them the same way as if they were *.exe.

```
// hello_cgi.cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Content-type:text/HTML" << endl << endl;
    cout << "<HTML><body>" << endl;
    cout << "Some text in HTML" << endl;
    cout << "</body></HTML>" << endl;
    return 0;
}
```

See this example in <http://www.vbmcgi.org/index.html?p=6>.

The program `hello_cgi.cgi`, if executed in line command, will produce the result below. For a CGI program, the HTTP protocol requires the line with “Content-type:text/HTML”, and the blank line that follows it. That’s called the HTTP header.

```
Content-type:text/HTML
```

```
<HTML><body>
Some text in HTML
</body></HTML>
```

The cgi executable program can be referenced with ordinary URL standards. Thus HTML code can link to CGI program as if it was an ordinary HTML file.

```
<!-- call_cgi.html -->
<HTML><body>
Click <a href="hello_cgi.cgi">here</a> to call a cgi program directly
</body></HTML>
```

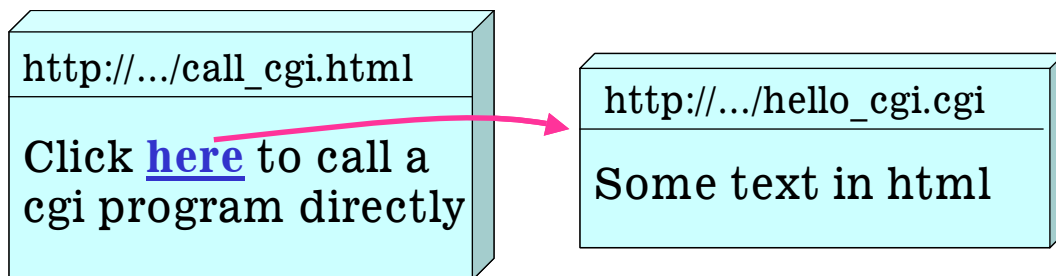


Figure 7: Since the CGI program has a URL, it can be linked directly from a HTML file.

An other way to call a CGI program is to use HTML tag `<form>`.

```
<!-- call_cgi_2.html -->
<html><body>
Example of cgi: <p>
<form action="hello_cgi.cgi" method="get">
<input type="submit" value="Send Data">
</form>
</body></html>
```

See the VBMcgi tutorial in the url below.

<http://www.vbmcgi.org/index.html?p=9>⁴

```
<!-- form_call_to_cgi.html -->
<HTML><body>
Hello CGI example: <p>
<form action="hello_cgi.cgi" method="get">
<input type="submit" value="send">
</form>
</body></HTML>
```

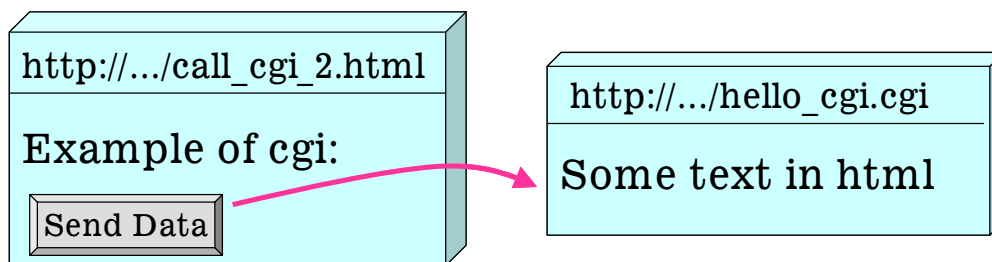


Figure 8: A HTML page linked to a CGI program by a form

1.3 Multiply table

Now let's see an example of a CGI program that does some kind of loop. A classical example is the multiply table, like shown below (still not using VBMcgi).

```
// multiply_table.cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Content-type:text/HTML" << endl << endl;
    cout << "<HTML><body>" << endl;
    int multiply_table = 5;
    for (int i=1 ; i <= 10 ; i++)
        cout << i* multiply_table << "<br>" << endl;
    cout << " </body></HTML>" << endl;
    return 0;
}
```

The output of this program, if seen in the console, is shown below.

```
Content-type:text/HTML
```

```
<HTML><body>
5<br>
10<br>
15<br>
20<br>
25<br>
30<br>
35<br>
40<br>
45<br>
```

⁴ In the site www.vbmcgi.org, the cgi programs have extension *.html, as well as *.cgi.

```
50<br>
</body></HTML>
```

See a link for that example in the VBMcgi site in the url below.

<http://www.vbmcgi.org/index.html?p=7>



Figure 9: Multiply table in the browser

Obviously, the “5”, that is fixed in this example, should be a parameter in a more general example. This will be done later.

1.4 CGI programs and the environment

The HTTP allows communication from HTML pages to the CGI programs using the environment. The environment is an entity that can store variables. One famous variable stored in the environment is the “path” (list of directories to be searched when an executable command is typed in). Environment variables are in string format. HTML variables such as name and content of HTML forms are passed as parameters to the CGI program. The HTTP server writes properly the environment variables before calling the CGI program.

Using the standard library’s function “getenv”, a variable from the environment can be read. Some of the environment variables that can be useful to CGI programs are read in the program below. To make it simple, there’s the function “printEnv”, that exhibits the variable name and its contents.

```
// getenv.cpp
#include <iostream>
using namespace std;
#include <stdlib.h> // getenv

void printEnv(const char *var)
{
    char *t = getenv(var);
    if (!t) t="";
```

```
cout << "Environment[" << var << "]" = \"\" << t << "\"<br>" << endl;
}

int main()
{
    cout << "Content-type: text/HTML" << endl << endl;
    printEnv("SERVER_SOFTWARE");
    printEnv("SERVER_NAME");
    printEnv("GATEWAY_INTERFACE");
    printEnv("SERVER_PROTOCOL");
    printEnv("SERVER_PORT");
    printEnv("REQUEST_METHOD");
    printEnv("HTTP_ACCEPT");
    printEnv("PATH_INFO");
    printEnv("PATH_TRANSLATED");
    printEnv("SCRIPT_NAME");
    printEnv("QUERY_STRING");
    printEnv("REMOTE_HOST");
    printEnv("REMOTE_ADDR");
    printEnv("AUTH_TYPE");
    printEnv("REMOTE_USER");
    printEnv("REMOTE_IDENT");
    printEnv("CONTENT_TYPE");
    printEnv("CONTENT_LENGTH");
    printEnv("HTTP_USER_AGENT");
    printEnv("HTTP_REFERER");
    return 0;
}
```

If the program `getenv.cgi` is executed in line command, the output will be like below (all variables empty).

```
Content-type: text/HTML

Environment[SERVER_SOFTWARE] = ""<br>
Environment[SERVER_NAME] = ""<br>
Environment[GATEWAY_INTERFACE] = ""<br>
Environment[SERVER_PROTOCOL] = ""<br>
Environment[SERVER_PORT] = ""<br>
Environment[REQUEST_METHOD] = ""<br>
Environment[HTTP_ACCEPT] = ""<br>
Environment[PATH_INFO] = ""<br>
Environment[PATH_TRANSLATED] = ""<br>
Environment[SCRIPT_NAME] = ""<br>
Environment[QUERY_STRING] = ""<br>
Environment[REMOTE_HOST] = ""<br>
Environment[REMOTE_ADDR] = ""<br>
Environment[AUTH_TYPE] = ""<br>
Environment[REMOTE_USER] = ""<br>
Environment[REMOTE_IDENT] = ""<br>
Environment[CONTENT_TYPE] = ""<br>
Environment[CONTENT_LENGTH] = ""<br>
Environment[HTTP_USER_AGENT] = ""<br>
Environment[HTTP_REFERER] = ""<br>
```

But if this same program is executed not by the user, but by the HTTP server, as a CGI program, the HTTP server will set data to some environment variables before calling the CGI program. The output seen in the browser will be like below (some variables not empty). See the VBMcgi example in the links below.

<http://www.vbmcgi.org/index.html?p=8><http://www.vbmcgi.org/getenv.cgi>

```

URL: http://www.vbmcgi.org/getenv.cgi

Environment[SERVER_SOFTWARE] = "Apache/1.3.12 Ben-SSL/1.41 (Unix) tomcat/1.0"
Environment[SERVER_NAME] = "vbmcgi.org"
Environment[GATEWAY_INTERFACE] = "CGI/1.1"
Environment[SERVER_PROTOCOL] = "HTTP/1.1"
Environment[SERVER_PORT] = "80"
Environment[REQUEST_METHOD] = "GET"
Environment[HTTP_ACCEPT] = "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*"
Environment[PATH_INFO] = ""
Environment[PATH_TRANSLATED] = ""
Environment[SCRIPT_NAME] = "/getenv.cgi"
Environment[QUERY_STRING] = ""
Environment[REMOTE_HOST] = ""
Environment[REMOTE_ADDR] = "146.164.50.16"
Environment[AUTH_TYPE] = ""
Environment[REMOTE_USER] = ""
Environment[REMOTE_IDENT] = ""
Environment[CONTENT_TYPE] = ""
Environment[CONTENT_LENGTH] = ""
Environment[HTTP_USER_AGENT] = "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"
Environment[HTTP_REFERER] = ""

```

Figure 10: Observing environment variables in the web browser

There's a lot of useful information in the environment variables. But we will not go much deeper into that.

One environment variable of particular interest is the `QUERY_STRING`. This variable stores the content of the HTML form variables (its names and its contents). This variable stores in a single string all form variables and its contents, in a standardized format. To avoid problem with forbidden chars, the form field names and its contents are converted to the "escape sequence".

One obvious task of a C++ library to support CGI programming is to read the `QUERY_STRING`, revert the escape sequence, and let easy access of all HTML form data and its contents.

In HTML, there are 2 methods to send form data – get and post. If the method get is used, the form data is stored in the URL, after the char "?". The formation of the `QUERY_STRING` is shown below.

```
field_1_name= field_1_content& field_2_name= field_2_content ...
```

```

<!-- test_getenv.html -->
<HTML><body>
<form action="mycgi.cgi" method="get">
Field one: <input type="text" name="field_1" size="10"><br>
Field two: <input type="text" name="field_2" size="10"><br>
<input type="submit" value="enviar">
</form>

```

</body></HTML>

Field one:

Field two:

```
http://.../mycgi.cgi?field_1=content_1&field_2=content_2
```

QUERY_STRING

Figure 11: A simple HTML form, and its query_string

For this HTML form, with these 2 fields with these 2 contents, the QUERY_STRING will be like below.

```
field_1=content_1&field_2=content_2
```

To separate what is the field name and correspondent field content is easy string manipulation looking for key chars '=' and '&'. Due to the escape sequence, it will never happen that either the field name or the field content have these 2 key chars. For instance: if content_1 is "aviação&", and content_2 is "a=b", the QUERY_STRING will be like below.

```
field_1=avia%E7%E3o%26&field_2=a%3Db
```

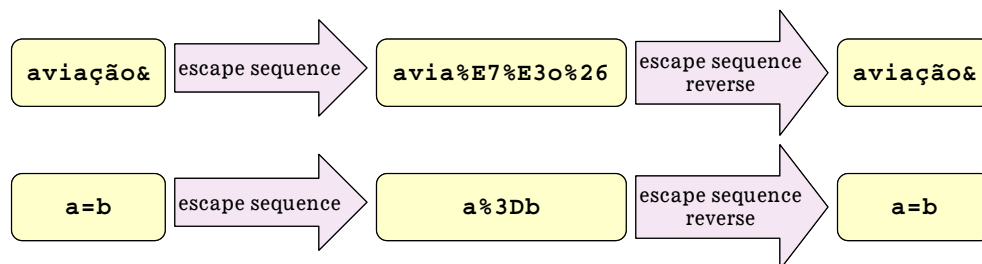


Figure 12: conversion to and from the escape sequence

It's the HTTP server that converts to the escape sequence. And it should be the CGI program to run the escape sequence reverse. By using VBMcgi, the escape sequence reverse is done automatically.

If the put method was used instead of the get method, then the form field names and its contents would be passed to the CGI program by the console, and not by the URL. One usefulness of this is to pass data that is large. The get method limits the QUERY_STRING to be no longer than 256 chars.

By using VBMcgi, the programmer does not need to care for method put or method. Both are used in the very same way.

If one wants to do personal tests with escape sequence and escape sequence reverse, those are methods implemented in the VBString classe (included in VBMcgi library). Try the code below (requires intalation of VBMcgi).

```
// code to test the scape sequence
#include "vbmcgi.h"
int main()
{
    VBString a;
    a = "1/2=0.5";
    cout << a << endl;
    a.scapeSequence();
    cout << a << endl;
    a.scapeSequenceReverse();
    cout << a << endl;
    return 0;
}
```

This should produce the output below.

```
avição&
avia%e7%e3o%26
avição&
```

1.5 History and future of VBMcgi

The VBMcgi library is a project idealized and maintained by the author of this document. It's a library developed from scratch, and the version 1.0 of it was the academic work of Alessandro Martins, under my supervision. The version 1.0 was made available in November of 2000 in web page www.vbmcgi.org.

Since its distribution, the usage of this library is the subject of regular academic study in DEL [1], and several projects were developed using it. From the use of many, gradually suggestions of improvement were implemented. In January 14th 2002, the version 2.0 was released, with several improvements and bug fixes. In September 04th 2003, the version 3.0 was released. This tutorial is related to version 3.0 of VBMcgi. See the complete history of VBMcgi in the link "history" of VBMcgi page.

VBMcgi is intended to remain free and LGPL licensed for ever. Anyone that contributes for it, will have the contributors name mentioned and the work made available for everyone.

Please, send your suggestions and questions to vbmcgi@yahoogroups.com, or access the discussion group page at <http://groups.yahoo.com/group/vbmcgi/>.

1.6 Installing VBMcgi

In this section, it is shown how to install the VBMcgi library in some useful cases. The library is developed to be multiplatform, but here it will be shown only how to use 2 compilers: Visual C++ 6.0sp5 in Windows and g++ 3.2 in linux.

1.6.1 Windows, Visual C++ 6.0

This subsection is for the users of Visual C++ 6.0 service pack 5. It is known that VBMcgi does not work without the service pack of Visual C++ 6.0.

Start by getting the distribution file VBMcgi_win_30.zip from the VBMcgi page. Unzip it for a working directory. There should be workspace file VBMcgi.dsw and 4 directories. Use the Visual C++ to open the workspace file. In the menu, command **B**uild – **B**atch **b**uild. The dialog box below should appear. Click Build and wait for the complete compilation.

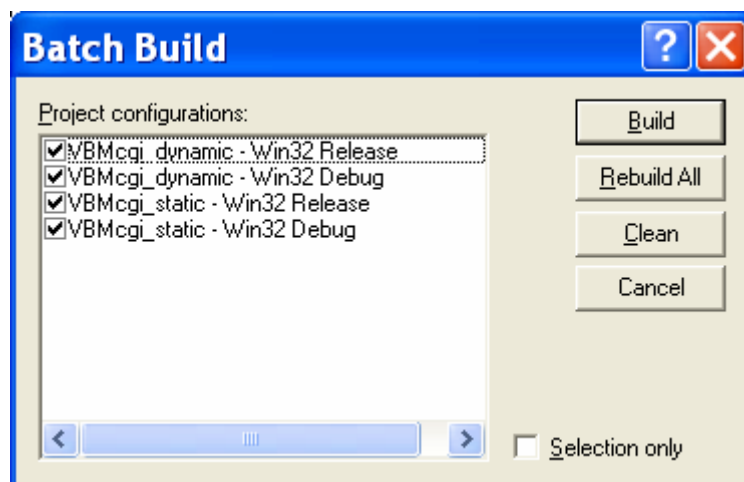


Figure 13: Dialog box for batch build

The build process should be carried out without problem. There are 4 outputs: version static and dynamic, each in release and debug version. Now, let's start with the easy to use static version of VBMcgi. Right-click the project VBMcgi_static and make it the active project. Expand the Source Files and see the file main.cpp.

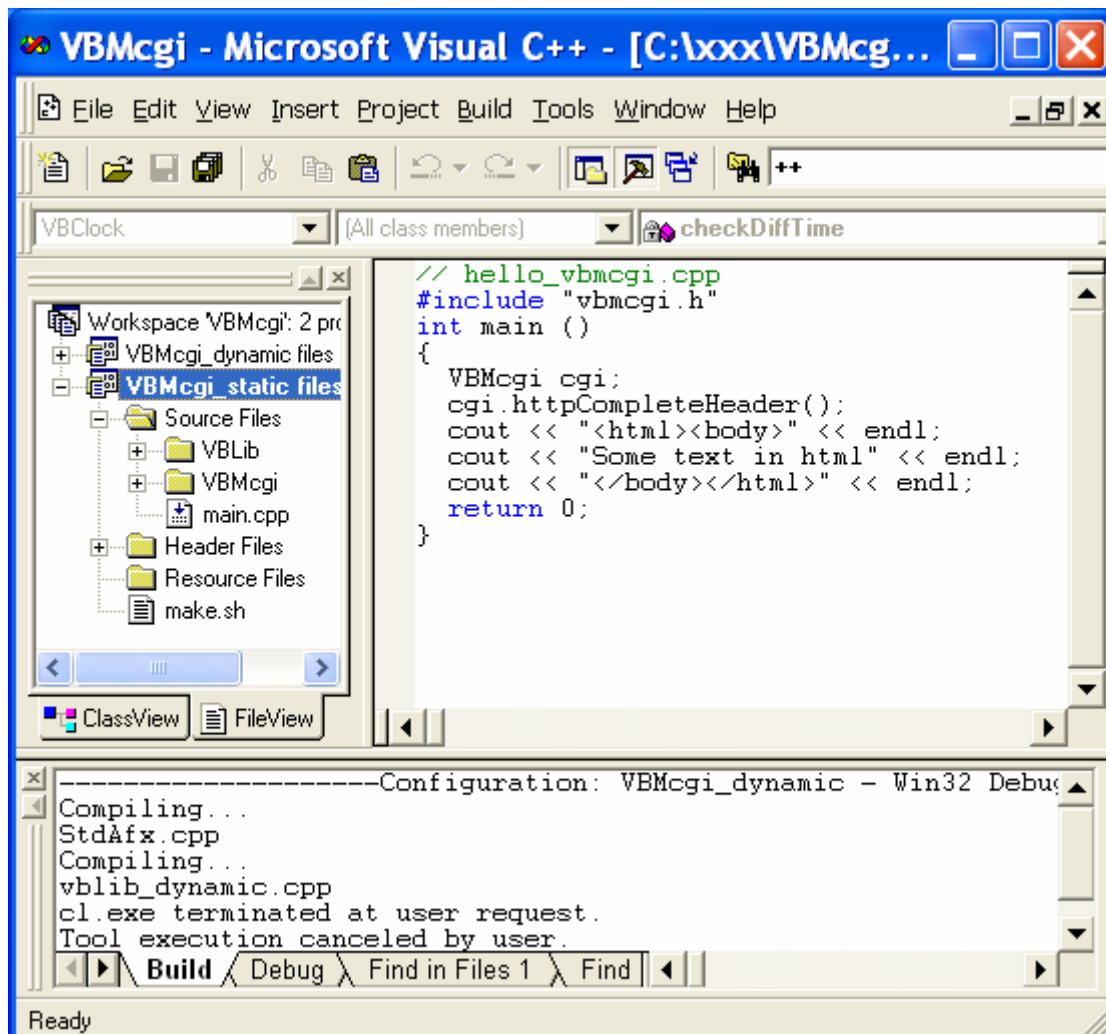


Figure 14: Visual C++ with the hello_vbmcgi.cpp as the main file

There's an example there but you can ignore it. Just erase the entire file and copy the first example hello_vbmcgi.cpp in the listing below.

```

// hello_vbmcgi.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpCompleteHeader();
    cout << "<HTML><body>" << endl;
    cout << "Some text in HTML" << endl;
    cout << "</body></HTML>" << endl;
    return 0;
}

```

To build the project, press F7. Execute it with <ctrl-F5>. The output should be as shown below. This hello VBMcgi example is worth only to check that one can download and compile the library properly. If run in command line, the output of this program is shown below.

Content-type: text/HTML

```
<HTML><body>
Some text in HTML
</body></HTML>
```

If you want to develop in the windows operating system, than it's a good idea to have a HTTP server installed to your computer. You can download and use the apache for windows HTTP server, or use some other HTTP server software. The download of apache HTTP server is available from the apache page in www.apache.org. In my machine, I got the distribution file below, downloaded from the apache's web site.

apache_2.0.47-win32-x86-no_ssl.msi

By clicking in this file, the installation of the apache web server begins. If you choose the recommended parameters for installation, that includes usage of port 80. Be sure that you don't have another web server installed to your computer. It can happen that there's another HTTP server installed. There exist the Microsoft's HTTP server called IIS. Open the control panel, click in the add software icon, choose windows components, and check if IIS HTTP server is installed. If it is, uninstall it.

After installing the apache HTTP server, it may be needed some special configuration, to allow it to use compiled cgi programs (something that the standard configuration won't let you to do). To alter the apache's configuration, find the file httpd.conf (possibly in the directory below).

C:\Program Files\Apache Group\Apache2\conf

Open it, and look for "<Directory />" string. After you find it, add "ExecCGI" to the options. In my httpd.conf, the Directory group looks like below.

```
<Directory />
Options FollowSymLinks ExecCGI
AllowOverride None
</Directory>
```

Place a copy of the executable hello_vbmcgi.exe (from hello_vbmcgi.cpp) to the cgi-bin directory of apache. In my computer it's the directory below.

C:\Program Files\Apache Group\Apache2\cgi-bin

Rename file hello_vbmcgi.exe to hello_vbmcgi.cgi (one method to do this is by pressing F2 in the windows explorer). Confirm that your apache is properly configured to accept binary cgi programs by seeing in the web browser one of the 2 url's below.

http://localhost/cgi-bin/hello_vbmcgi.cgi

http://127.0.0.1/cgi-bin/hello_vbmcgi.cgi

Here is an useful tip. Select the "release" version for the cgi program, instead of "debug" version. While developing cgi programs, one usually does not debug the program as if it was an ordinary application. That's because the cgi program is executed inside an environment that has its variables configured by the HTTP server. Therefore, the inner VisualC++'s debug features are not used for debugging cgi programs. If you select "release" version for your program, it will

be of much smaller size. To select it, click on menu Build – Set Active Configuration, and select “release” in the dialog box, like shown in the figure below.

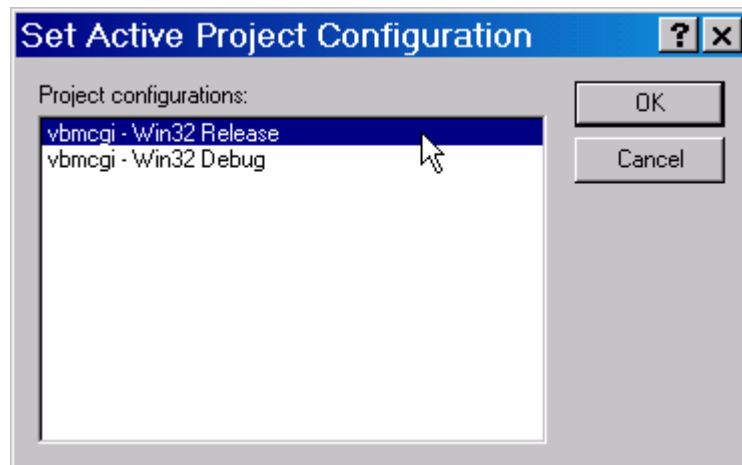
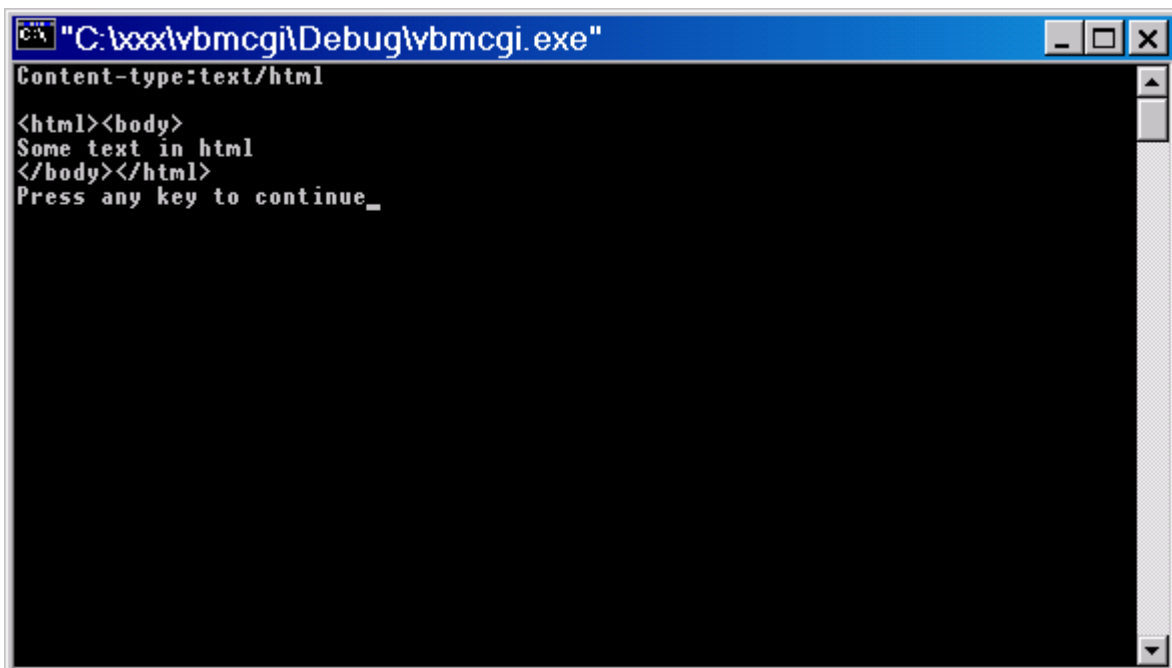


Figure 15: selecting “release” version of a program in Visual C++

If you try to execute it locally from the Visual C++ 6.0 sp5, the output should be like below.



```
"C:\xxx\vbmcgi\Debug\vbmcgi.exe"
Content-type:text/html
<html><body>
Some text in html
</body></html>
Press any key to continue_
```

Figure 16: output of hello_vbmcgi program executed locally from Visual C++

Suppose that you are developing web software to be hosted in some commercial web host service (there are several ones to choose). Most of them use either Windows or Linux as the server operating system. Since your compiler is Visual C++, be sure that the operating system there is Windows. Develop locally your programs using a computer with Windows. When you get satisfactory results, upload the HTML files and the cgi programs to the server. Usually that's done

using ftp. There's no need to upload the C++ source code of the cgi programs, but only the executable files (*.exe or *.cgi).

1.6.2 Unix, Gnu C++ compiler

It is assumed here that you are using some unix compatible system, and you have the gnu C++ compiler properly installed.

Get the distribution file VBMcgi_unix_30.zip from the VBMcgi page. Unzip this file to a working directory. Say this directory is /users/me/vbmcgi. Set attributes of file make.sh to executable. This can be done with the command below.

```
> chmod 755 make.sh
```

Run the shell script make.sh to build the VBMcgi library. This can be done with the command below.

```
> ./make.sh
```

If everything is ok (e.g. gnu compiler g++ is properly installed), you should see the output below.

```
Rebuilding the VBMcgi library 3.0. see http://www.vbmcgi.org
compiling static VBMcgi
binding static VBMcgi
building dynamic VBMcgi
```

Now, let's test the installation of VBMcgi. Go to another working directory. Create a file to test VBMcgi named hello_vbmcgi.cpp, like below.

```
// hello_vbmcgi.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpCompleteHeader();
    cout << "<HTML><body>" << endl;
    cout << "Some text in HTML" << endl;
    cout << "</body></HTML>" << endl;
    return 0;
}
```

Create 2 shell script files to help compile with vbmcgi. One, named vbmcgi.sh is like below.

```
# vbmcgi.sh
echo VBMcgi version3.0 September 4th, 2003
g++ -w $1.cpp -o $1.cgi /users/me/vbmcgi/libvbmcgi.a -I/users/me/vbmcgi $2 $3 $4 $5 $6
```

Other, named vbmcgi_so.sh is like below.

```
# vbmcgi_so.sh
echo VBMcgi version3.0 September 4th, 2003
g++ -w $1.cpp -o $1.cgi /users/me/vbmcgi/vbmcgi.so -I/users/me/vbmcgi $2 $3 $4 $5 $6
```

Make them executable with the command below

```
> chmod 755 vbmcgi.sh
> chmod 755 vbmcgi_so.sh
```

Now you can compile hello_vbmcgi using VBMcgi static, with the command below. File hello_vbmcgi.cgi will be created.

```
> ./vbmcgi.sh hello_vbmcgi
```

If you want to compile `hello_vbmcgi` using VBMcgi dynamic, use the command below. File `hello_vbmcgi` will be created.

```
> ./vbmcgi_so.sh hello_vbmcgi
```

If you want, the cgi program can be run in line command directly.

```
> ./hello_vbmcgi.cgi  
Content-type:text/HTML
```

```
<HTML><body>  
Some text in HTML  
</body></HTML>  
>
```

For the this line command program to work as a cgi program, it should be located in the directory mapped by the http server. A probable name for this directory is `/user/me/public_html/cgi-bin`.

If more than one cpp file is used for a cgi program, the extra files can be used like the example below.

```
> ./vbmcgi.sh hello_vbmcgi extra_file_1.cpp extra_file_2.cpp extra_file_3.cpp
```

Suppose that you are developing web software to be hosted in some commercial web host service (there are several ones to choose). Most of them use either Windows or Linux as the server operating system. Since your compiler is gnu C++ compiler (g++), be sure that the operating system there is some unix (includes Linux). The safest method to assure that everything will work properly is to compile everything in the server's computer. Some commercial web host services provide a shell for the clients. You can log the server's shell using a ssh or telnet client. The server should have the gnu C++ properly installed (something not rare, since this is distributed for free, together with most linux distributions). Upload the HTML files and the source of cgi programs to the server. Usually that's done using ftp. Compile the sources there, and get the executable *.cgi files. After that, if you want, you can erase the C++ source code of the cgi programs. If you're personal working computer has an operating system and C++ compiler compatible with the server's computer, you can compile locally and upload only the *.cgi files.

2) VBMcgi basic tutorial

2.1 Main characteristics and advantages of using VBMcgi

If you choose to use VBMcgi, you will:

- Keep and cultivate culture of using C++, while not missing the web/CGI development!
- Use the widely proven 3-tier software architecture. So, your software will be developed faster, and have good maintainability. There will be no programming in the presentation tier! You can have a team where only the webmaster knows about C++. The webdesigner needs only to know about HTML and other presentation tier technologies. Even if the webdesigner and the webmaster are the same person, there's advantage in isolating the tiers, because the cleanness of the 3-tier architecture helps understand and maintain the software.
- Deliver your web system in binary format. This is the best method to preserve your intellectual rights. Your customer won't be able to easily reverse engineer what you've invested to develop. You can develop web systems to be used in your client's server, and still protect your intellectual rights. No script technology gives you that!
- Easyness of installation. Since the CGI programs are delivered in binary executable format, there's no need to install any specific interpreter in the server. Just place the CGI programs in the proper directory and your system will be running⁵. VBMcgi can be used in any computer using Windows or Unix. There's no problem with VBMcgi if using commercial web hosting firms.

2.2 Hello VBMcgi

If VBMcgi is properly installed to your computer, than you should be able to compile C++ program using this library. The file `hello_vbmcgi.cpp` below is a very simple example that uses VBMcgi. Its usefulness is to confirm that you are using VBMcgi.

```
// hello_vbmcgi.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpCompleteHeader();
    cout << "<HTML><body>" << endl;
```

⁵ The HTTP server must be configured to allow binary CGI programs.

```
cout << "Some text in HTML" << endl;
cout << "</body></HTML>" << endl;
return 0;
}
```

The file `hello_vbmcgi.cpp` includes the header file `vbmcgi.h`, with all necessary declarations. This header file also includes some of the standard headers, like `iostream`. The statement “using namespace std;” is also present inside the file `vbmcgi.h`. The main class to be used is called `VBMcgi`. In the beginning of the main function, there’s an instantiation of the object “cgi” of type “VBMcgi”. After this, the argumentless method “`httpCompleteHeader`” is called. Than some data is sent to the console. At last, the function `main` returns something.

See example of that in <http://www.vbmcgi.org/index.html?p=9>.

2.3 Decoding HTML form

The VBMcgi library has support for HTML form decode. This is done by calling the “`formDecode`” method from the `cgi` object (of type `VBMcgi`). This method reads the `QUERY_STRING`, separates the form names and its contents, reverses the escape sequence, and place the result to an `cgi`’s inner attribute, which is a linked list (of a pair of strings). To easily allow access to the form variables, there’s the method “`getVarContent`” (that should be called after the method “`formDecode`”).

The method `formDecode` encapsulates difference of `get` and `post` method. So, the developer does not need to care about this, nor recompile the CGI program if change method `get` to `post` or vice-versa.

It’s strongly recommended that you do not treat string in the C style (array of `char`). There are many bugs that can happen if you do that. The recommended method to treat strings is to use the `VBString` class (already inside VBMcgi library). See details about `VBString` class and other `VBLib` features seeing its manual at the URL below.

<http://www.vbmcgi.org/vbllib>

2.4 Multiply table with parameter

Let’s see a sample of how to decode the HTML form using VBMcgi. To do that, you have to call the “`formDecode`” method, and latter get the content of a form variable with the method “`getVarContent`”, passing the form field’s name, and receiving as return a string (`VBString`) with the field contents. Convert the string to integer with the `atoi` function of the standard library, and do the loop to calculate the multiply table using the number form the HTML form as parameter.

```
// multiply_table_with_parameter.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpCompleteHeader();
    cgi.formDecode(); // decode form vars
```

```

cout << "<html><body>" << endl;
VbString mt_str = cgi.getVarContent("mtable"); // get mtable as string
int multiply_table = atoi(mt_str); // convert to int
for (int i=1 ; i <= 10 ; i++)
    cout << i*multiply_table << "<br>" << endl;
cout << "</body></html>" << endl;
return 0;
}

```

Suppose one answers “8” to the HTML form of the multiply table with parameter. The output of it, in the client’s web browser, will be like shown below. Remark the query string showing the data of the HTML form in the URL.

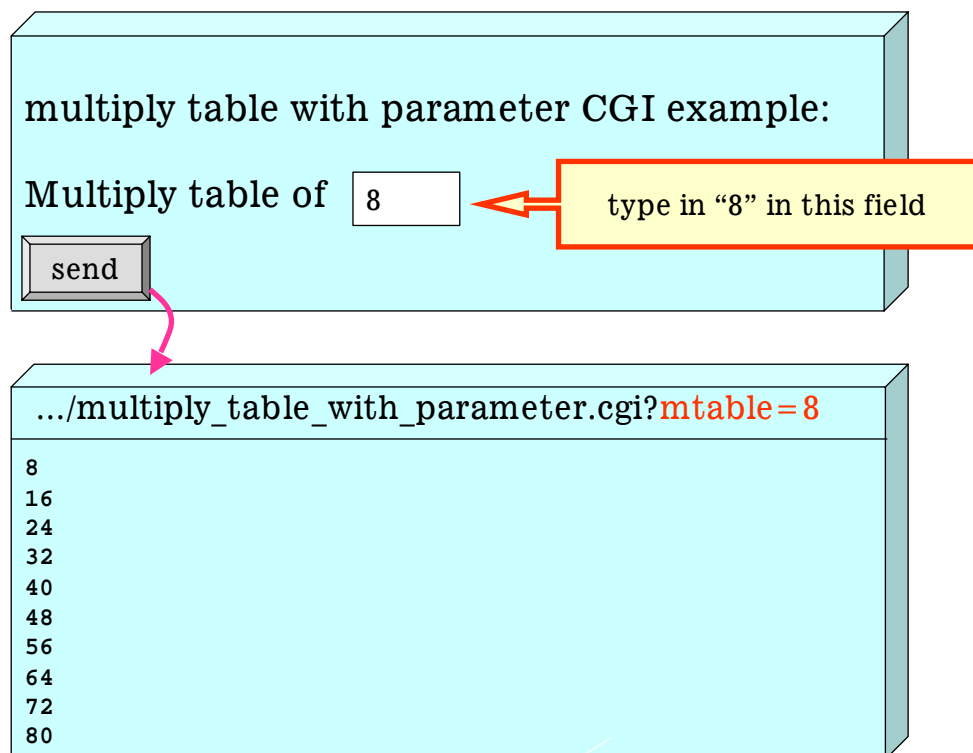


Figure 17: answering “8” (as an example) to the HTML form, to calculate the multiply table

See this example in <http://www.vbmcgi.org/index.chtml?p=10>.

2.5 String parameters

Let’s see another example of how to decode the HTML form using VBMcgi. Let the HTML code below.

```

<form action="string_parameter.cgi"
  method="get">
<table border="0">
<tr>
<td align="right">Name:</td>
<td ><input type="text" name="s_name"
  size="20"></td>
</tr>
<tr>
<td align="right">Telephone:</td>
<td><input type="text"

```

```

name="s_telephone" size="20"></td>
</tr>
<tr>
<td align="right">Zip code:</td>
<td ><input type="text" name="s_zip" size="20"></td>
</tr>
<tr>
<td align="right">email:</td>
<td ><input type="text" name="s_email" size="20"></td>
</tr>
</table>
<input type="submit" value="send">
</form>

```

This will produce a HTML form like the figure below.



Figure 18: HTML form (with some data filled in)

To decode this form, one can use the C++ code listed below.

```

// string_parameter.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpCompleteHeader();
    cgi.formDecode();
    VBString str_name = cgi.getVarContent("s_name");
    VBString str_telephone = cgi.getVarContent("s_telephone");
    VBString str_zip = cgi.getVarContent("s_zip");
    VBString str_email = cgi.getVarContent("s_email");

    cout << "<html><body>" << endl;
    cout << "Your name is: <b>" << str_name << "</b><br>" << endl;
    cout << "Your telephone is: <b>" << str_telephone << "</b><br>" << endl;
    cout << "Your zip code is: <b>" << str_zip << "</b><br>" << endl;
    cout << "Your email is: <b>" << str_email << "</b><br>" << endl;
    cout << "</body></html>" << endl;
    return 0;
}

```

The output in the client's browser will be like below.

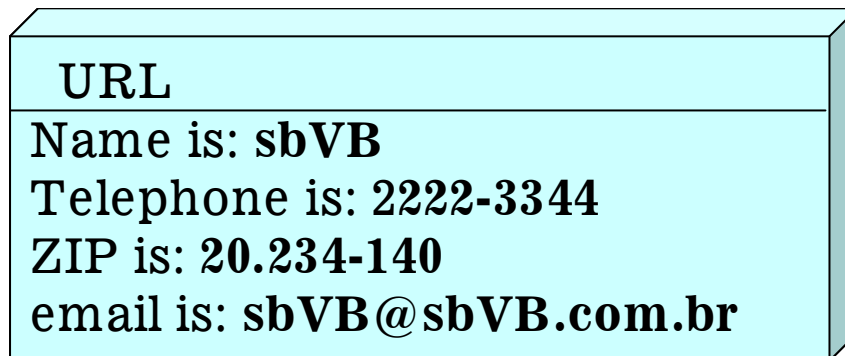


Figure 19: Output of string_parameter.cgi in client's browser

For the data typed in, the client's browser will have the URL below (a single long line).

```
http://www.vbmcgi.org/string_parameter.cgi?s_name=sbVB&s_telephone=2222-3344&s_zip=20.234-140&s_email=sbVB@sbVB.com.br
```

The examples so far show how to use VBMcgi to decode the HTML form, but still do not isolate tiers (the main purpose of existence of VBMcgi).

See this example in <http://www.vbmcgi.org/index.html?p=11>.

2.6 "3-tier" web software architecture

The software should be developed in such architecture that small changes in the requirements lead to small effort. The keyword, and the utmost goal is have good maintainability. To develop web/CGI software using 3-tier architecture is to use an architecture that has proven to provide good maintainability.

The tier 1 is the presentation tier. In web/CGI software, this tier means the work of the Webdesigner, that is, the one that cares about the site's design. Knows well HTML, DHTML, JavaScript. He uses tools like DreamWeaver and FrontPage.

The tier 2 is the tier for for implementation of business rules. In web/CGI software, this tier means the work of the Webmaster, that is, the one that cares about the implementation of the site's interactivity. He knows about programming and often about data base. Uses languages like C++ (and VBMcgi), PHP, ASP, JSP & Java, Perl, Cold Fusion, etc.

The tier 3 is the database tier. In web/CGI software, this tier means the work of the Webdesigner, that is, the DBA (data base administrator) that cares about data modeling. Knows about SQL, relational databases, etc. Uses software like Oracle, SQL server, MySql, DB2, PostgreSQL, etc.

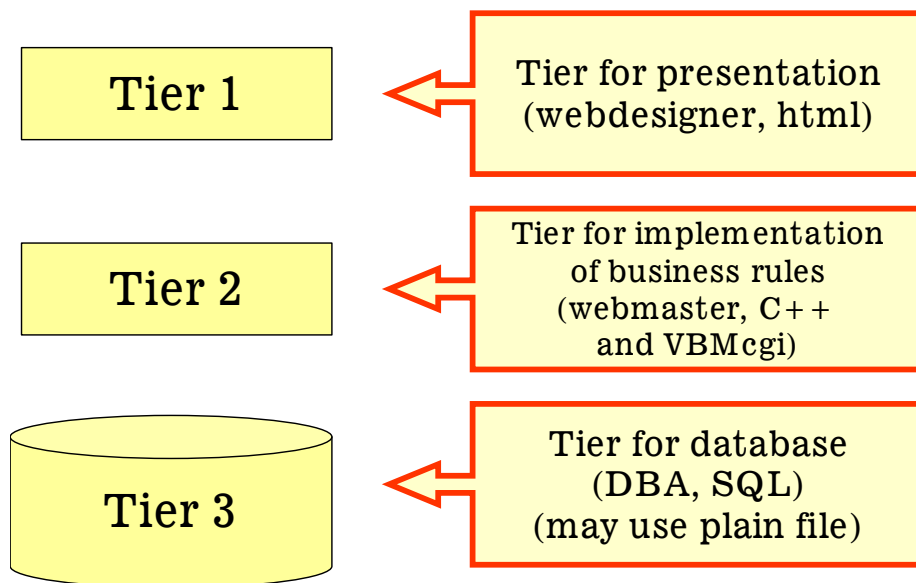


Figure 20: The 3 tiers of the software system

Remark that not to isolate tiers does not mean that the software won't work. It only means that to maintain it can take larger effort. If the software system is big complex and/or has its requirements ever changing, this is good reason for one to choose 3-tier architecture.

Some of the script languages that are widely used for web/CGI software development, such as PHP and Microsoft's ASP, mix tier 1 and tier 2. Let a simple page in ASP, as shown below. It is a page with HTML code, and when necessary, the "<%>" opens the ASP language, to be closed by the "<%>". So, to change the page's design, HTML code should be changed in this file. To change something in the business rules, the ASP part of it should be changed, in the same file. That is, it's difficult to have both webmaster and webdesigned working in this page at the same time. Still, since they both change this file, a version conflict might arise.

```
<!-- page.asp -->
<html><body>
<t1>Title</t1>
<%
Dim famname(5)
famname(0) = "Marcos"
famname(1) = "Ana"
famname(2) = "Camila"
famname(3) = "Mateus"
famname(4) = "Marcia"
famname(5) = "Villas"
For i = 0 to 5
  response.write(famname(i) & "<br>")
Next
%>
</body></html>
```

The code above will be processed in the server, and the client will see like below.

```
<!-- page.asp -->
```

```
<html><body>
<t1>Title</t1>
Marcos<br>
Ana<br>
Camila<br>
Mateus<br>
Marcia<br>
Villas<br>
</body></html>
```

2.7 VBMcgi golden rule and tier isolation

The golden rule of VBMcgi illustrates its philosophy.

To isolate the work of webdesigner (tier 1, for presentation) and webmaster (tier 2, for implementation of business rules)

Figure 21: VBMcgi golden rule

VBMcgi provides the isolation of tier 1 and tier 2 with the two features below.

- string change feature
- call function feature

2.8 String change feature

The string change feature is this. The VBMcgi object registers any number of pairs of strings by using the method “addBySource”. A string pair has a search string and a replace string. There’s also the “out” method, that receives an HTML file as argument. The “out” method will read the HTML file, and line per line will search and replace each string pair previously registered. The resulting line will be sent to the console (thus to the client’s web browser).

This provides tier isolation, because the webmaster (that writes C++ code) does not have to know any of the complexities of the HTML file. A very complex HTML code can be used. The HTML authoring tool that the webdesigner uses do not impact the work of the webmaster. The HTML file can be changed without communicating that fact to the webmaster. The webmaster can change the search/replace data without communicating that fact to the webdesigner. If needed, the CGI programs and the HTML data can be stored in different computers.

The search string should be improbable char sequences to avoid unwanted changes. Here is an example of good char sequences for search string: s_name, s_telephone. Here is an example of bad char sequences for search string: name, telephone. This is a bad char sequence because if the word “name” exists in the HTML data as a normal data text, it will be subject of string change feature as well. If one attempts to place a very common char sequence in the search string,

such as “a”, than the string change feature will change all “a” occurrences to the replace string, resulting in something strange.

The registered search strings should be left-unique. Here is an example of non-left-unique strings: `s_client`, `s_clientTelephone`. In this case, one recommended modification will be to change the sequences to `s_clientName`, `s_clientTelephone`.

The string change feature does not have inner intelligence. It’s a dumb and fast feature that searches and replaces strings in the client’s web browser. All intelligence should be develop by carefully selecting what to change using this feature.

Below there’s an example of using the string change feature. In this example, both search and replace strings are hard-coded. As you can see, this example is easily adaptable to one wher the replace strings come from something soft-coded, such as a database, a file, an HTML form, etc.

```
// string_change_feature.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;

    // add some string exchanges
    cgi.addBySource("s_name","sbVB");
    cgi.addBySource("s_telephone","2222-3344");
    cgi.addBySource("s_zip","20.234-140");
    cgi.addBySource("s_email","sbvb@sbvb.com.br");

    // open html file, execute string changes and put to browser
    cgi.out("person.html");
    return 0;
}
```

Consider a snippet of file `person.html` to be like below.

```
<!-- person.html -->
Name is: <b>s_name</b><br>
Telephone is: <b>s_telephone</b><br>
ZIP is: <b>s_zip</b><br>
email is: <b>s_email</b><p>
```

In this example, the output is 100% predictable, and will be like below.

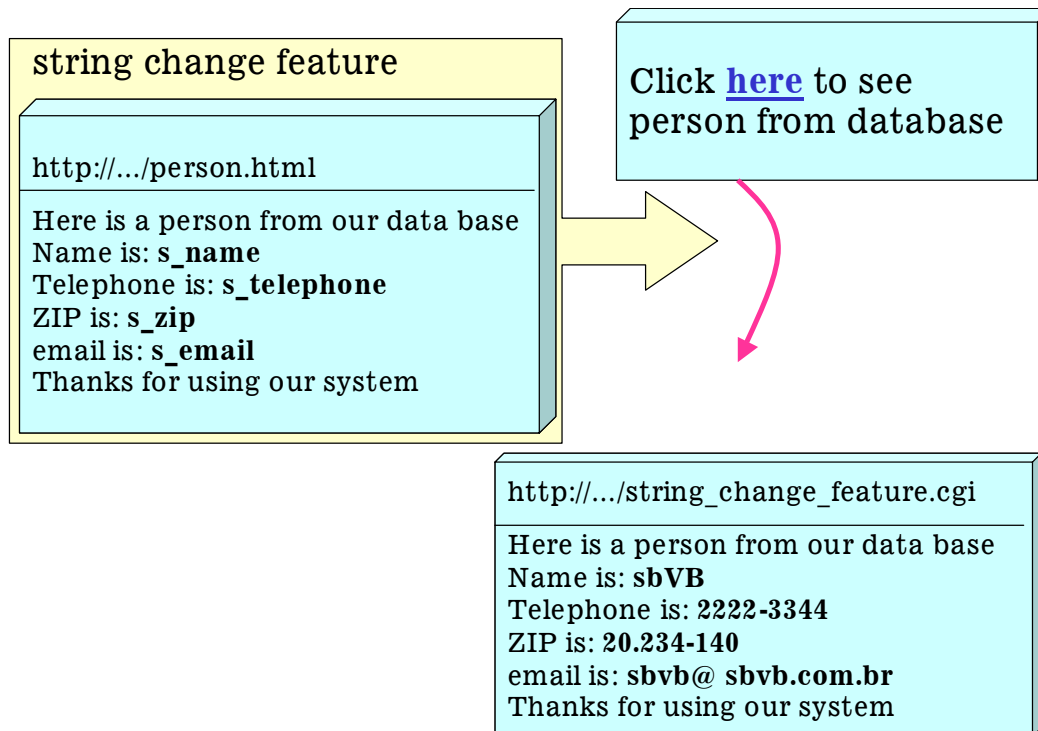


Figure 22: String change feature

See this example in <http://www.vbmcgi.org/index.html?p=12>.

2.9 formDecode adds variables

VBMcgi has 2 linked lists of pairs of strings as attributes. One to store the decodification the HTML form and other to store the pairs for the string change feature. There's a possibly useful feature that automatically places data from the HTML form to the string change list. This is done by setting the Boolean parameter of method formDecode to "true".

In the example below, the "string parameters" example is adapted by using the "string change feature" and the "formDecode adds variables".

```
// formdecode.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.formDecode(true); // decode form variables and add variables
    // to the "string change feature"

    // cgi.formDecode(false); // same as cgi.formDecode();
    // if the boolean constant "false" is added
    // as parameter of formDecore, the variables are NOT added
    // to the "string change feature"

    // open html file, execute string changes and put to browser
    cgi.out("person.html");
    return 0;
}
```

See this example in <http://www.vbmcgi.org/index.html?p=13>.

2.10 Complete form reference

This section shows the all HTML form elements, how to define them in HTML, and how to decode them using VBMcgi.

2.10.1 textbox

The HTML code below produces a form with a textbox.

```
<HTML><body>
textbox:<br>
<form action="formdecode.cgi" method="get">
Name:<input type="text" name="fieldName" size="20"><br>
<input type="submit" value="send">
</form>
</body></HTML>
```

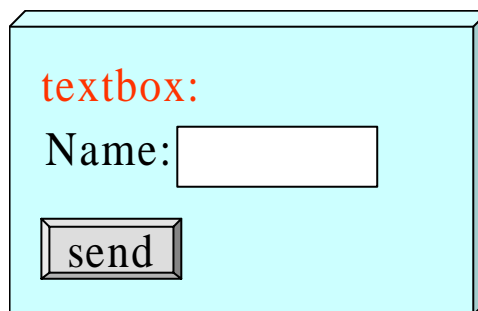


Figure 23: HTML textbox seen in browser

A CGI program to decode the textbox, getting the string of the content of the field, is shown below. The content of this field is necessarily a string. If you want this data to be a number, you should call a function to convert it (such as `atoi` or `atof`).

```
// formdecode.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.formDecode();
    VBString name = cgi.getVarContent("fieldName");
    // continue cgi code
}
```

2.10.2 textarea

The HTML code below produces a form with a textarea. Usually the textarea is chosen when asking for something that is big. For instance: "Enter your opinion about something".

```
<HTML><body>
textarea (scrolling text box):<br>
<form action="formdecode.cgi" method="get">
Enter your opinion about something:<br>
<textarea name="opinion" rows="4" cols="55">Initial value</textarea>
<input type="submit" value="send">
</form>
</body></HTML>
```

textarea (scrolling text box):
Enter your opinion about something:
Initial value
send

Figure 24: HTML textarea seen in browser

The CGI program to decode the form's textarea is much similar to the one used to decode the textbox. The content can be very big (if post method is used instead of get method, the content length is virtually non-limited). A VBString can receive data of any size. Even though the content is only one string, it can contain the char '\n' (return) in it. The developer should treat this if wanted.

```
// formdecode.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    VBString str = cgi.getVarContent("opinion");
    // continue cgi code
}
```

2.10.3 checkbox (caixa de checagem)

The HTML code below produces a form with checkbox fields. Each field has an independant Boolean value. Remark that each field has a different name, and each one has a value associated with it (in this case is "ON").

```
<HTML><body>
checkbox:<br>
<form action="formdecode.cgi" method="get">
You like:<br>
<input type="checkbox" name="like_sports" checked value="ON">Sports<br>
<input type="checkbox" name="like_books" checked value="ON">Books<br>
<input type="checkbox" name="like_news" checked value="ON">News<br>
<input type="submit" value="send">
</form>
</body></HTML>
```

checkbox:
You like:
 Sports
 Books
 News
send

Figure 25: HTML checkbox seen in browser

VBMcgi has a specific method to decode checkbox, returning a Boolean directly. The method is called `getCheckBox`. The first argument is the checkbox name, and the second one is the value (string) associated with it in the HTML. It returns true if the checkbox is checked. An example code to decode it is shown below.

```
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    bool b_sports = cgi.getCheckBox("like_sports","ON");
    bool b_books  = cgi.getCheckBox("like_books","ON");
    bool b_news   = cgi.getCheckBox("like_news","ON");
    // continue cgi code
}
```

2.10.4 radio button

The radio button element in an HTML form is an item that lives in a group. One HTML form can have any number of radio button groups, each group having any number of items. Within a group, the selection of an item makes all other items, that is: exclusive selection.

In the example below, there's one group of radio button named "after_dinner". VBMcgi decodes it using the method `getVarContent`, that returns the string associated with the option selected. In the example, the options have "1", "2", "3" or "4" as associated strings. This is convenient, because it can be converted to integer.

```
<HTML><body>
radio button:<br>
<form action="formdecode.cgi" method="get">
After dinner you:<br>
<input type="radio" name="after_dinner" value="1">Brush your teeth<br>
<input type="radio" name="after_dinner" value="2">Watch TV<br>
<input type="radio" name="after_dinner" value="3" checked>Study<br>
<input type="radio" name="after_dinner" value="4">Read a book
</form>
</body></HTML>
```



radio button:
After dinner you:
 Brush your teeth
 Watch TV
 Study
 Read a book

Figure 26: HTML radio button seen in browser

The CGI program below to decode the form's radio button and gets the string associated with the choice selected, convert it into int, and use as index to an

array of `char*`. So, there's a message hardcoded in the C++ code associated with the choice from the HTML form.

```
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.formDecode(); // decode form variables
    VBString n_str = cgi.getVarContent("after_dinner");
    int n = atoi(n_str);
    const char *stringsAfterDinner[4] = {
        "brush the teeth",
        "give TV a good watch",
        "go study for a while",
        "read the favorite book"
    };
    // continue cgi code
    cgi.httpCompleteHeader();
    cout << stringsAfterDinner[n-1] << endl;
    return 0;
}
```

2.10.5 drop down

The element “drop down” is also known as “list menu”. Like the radio button, there's a string associated to each option. VBMcgi's method `getVarContent` return the string associated with the option selected.

```
<HTML><body>
drow down:<br>
<form action="formdecode.cgi" method="get">
Your favorite color is: <br>
<select name="favorite_color" size="1">
<option value="1">Green</option>
<option value="2">Red</option>
<option value="3">Blue</option>
</select>
</form>
</body></HTML>
```

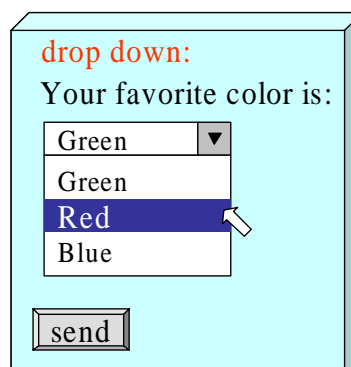


Figure 27: HTML drop down seen in browser

The CGI program below to decode the form's drop down and gets the string associated with the choice selected, convert it into int, and use as index to an array of `char*`. So, there's a message hardcoded in the C++ code associated with the choice from the HTML form.

```
#include "vbmcgi.h"
int main()
```

```
{
  VBMcgi cgi;
  cgi.formDecode(); // decode form variables
  VBString n_str = cgi.getVarContent("favorite_color");
  int n = atoi(n_str);
  const char *stringsForColor[3] = {
    "the color green",
    "red, a hot color",
    "blue, the color of the sky"
  };
  // continue cgi code
  cgi.httpCompleteHeader();
  cout << stringsForColor[n-1] << endl;
  return 0;
}
```

See the complete form decode example in the URL below.

<http://www.vbmcgi.org/index.shtml?p=14>.

2.1.1 Call function feature

The call function feature is the second and more powerful VBMcgi feature to allow isolation of tier 1 (presentation) and tier 2 (business rules). This is to be used when the string change feature is not good enough.

The idea is to allow the webmaster to call a function inside the HTML code. The “out” method looks for a sequence of chars like below.

VBMCGI_CALL(userFunction)

If found, the char sequence beginning with ‘V’ and ending with ‘)’ will not be sent to the console (that is, the client’s web browser). In its place, the out method will call a C++ global function called “userFunction”, and whatever this function sends to the console will be sent to the client’s web browser. The call function feature is well suited for web pages with complex web design, but having parts that are relatively simple (like tables from a database). The call function feature works with some similarity to the SSI (Server Side Includes) feature that sometimes is configured to exist in the HTTP server, but the call function feature gives more power to the webmaster.

Optionally, if using dynamic link (*.dll in Windows or *.so in Unix), the webmaster can use the same function in more than one CGI program. By updating the dll or so, all CGIs will be affected, without need to recompilation. The webmaster can develop a function, and let an unlimited number of CGIs be developed using it. If it comes the time to maintain the software, by changing the dll or so, all CGIs will be affected.

The userFunction is a global C++ function to be used as a “call back” from the VBMcgi object. This function must have a fixed prototype, like below (even if the arguments are not to be used). To register the userFunction in the VBMcgi object, the method addFunction is used.

```
// prototype of a VBMcgi callback function
void userFunction(VBMcgi & userCgi, void *p);
```

Let the example where the webmaster wants to develop a callback function to exhibit the multiply table of a parameter. This function is to be called multiplyTable. The CGI program will be like below.

```
// call_function.cpp
#include "vbmcgi.h"
void multiplyTable(VBMcgi & userCgi, void *p) {
    // get s_mtable as string
    VBString mt_str = userCgi.getVarContent("s_mtable");
    int multiply_table = atoi(mt_str); // convert to int
    for (int i=1 ; i <= 10 ; i++)
        cout << i*multiply_table << "<br>" << endl;
}
int main()
{
    VBMcgi cgi;
    // decode form vars and add to "string change feature
    cgi.formDecode(true);
    cgi.addFunction(multiplyTable);
    cgi.out("showMultiplyTable.html");
    return 0;
}
```

The HTML code to call this CGI program will be like below.

```
<!-- HTML snippet to call "call_function.cgi" -->
<form action="call_function.cgi" method="get">
Multiply table of
<input type="text" name="s_mtable"><br>
<input type="submit" value="send">
</form>
```

There should be a file called showMultiplyTable.html to be the presentation tier for this CGI. A snippet of its HTML code is shown below.

```
<!-- HTML snippet of showMultiplyTable.html -->
The multiply table of <b>s_mtable</b> is shown below<br>
VBMCGI_CALL(multiplyTable)
```

In the figure below, there's a pictorial illustration of the effect of the call function feature.

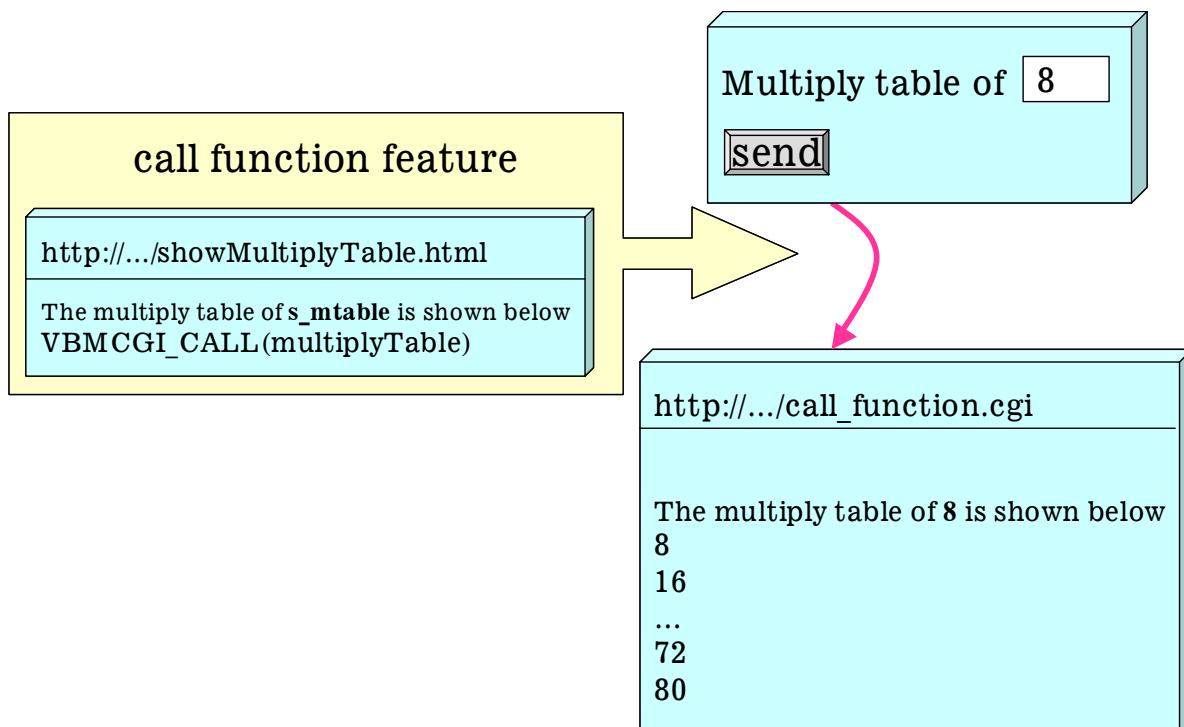


Figure 28: Example of call function feature

It should be remarked that the VBMcgi object whose lvalue is passed as the first parameter of the callback function is the same VBMcgi object that the function was registered. So, if the method formDecode was called by the VBMcgi object outside the callback function, the getVarContent method can be used inside the callback function. Similarly, if one uses the method addBySource inside the callback function, that will affect the VBMcgi object outside the callback function.

See the complete call function example in the URL below.

<http://www.vbmcgi.org/index.shtml?p=15>.

Here is an illustrative real world example of usage of the call function feature. Let be a web site for commerce. Assume that there's a requirement to add a "shopping car" feature to the site. So, there's a database table properly defined somewhere with the data that the current client intends to purchase, but hasn't checked out yet. The webmaster can write a callback function to display the shopping car table, and instruct the webdesigner to add this table to wherever is needed. Say this callback function is named "shoppingCar". The HTML design of the pages that are to display the shopping car are something that is defined and developed in the presentation tier, in HTML, and has nothing to do with the webmaster work. The webdesigner should write VBMCGI_CALL(shoppingCar) in the place he wants to see the table.

Here is another illustrative real world example of usage of the call function feature. Let be a web site for real state advertisements. Suppose that the real

state data has attributes like quarter, number of rooms, area, city, state, etc. Suppose that the user selects the drop down to look for real state in state A. By knowing the state, the database will place the city drop down of the cities of that state. The user then selects the city, and the quarters of that city (stored in the database) will appear in the next drop down. The webmaster can write a callback function named “cities”, that display the drop down with the cities of a given state, and another callback function named “quarters”, that display the drop down of the quarters of a given pair of state and city. After choosing the state, city and quarter, if there’s no 3-room real state available in this area, this option should not be made available to the drop down. So, there could be yet another callback function named “rooms”, that display the drop down of existing real states of the database for a given state, city and quarter. If developing the web software like this, if one registers a 3-room real state in the given state, city and quarter, the option will appear to the HTML without need to recompile any CGI program.

2.11.1 Call function feature passing parameters from C++

For many reasons, one might need to pass parameters from C++, to the callback function. There’s already a reference to the VBMcgi object as the first parameter of the callback function. But how to pass extra parameters?

One can not know what are the parameters that are to be passed to the callback function, and the callback function must have fixed prototype. So, the extra parameters are passed using the second parameter with the opaque pointer (void*). All the parameters that the webmaster wants to pass to the callback function should be in a single object. This do not impose limitations to what or how many parameters are to be passed. The pointer to a single object is passed to the callback function. That’s enough for a general purpose passing of parameters from C++ to the callback function. Inside the callback function, the pointer to the object is stored in the opaque pointer p (void*), that needs to be type casted to a pointer to the actual object passed. To use the “passing parameters from C++” of the call function feature, when registering the callback function, use the method addFunctionObj, with 2 parameters: the function and the object to be passed.

In the example below, there’s a userClass with 3 attributes. Inside the main function, the userClass instantiates the object userObject. In the main function, this object is filled with data. The callback function multiplyTable is registered using the addFunctionObj method, passing the multiplyTable function and the object userObject. Inside this version of the multiplyTable callback function, a pointer to userClass, named p_userObject, is instantiated and the content of opaque pointer p is type casted and copied to it. The attributes of the object are accessed inside the callback function by referencing the p_userObject, with the “->” operator.

```
// call_function_with_parameters.cpp
#include "vbmcgi.h"
```

```
struct userClass
{
    int m_i;
    double m_d;
    VBString m_str;
};

void multiplyTable(VBMcgi & userCgi, void *p)
{
    userClass *p_userClass;
    // make type casted p point to pointer to user class
    p_userClass = (userClass*)p;

    // put some HTML to console,
    // using the values that were received as parameters
    cout << "<hr>" << endl;
    cout << "The value of i is " << p_userClass->m_i << "<br>" << endl;
    cout << "The value of d is " << p_userClass->m_d << "<br>" << endl;
    cout << "The value of str is " << p_userClass->m_str << endl;
    cout << "<hr>" << endl;

    // now do the standard multiply table
    // get mtable as string
    VBString mt_str = userCgi.getVarContent("s_mtable");

    int multiply_table = atoi(mt_str); // convert to int
    for (int i=1 ; i <= 10 ; i++)
        cout << i*multiply_table << "<br>" << endl;
}

int main()
{
    userClass userObject; // instantiate a user object
    // fill with some data
    userObject.m_d = 2.2;
    userObject.m_i = 4;
    userObject.m_str = "abc";

    VBMcgi cgi;
    cgi.formDecode(true); // decode form vars and add to "string change"

    // register the function and the object
    cgi.addFunctionObj(multiplyTable,userObject);

    // call the HTML out method
    cgi.out("showMultiplyTable.html");
    return 0;
}
```

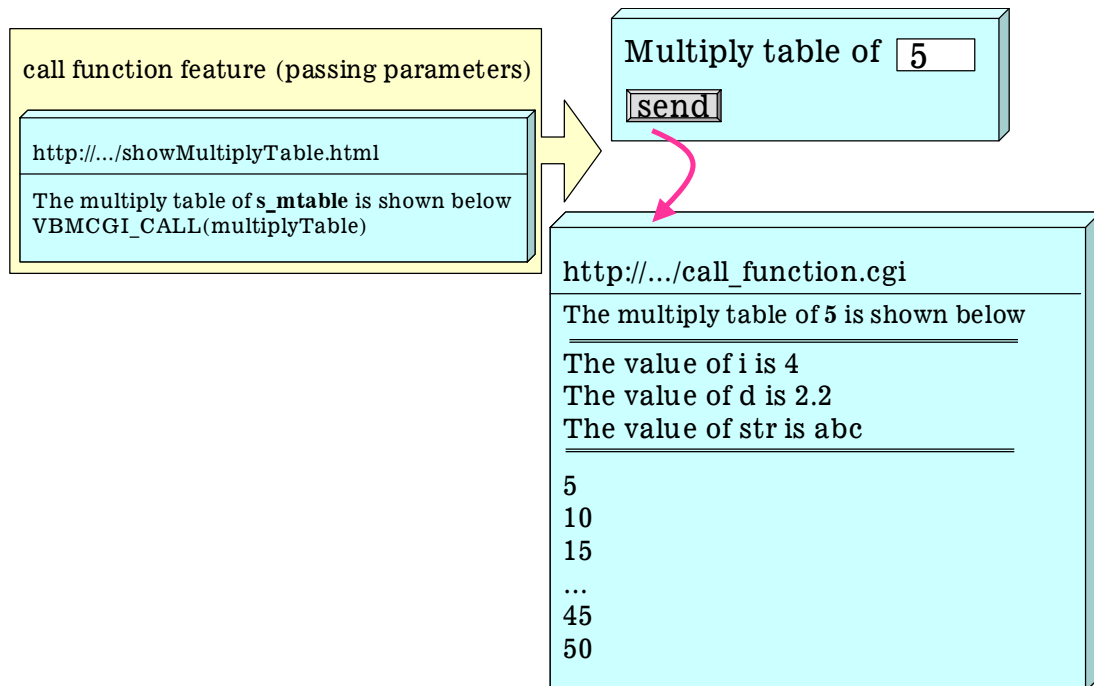


Figure 29: Example of call function feature passing parameters from C++

See the complete call function with parameters from C++ example in the URL below. <http://www.vbmcgi.org/index.html?p=16>.

2.11.2 Call function feature passing parameters from HTML

Another possibility of the callback function is to receive parameters directly from HTML. This is done by adding extra parameters after the callback function name in the char sequence `VBMCGI_CALL`, separated by comma `,`. For instance, if one wants to pass HTML parameters `'1'`, `'2'` and `'stringParameter'`, to the callback function `'userFun'`, the webdesigner should write in his HTML code like below.

```
VBMCGI_CALL(userFun,1,2,stringParameter)
```

One advantage of the HTML parameters is that to change them do not require to recompile the CGI programs. The code below is a CGI program with a `userFun` callback function that receives HTML parameters. The number of parameters is given by the method `getHtmlParameterNumber`, and the parameters (as strings) are returned by the method `getHtmlParameter`, given the argument of the number of the HTML parameter.

```
// call_function_with_html_parameters.cpp
#include "vbmcgi.h"

void userFun(VBMcgi & cgi, void *p)
{
    cout << "inside the user fun<br>" << endl;
    unsigned number_of_parameters = cgi.getHtmlParameterNumber();
    for (unsigned i = 0 ; i < number_of_parameters ; i++)
    {
        cout << "Parameter[" << i << "]=" <<
```

```

        cgi.getHtmlParameter(i) << "<br>";
        if (i < number_of_parameters-1) cout << endl;
    }
}

int main()
{
    VBMcgi cgi;
    cgi.addFunction(userFun);
    cgi.out("test.html");
    return 0;
}

```

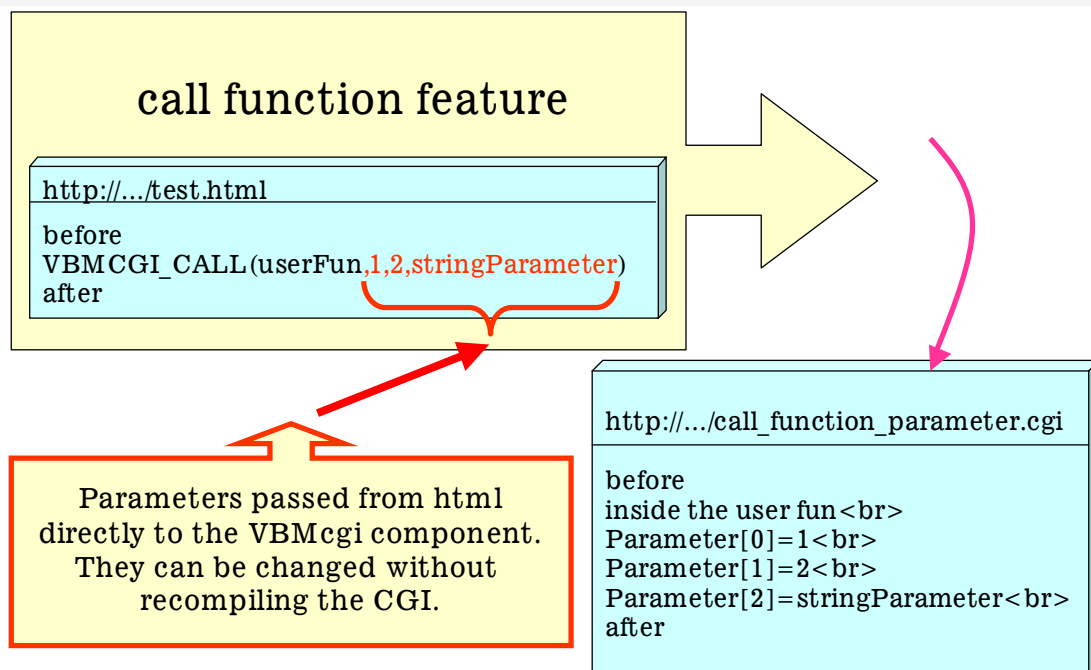


Figure 30: Example of call function feature passing parameters from HTML

See the complete call function with parameters from HTML example in the URL below. <http://www.vbmcgi.org/index.shtml?p=17>.

2.12 Cookies

The cookies are an important feature of web software development, unfortunately not very well understood. There's a lot to tell about cookies. This section is not intended to be a definitive reference about cookies, but an introduction about the basic features about cookies, and how to use them with C++ and VBMcgi.

For short, a cookie is a pair of strings and a few other attributes. The cookie is placed by the HTTP server to the HTTP client. By doing so, the client can be identified later (otherwise, it wouldn't be possible to identify the client). So, the cookie brings to the web system the concept of "state". Suppose any web software that the user logs in. There's some kind of sensible information that only a logged user should be allowed to access. That's the case for WebBanking. The cookies store the state that a user is logged.

An imaginative software developer can figure out several ways to take advantage of the concept of cookie. Many companies use cookies to identify the customer and offer customized content. In Amazon.com, if you buy a book about C++, the next time you go there there's suggestion for you to buy books of related subject. A web site can be developed with links all being as CGI programs, and if the cookie identifies the user, a profile of what the user does can be stored. So, one can have a "profiler site", that if the user often clicks on "travel" related links, the database creates a profile that this user likes "travel" subjects. The next step could be to place "travel" related banners to him.

There are 2 types of cookies:

- end-of-session (in memory)
- persistent (become file)

The end-of-session cookie is stored in the client's web memory. It does not become a file in the file system. This kind of cookie is used often in systems to log for a while, like WebBanking. The persistent cookie creates a file in the client. So, the client can turn off the computer and turn on the other day and still be identified by the web system.

An example of usage of persistent cookies is in a poll. The webmaster wants the voters to vote only once. But how to control that? If you send a cookie after the voting, you can identify those that already voted. If an user votes and erases all cookies (something that the webmaster can't prohibit the client to do), he would be able to vote again.

The cookies exist inside the HTTP header. A CGI program can send any number of cookies to the web client. A simple CGI program that sends 3 cookies will send the first line of the HTTP header "Content-type:text/html<return>, 3 lines with cookies (each followed by <return>), and than a final <return>. The 2 consecutive <return> chars make the HTTP header to finish. In the console, the output of a program that sends 3 cookies should look like below.

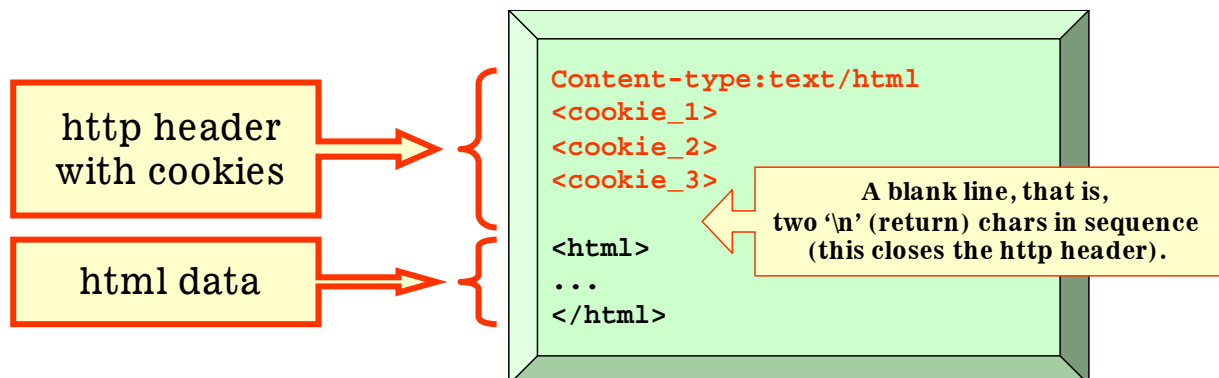


Figure 31: Cookies being sent inside the HTTP header

Using C++ and VBMcgi, one can use the code below to produce a program that sends 3 fixed cookies to the client. Remark that instead of using the method `httpCompleteHeader`, the method `httpHeader` is used. This method places the

almost complete HTTP header, but without the last <return>. So, technically the HTTP header does not end after this line. If you add manually another <return> to the console just after the method `httpHeader`, than you do exactly that the `httpCompleteHeader` does. But in this example, the cookies are being sent before closing the HTTP header. So send a cookie, first define the cookie name and cookie value, with the method `setCookieNameValue`. Set than the expiration with the method `setCookieExpires`. Than, send the cookie with `sendCookie`. Do that 3 times, and there will be 3 cookies. To close the HTTP header, just add the code “`cout << endl;`” After this, use the “`out`” method to invoke the HTML file in the presentation tier.

```
// cookies_3.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpHeader(); // begin the http header
    // set parameters of cookie 1
    cgi.setCookieNameValue("cookie name 1", "cookie value 1");
    cgi.setCookieExpires("end-of-session");
    cgi.sendCookie(); // send cookie to browser
    // set parameters of cookie 2
    cgi.setCookieNameValue("cookie name 2", "cookie value 2");
    cgi.setCookieExpires("end-of-session");
    cgi.sendCookie(); // send cookie to browser
    // set parameters of cookie 3
    cgi.setCookieNameValue("cookie name 3", "cookie value 3");
    cgi.setCookieExpires("end-of-session");
    cgi.sendCookie(); // send cookie to browser
    cout << endl; // end the http header
    cgi.out("test.html"); // html code to the console
    return 0;
}
```

The output of the CGI program is shown below.

```
Content-type:text/HTML
Set-Cookie: cookie name 1=cookie value 1;expires=
Set-Cookie: cookie name 2=cookie value 2;expires=
Set-Cookie: cookie name 3=cookie value 3;expires=

<HTML>
test.html
</HTML>
```

2.12.1 Configuring the browser to see the cookies

Very often the client's web browser is configured to “swallow” the cookie without any prompt. But it can be useful to have a prompt every time a cookie comes in. To do that, it is necessary to properly configure the client's web browser. See how to configure your browser to prompt for a cookie in the URL below.

<http://www.vbmcgi.org/youReceivedACookie.html>

See an example of receiving 3 cookies in the URL below.

<http://www.vbmcgi.org/index.chtml?p=18>.

2.12.2 Persistent cookies

If the expiration parameter of the cookie is not “end-of-session”, but a date-time string, than the cookie will become a persistent cookie, and a file in the web client’s computer. The cookie will be valid until the date-time the developer placed in it. There’s no infinite duration cookie. But the date-time can be placed in a long future, so for practical purposes its infinite duration.

The directory where the file of the cookie is stored in the client’s computer is something that varies according with operating system, web client, etc.

An example of CGI program that produces persistent cookie is shown below.

```
// persistent_cookie.cpp
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.httpHeader(); // begin the http header
    // set parameters of cookie
    cgi.setCookieNameValue("cookie name", "cookie value");
    cgi.setCookieExpires("31-Mar-2002 23:59:59 GMT");
    cgi.sendCookie(); // send cookie to browser
    cout << endl; // end the http header
    cgi.out("test.html"); // html code to the console
    return 0;
}
```

The output of the CGI program is shown below.

```
Content-type:text/HTML
Set-Cookie: cookie name=cookie value;expires=31-Mar-2002 23:59:59 GMT

<HTML>
test.html
</HTML>
```

2.12.3 The cookie’s domain

The cookie has an attribute that is the domain that generated it. So, if the site domain_A.com sends the user a cookie with cookieName field having “userID”, and domain_B.com do the same thing, there will be no confusion to detect those.

A web page can contain aggregate elements of other pages. For instance, a web page under the domain_A.com can show an element that is a figure from domain_B.com. If the user clicks in the figure, and the figure is associated to a CGI, this CGI will have the domain attribute of domain_B.com, even though the user looks like navigating in the domain_A.com

2.12.4 Login to web using cookies

By using cookies, one can create a web site that allows a user to login to it. Such system is of very much use for many applications. Below, there is a simple web system that requires login for the user.

A web system that allows login begins with an open login page (in this case with the file login_using_cookies.html). A form asks for login and password, and

checks somehow to see if the login is to be accepted or not. If not accepted, direct the user to a page that tells him that the login was not correct, and a link to try again (please_login.html). If the login is accepted, the login.cgi should send a cookie to the user, to mark him as a "logged in user", and then show a menu of protected procedures (an "out" of procedure_menu.html). In the menu, the user can select the procedures or logout (a choice must always lead to a cgi program; in this case, the logged.cgi). If the user selects logout, the cookie is deleted (actually, a "blank" cookie is sent to overwrite the old one). Then a message of being logged out is given (an "out" of you_are_logged_out.html).

The user can also choose procedures (1,2,3 in this case) from the procedure menu. By choosing it, the logged.cgi will check the existence of the cookie, and in case of existence, proceed with procedure. Otherwise, show a message that the procedure was denied (in this case, an "out" of cannot_run_procedure.html).

A user can only get the cookie if the login.cgi gives it to the user, and this is done only if the password is checked. If a hacker tries direct access to protected procedures, the logged.cgi will always check the existence of the cookie. Since there is no cookie, the hacker will go to the cannot_run_procedure.html page.

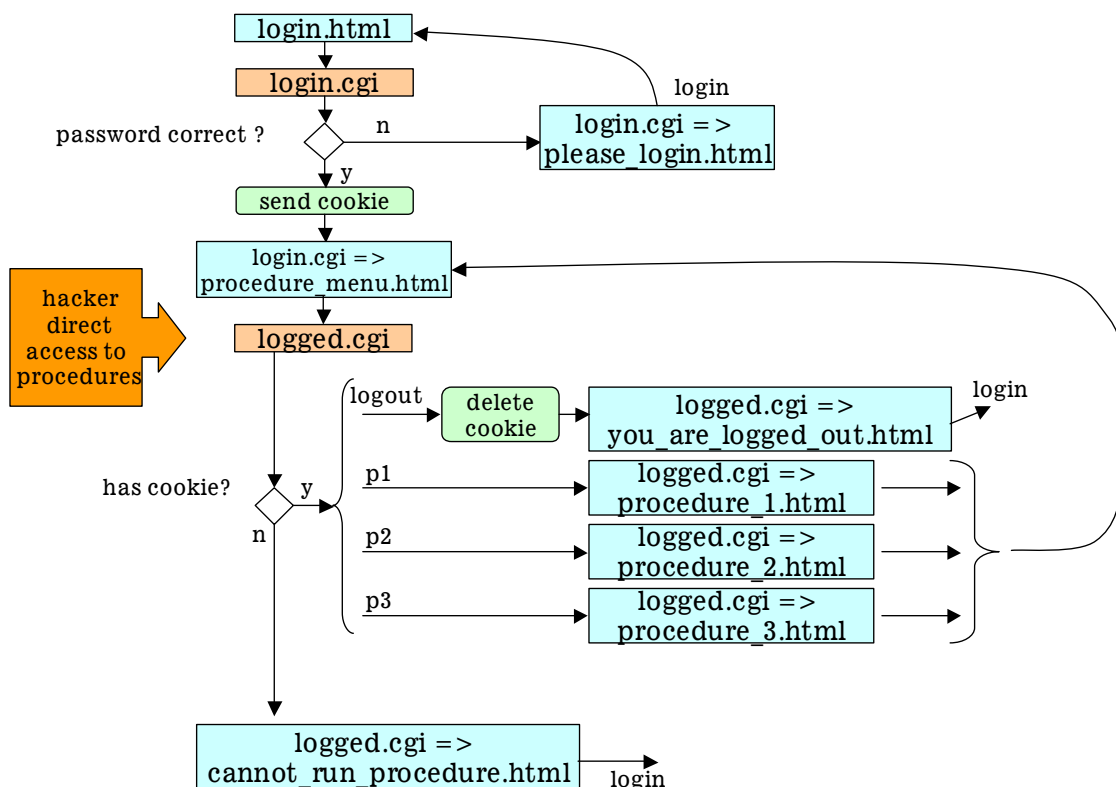


Figure 32: Map of a web system with login, using cookies

In the URL below, there's a live example.

<http://www.vbmcgi.org/index.shtml?p=19>

The source code of the CGI program login.cpp is shown below.

```
// login.cpp
```

```
#include "vbmcgi.h"

int main()
{
    VBString correctLogin = "abc";
    VBString correctPasswd = "def";
    VBString cookieName = "user_id";
    VBString cookieValue = "1234";

    VBMcgi cgi;
    cgi.formDecode();
    cgi.addFunction(VBMcgi_showFile);
    VBString login = cgi.getVarContent("s_login");
    VBString passwd = cgi.getVarContent("s_passwd");
    bool loginOK = ((correctLogin == login) &&
        (correctPasswd == passwd));

    if (loginOK) {
        cgi.httpHeader();
        cgi.setCookieNameValue(cookieName,cookieValue);
        cgi.setCookieExpires("end-of-session");
        cgi.sendCookie();
        cout << endl; // close header
        cgi.out("procedure_menu.html");
    }
    else { // if login is not OK
        cgi.out("please_login.html");
        cout << login << ";" << passwd << endl;
    }
    return 0;
}
```

The source code of the CGI program `logged.cpp` is shown below.

```
// logged.cpp

#include "vbmcgi.h"

int main ()
{
    VBString cookieName = "user_id";
    VBString correctCookieValue = "1234";

    VBMcgi cgi;
    cgi.addFunction(VBMcgi_showFile);
    VBString cookieValue = cgi.getCookieValue(cookieName);
    if (cookieValue != correctCookieValue)
    {
        cgi.addBySource("#cookieName",cookieName);
        cgi.addBySource("#cookieValue",cookieValue);
        char *cookie = getenv("HTTP_COOKIE");
        if (cookie)
            cgi.addBySource("#rawCookie",cookie);
        cgi.out("cannot_run_procedure.html");
        exit(1);
    }
    cgi.formDecode();
    VBString procedureStr = cgi.getVarContent("s_proc");
    int proc = atoi(procedureStr);

    switch (proc) {
    case 0: // logout
        cgi.httpHeader();
        cgi.setCookieNameValue(cookieName,""); // send a blank cookie to logout
        cgi.setCookieExpires("end-of-session");
```

```
    cgi.sendCookie();
    cout << endl; // end of header
    cgi.out("you_are_logged_out.html");
    break;
case 1: // procedure 1
    cgi.out("proc_1.html");
    break;
case 2: // procedure 2
    cgi.out("proc_2.html");
    break;
case 3: // procedure 3
    cgi.out("proc_3.html");
    break;
case 4: // procedure menu
    cgi.out("procedure_menu.html");
    break;
default:
    cgi.out("please_login.html");
    break;
} // end of switch;
return 0;
}
```

See an example of login to web using cookies in the URL below.

<http://www.vbmcgi.org/index.chnl?p=19>.

2.13 gif or jpeg output (non HTML data in CGI programs)

A CGI program can output non-HTML data. For instance: a gif or jpg can be the result of a CGI program. Audio, video, pdf, streams, etc. can also be output of CGI programs. To use this feature, the webmaster should properly set the first line of the HTTP header.

In the example below, the CGI will produce gif data. In this simple example, the data itself comes from a gif file. So, the CGI program sends the proper HTTP header and then opens binary the gif file, and sends its contents to the console byte per byte.

```
// gif_output.cpp

#include <fstream>
#include "vbmcgi.h"

int main()
{
    VBMcgi cgi;

    const char *file = "vbmcgi_logo.gif";

    ifstream gifFile(file);
    if (!gifFile) {
        cgi.httpCompleteHeader();
        cout << "Could not open gif file for reading" << endl;
        exit(1);
    }

    cgi.httpCompleteHeader("image/gif");
    unsigned char z;
    while (true) {
        gifFile.read((char*)&z, sizeof(char));
        if (gifFile.eof()) break;
    }
}
```

```
// copy binary to console (cout)
cout.write((char*)&z, sizeof(char));
}
return 0;
}
```

See an example redirection in the URL below.

<http://www.vbmcgi.org/index.html?p=21>

3) VBMcgi advanced tutorial

3.1 Get DateTime

It is not difficult to develop a function to read date and time from the server as a string. In this example, a sample page is exhibited having the current date and time. The code news.cpp below does that.

```
// news.cpp

#include <time.h>
#include "vbmcgi.h"

VBString getDateTime() {
    const char *months[] = {
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December"
    };
    const char *weekDay[] = {
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"
    };
    time_t ltime;
    tm *t;
    time( &ltime ); // get time and date
    t = localtime( &ltime ); // convert to struct tm

    VBString ret;

    // weekday
    ret += weekDay[t->tm_wday];
    ret += ", "; // separator

    // date
    ret += months[t->tm_mon];
    ret += " ";
    ret += t->tm_mday;
    switch (t->tm_mday) {
        case 1: ret += "st"; break;
        case 2: ret += "nd"; break;
        case 3: ret += "rd"; break;
        default: ret += "th";
    }
}
```

```

}
ret += ", ";
ret += t->tm_year + 1900;
ret += ", "; // separator

// time
ret += t->tm_hour;
ret += ":";
if (t->tm_min < 10) ret += "0";
ret += t->tm_min;
ret += ":";
if (t->tm_sec < 10) ret += "0";
ret += t->tm_sec;

return ret;
}

int main()
{
    VBMcgi cgi;
    VBString dateTime = getDateTime();
    cgi.addBySource("#dateTime",dateTime);
    cgi.out("news.html");
    return 0;
}

```

3.2 Redirection

A CGI program can redirect the web user to the URL the webmaster wishes to. To do that using VBMcgi, use the method `redirect`.

```

<!-- HTML snippet to call "redirect.cgi" -->
Redirect to:
<form method="get" action="redirect.cgi">
<input type="text" name="redirect"
value="http://www.cplusplus.com/" size="80">
<input type="submit" value="Submit">

```

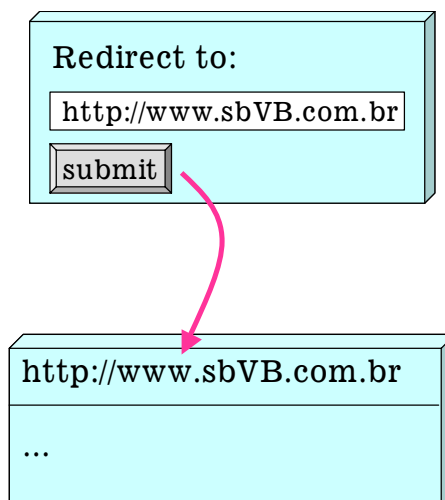


Figure 33: CGI program redirecting the client's web browser

The source code of `redirect.cgi` is shown below.

```

// redirect.cpp

```

```
#include "vbmcgi.h"
int main()
{
    VBMcgi cgi;
    cgi.formDecode();
    VBString url = cgi.getVarContent("redirect");
    cgi.redirect(url);
    return 0;
}
```

The output of the CGI program is shown below.

```
status: 301
location: http://www.sbVB.com.br/
```

See an example redirection in the URL below.

<http://www.vbmcgi.org/index.html?p=23>

4) References

[1] DEL – Departamento de Engenharia Eletrônica e de Computação da UFRJ – <http://www.del.ufrj.br> (*Department of Electronic and Computer Engineering of Federal University of Rio de Janeiro*)

[2] Apache group. <http://www.apache.org>