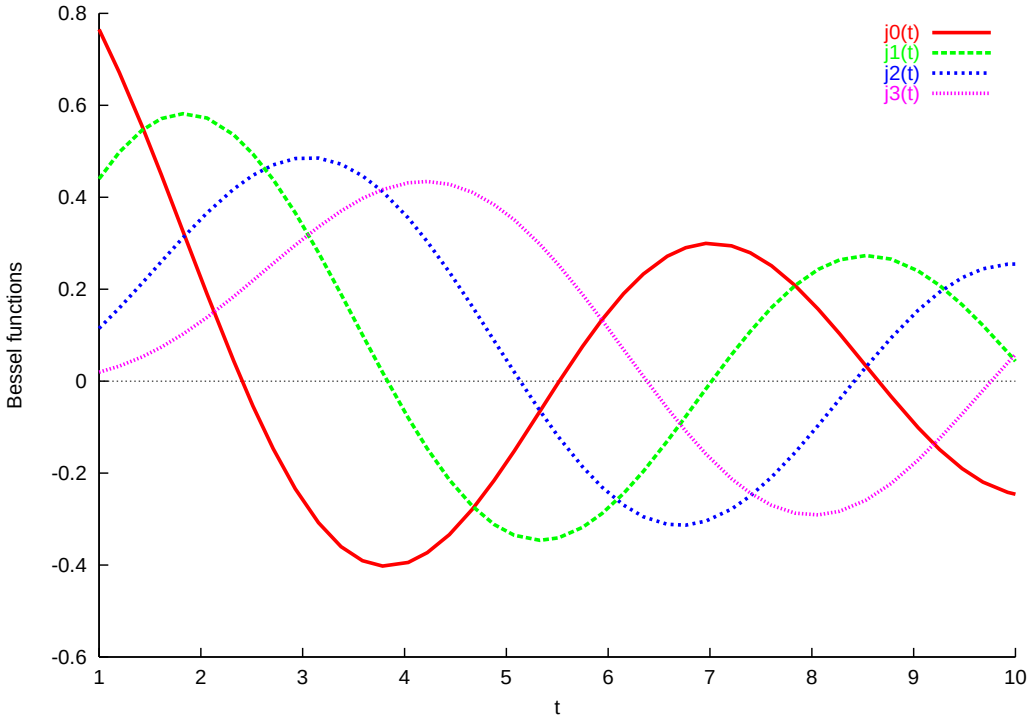
SoftIntegration®

The Ch Language Environment

CGI Toolkit

Version 3.7.0

User's Guide



How to Contact SoftIntegration

Mail	SoftIntegration, Inc. 216 F Street, #68 Davis, CA 95616
Phone	+ 1 530 297 7398
Fax	+ 1 530 297 7392
Web	http://www.softintegration.com
Email	info@softintegration.com

Revision 3.7.0, February 2008

Typographical Conventions

The following list defines and illustrates typographical conventions used as visual cues for specific elements of the text throughout this document.

- Interface components are window titles, button and icon names, menu names and selections, and other options that appear on the monitor screen or display. They are presented in boldface. A sequence of pointing and clicking with the mouse is presented by a sequence of boldface words.

  Example: Click **OK**

  Example: The sequence **Start->Programs->Ch6.0->Ch** indicates that you first select **Start**. Then select submenu **Programs** by pointing the mouse on **Programs**, followed by **Ch6.0**. Finally, select **Ch**.

- Keycaps, the labeling that appears on the keys of a keyboard, are enclosed in angle brackets. The label of a keycap is presented in typewriter-like typeface.

  Example: Press `<Enter>`

- Key combination is a series of keys to be pressed simultaneously (unless otherwise indicated) to perform a single function. The label of the keycaps is presented in typewriter-like typeface.

  Example: Press `<Ctrl><Alt><Enter>`

- Commands presented in lowercase boldface are for reference only and are not intended to be typed at that particular point in the discussion.

  Example: "Use the **install** command to install..."

  In contrast, commands presented in the typewriter-like typeface are intended to be typed as part of an instruction.

  Example: "Type `install` to install the software in the current directory."

- Command Syntax lines consist of a command and all its possible parameters. Commands are displayed in lowercase bold; variable parameters (those for which you substitute a value) are displayed in lowercase italics; constant parameters are displayed in lowercase bold. The brackets indicate items that are optional.

  Example: **ls** [**-aAbcCdfFgilLmnopqrRstux1**] [*file* ...]

- Command lines consist of a command and may include one or more of the command's possible parameters. Command lines are presented in the typewriter-like typeface.

  Example: `ls /home/username`

- Screen text is a text that appears on the screen of your display or external monitor. It can be a system message, for example, or it can be a text that you are instructed to type as part of a command (referred to as a command line). Screen text is presented in the typewriter-like typeface.

  Example: The following message appears on your screen

```
usage:  rm [-fiRr] file ...


ls [-aAbcCdfFgilLmnopqrRstux1] [file ... ]
```

- Function prototype consists of return type, function name, and arguments with data type and parameters. Keywords of the Ch language, typedefed names, and function names are presented in boldface. Parameters of the function arguments are presented in italic. The brackets indicate items that are optional.

  Example: **double derivative**(**double** (**func*)(**double**), **double** *x*, ... [**double** **err*, **double** *h*]);

- Source code of programs is presented in the typewriter-like typeface.

  Example: The program **hello.ch** with code

```
int main() {
    printf("Hello, world!\n");
}
```

  will produce the output `Hello, world!` on the screen.

- Variables are symbols for which you substitute a value. They are presented in italics.

  Example: module *n* (where *n* represents the memory module number)

- System Variables and System Filenames are presented in boldface.

  Example: startup file **/home/username/.chrc** or **.chrc** in directory `/home/username` in Unix and **C:\>_chrc** or **_chrc** in directory C:\> in Windows.

- Identifiers declared in a program are presented in typewriter-like typeface when they are used inside a text.

  Example: variable `var` is declared in the program.

- Directories are presented in typewriter-like typeface when they are used inside a text.

  Example: Ch is installed in the directory `/usr/local/ch` in Unix and `C:/Ch` in Windows.

- Environment Variables are the system level variables. They are presented in boldface.

  Example: Environment variable **PATH** contains the directory /usr/ch.

Table of Contents

Chapter 1

System Administration in Windows

This chapter describes the system requirement, installation, and system administration for SoftIntegration CGI Toolkit in in Windows. To install and configure Ch CGI to run in Windows, you have to login with an administrative privilege.

1.1 System Requirement for Windows NT/2000/XP

The Ch language environment shall be installed before Ch CGI Toolkit can be installed. One of Ch Standard, Professional, and Student Editions can be used for CGI programming. To install and use CGI toolkit in Windows, the following minimum requirements should be met:

- Operating System: Windows NT/2000/XP. For Windows NT, it shall be NT workgroup or Server 4.0 with SP3 or above, Windows 2000/XP/2003/Vista.
- CPU: with a 486 or higher microprocessor.
- Memory: a minimum of 16 Megabytes of RAM.
- Disk Space: 2 Mb.

The CGI Toolkit runs in the following Web servers:

- Microsoft Internet Information Server (IIS) 4.0 or above for Windows NT server and Windows 2000/XP
- Apache Web Server 1.2.6 or above
- Netscape Enterprise Server 3.0 or above

Note: the CGI Toolkit is likely to work in other Web servers. However, other Web servers not listed here are yet to be tested and verified.

1.2 Installation in Windows

1.2.1 Install CGI Toolkit in Windows

Before starting the installation, close all running applications. If you have installed an older version of the CGI Toolkit before, uninstall it off the system first.

To start the installation process from a CD:

1. Login to the computer with an Administrator privilege under NT/2000/XP.
2. Insert the Ch setup CD into the CD-ROM drive.
3. On Windows NT/2000/XP, the setup process starts automatically if AutoPlay for CDs is enabled. Click `Next` to continue.

   If AutoPlay for CDs is not enabled, use Windows Explorer to navigate from the root directory of the CD. Then, double-click the `Setup.exe` file.
4. Read and accept the SoftIntegration license agreement.
5. Enter the product Key
6. Accept default folder names.
7. Accept the typical installation and press `Next`
8. Follow the instructions of the setup program to install the CGI Toolkit on your computer.
9. Click `Finish` to complete the installation

If you reinstall the IIS web server or install Ch before the installation of IIS web server, you need to run the command `CHHOME\toolkit\bin\setIISMetabase.exe`. The error message created by setIISMetabase.exe can be found at setIISMetabase.log. If no error message is found, it means the program runs correctly. **Note that CHHOME is not the string "CHHOME". Rather, it is the Windows filesystem path under which Ch is installed.** For instance use **C:\Ch** for CHHOME in Windows.

Note: You are able to quit the installation at any time by pressing the `<Cancel>` button displayed in every dialog box during the installation. You can also move back and forth to review your settings by clicking the `<Back>` and `<Next>` buttons.

1.2.2 Uninstall CGI Toolkit in Windows

Stop all the Ch programs.

Click **Control Panel** in **My Computer**. Click **Add/Remove Programs**, select **SoftIntegration CGI Toolkit x.xx**, where `x.xx` is the version number such as 1.00. then Click **Add/Remove ...**. Press **Yes** if you are asked to completely remove the CGI Toolkit and all of its components.

1.3 Web Server Configuration and Setup CGI in Windows

This section addresses issues related to setup and system administration of CGI Toolkit in Windows NT/2000/XP. To run Ch CGI scripts successfully under windows, all CGI scripts need to have **.ch** as a file extension.

1.3.1 IIS in Windows NT/2000/XP

Install IIS

To install the Internet Information Services (IIS), follow the procedure as follows.

- click Control Panel
- click Add/Remove Programs
- click Add/Remove Windows Components
- check the Internet Information Services Box

Setup Web Server

If you already have the Internet Information Server (IIS) setup prior to installing CGI Toolkit, then the installation of CGI Toolkit will automatically setup registry and metabase settings for you.

If you reinstall IIS or install the IIS after the installation of CGI Toolkit, you need to run the command `CHHOME\toolkit\bin\setIISMetabase.exe` to automatically configure web server for you.

- copy C:/Ch/toolkit/demos/cgi/chhtml to C:/inetpub/wwwroot/chhtml
- create directory C:/inetpub/cgi-bin
- copy C:/Ch/toolkit/demos/cgi/chcgi to C:/inetpub/cgi-bin/chcgi

Add index.html as a Default Web Page in Windows NT/2000/XP

- click Control Panel again
- click Performance and Maintenance
- click Administrative Tools
- click Internet Information Services
- click the local computer directory
- In the local computer directory, click Web Sites
- right click Default Web Site, click Properties
- click the Documents tab
- add default document index.html

In some Windows XP system, you may need to follow the following steps to add index.html as a default Web page.

- click Control Panel again
- click Performance and Maintenance
- click Administrative Tools

- click Computer Management
- click to expand Services and Applications
- click Internet Information Services
- click Web Sites
- right click Default Web Site, click Properties
- click the Documents tab
- add default document index.html

Create IIS Virtual Directory for CGI

You need to check with your Web server administrator to get information about the server document root directory and CGI directory, as well as its availability.

Here are the steps for you to follow.

1. First launch the Internet Information Services snap-in.

   (a) click Control Panel again
   (b) click Performance and Maintenance
   (c) click Administrative Tools
   (d) click Internet Information Services

   In some Windows XP system, you may need to follow the following steps to launch the Internet Information Services snap-in.

   (a) click Control Panel again
   (b) click Performance and Maintenance
   (c) click Administrative Tools
   (d) click Computer Management
   (e) click to expand Services and Applications
   (f) click Internet Information Services

2. Create a web server virtual directory, `cgi-bin`. If you have already had one, skip to (3).

   - First, create a `cgi-bin` directory in the Web server root directory, which by default is `C:\inetpub` and you need to create the directory `C:\inetpub\cgi-bin` if it does not exist.
   - Then select the Web site to which you want to add a directory. Left click the `Action` button next to the `File` button, point to `New`, and select `Virtual Directory`.
   - In the New Virtual Directory Wizard for alias, enter `cgi-bin`.
   - Enter the Web Site Content Directory with the directory you just created, which by default is `C:\inetpub\cgi-bin`.
   - In Access Permission, Make sure options `Read`, `Run scripts`, and `Execute` are checked, while the other options are left unchecked.

Note: If you are using NTFS, you can also create a virtual directory by right-clicking a directory in Windows Explorer, click Sharing, and then selecting the Web Sharing property sheet.

3. Copy all files from `CHHOME\toolkit\demos\CGI\chcgi` including the directory chcgi itself to the directory where virtual directory `cgi-bin` is associated, which by default is `C:\inetpub\cgi-bin`. You will get `C:\inetpub\cgi-bin\chcgi`

4. Copy all files from `CHHOME\toolkit\demos\CGI\chhtml` including the directory chcgi itself to the Web Server document home directory, which is `C:\inetpub\wwwroot` by default. You will get `C:\inetpub\wwwroot\chhtml`

Start and Stop IIS Web Server

- click Control Panel again
- click Performance and Maintenance
- click Administrative Tools
- click Internet Information Services
- click the local computer directory
- in the local computer directory, click Web Sites
- right click Default Web Site, click Start to start the web server or click Stop to stop the web server.

In some Windows XP system, you may need to follow the following steps to start or stop the IIS Web server.

- click Control Panel again
- click Performance and Maintenance
- click Administrative Tools
- click Computer Management
- click to expand Services and Applications
- click Internet Information Services
- click Web Sites
- right click Default Web Site, click Start to start the web server or click Stop to stop the web server.

Testing Ch CGI in the Web Server

Start the web browser, type one of the following URLs,

```
http://localhost/iishelp/
http://localhost/chhtml/
http://localhost/chhtml/index.html
http://YourComputerName.YourDomain/chhtml/index.html or
http://YourComputerName/chhtml/index.html
```

To test a simple Ch CGI script, you can test the following **hello.ch** code in the cgi-bin directory `C:\inetpub\cgi-bin\chcgi\hello.ch`:

```
#!/bin/ch
printf("HTTP/1.0 200 OK\n");
printf("Content-Type: text/html\n\n");
printf("<HTML>\n");
printf("<HEAD>\n");
printf("<Title> Hello World </Title>\n");
printf("</Head>");
printf("<BODY>\n");
printf("<h4> Hello, world </h4>\n");
printf("</BODY>\n");
printf("</HTML>\n");
```

The above code can be written in CGI classes as follows:

```
#!/bin/ch
#include <cgi.h>
class CResponse Response;
Response.begin();
Response.title("Hello World");
printf("<h4> Hello, world </h4>\n");
Response.end();
```

In a plain text, the code can be written as:

```
#!/bin/ch
printf("Content-type: text/plain\n\n");
printf("Hello, world\n");
```

Start the web browser, type one of the following URLs,

```
http://localhost/cgi-bin/chcgi/hello.ch
http://YourComputerName.YourDomain/cgi-bin/chcgi/hello.ch
http://YourComputerName/cgi-bin/chcgi/hello.ch
```

Note: The first line `#!/bin/ch` can be taken away from **hello.ch** and it still works. We keep it there for consistency with scripts running under Unix.

Delete Ch demos and samples

If you are going to run Ch programs on a production line, you may want to keep only your production code. You can either delete all Ch provided code or delete the `cgi-bin` virtual directory

In the Internet Information Services snap-in, select the virtual directory `cgi-bin` that you want to delete. Click the `Action` button next to `File button`, and select `Delete`.
Deleting a virtual directory does not delete the corresponding physical directory or files.

1.3.2 Apache Web Server in Windows NT/2000/XP

If you want to put all of your CGI scripts into one directory, add the following line to your httpd.conf or srm.conf file. (**Note**: C:/Program Files/Apache Group/Apache/cgi-bin/ is the directory where CGI scripts are intended to be put. It can be any directory.)

```
ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache/cgi-bin/"
```

Make sure do not use

```
ScriptAlias /cgi-bin/ "C:\Program Files\Apache Group\Apache\cgi-bin\"
```

After you have made this change, stop and restart the Apache service.

The following is a sample portable Ch CGI code that will work in Apache in both Windows and Unix. In windows, you need to copy `CHHOME\bin\ch.exe` to `X:\bin\ch.exe` first, where X is the drive where the Apache server is installed such as **C:**.

```
#!/bin/ch
printf("HTTP/1.0 200 OK\n");
printf("Content-Type: text/html\n\n");
printf("<HTML>\n");
printf("<HEAD>\n");
printf("<Title> Hello World </title>\n");
printf("</Head>");
printf("<BODY>\n");
printf("<h4> Hello, world </h4>\n");
printf("</BODY>\n");
printf("</HTML>\n");
```

Apache provides an emulation of the UNIX shell (`#!/path/to/ch`) syntax. Thus, a path to a valid interpreter can be put at the top to make it work. You may change `#!/bin/ch` to `#!C:\bin\ch.exe` in the above sample code if Ch is installed in C drive.

If you want to enable CGI scripts based on the file extension **.ch** and CGI outside ScriptAliased directories, add the following line to either httpd.conf or srm.conf:

```
AddHandler cgi-script .ch
```

By default, CGI scripts are not allowed in your DocumentRoot directory, but they are allowed in other document directories. Document directories are created with the Alias command in either httpd.conf or srm.conf:

```
Alias /document/ "D:/documentsamp/"
```

You can then include files that end in .ch within a document directory. You will still need to include the #! line with the full path to the ch.exe interpreter, as shown earlier.

If you want to allow CGI scripts in the DocumentRoot directory, add the ExecCGI option to the Options directive between the <Directory> and </Directory> entry for your DocumentRoot in httpd.conf or access.conf.

After you have updated it, your Options directive may look something like:

```
Options Indexes FollowSymLinks ExecCGI
```

1.3.3 Netscape Enterprise Web Server in Windows NT/2000/XP

The following information is for Netscape FastTrack Server 2.0. Other 2.0 and 3.0 Netscape Servers (Communications, Enterprise) should be similar.

To set up Ch for Win32 to run on the FastTrack Server, you need to install the FastTrack Server based on the documents from Netscape.

After the installation of Ch, you need to restart the Web server.

Set up a Shell CGI directory to run Ch CGI scripts. A regular CGI directory will not work – that is only for executable files. You set this up with the FastTrack Administrator; see the documentation for details.

If you would like to access Ch CGI scripts in other directories, you need to associate an extension, such as .ch, with the MIME type. Before you follow these steps, you must add at least one Shell CGI directory - this will enable shellcgi on your server (you can delete this directory, and shellcgi will remain enabled). Follow these steps to associate .ch with the MIME type:

- In the Server Administrator, click Server Preferences, then select MIME Types from the frame on the left.
- Add a new MIME Type with magnus-internal/shellcgi as the Content Type, and ch as the File Suffix. If a type for magnus-internal/shellcgi already exists, simply add ch to the list of File Suffixes. Don't include the leading dot on the file suffix.
- Save and apply these changes. You should be able to put a Ch CGI script in any directory, provided the script ends with the .ch suffix.

If you are having trouble running CGI scripts on your Netscape server, check the following:

- Ensure that the script is readable by the account used by the Netscape service. Generally, this means you should make the script readable by the Everyone group.
- Ensure that all supporting files, like the Ch binary files, the Ch library files, and the modules that you use, are all readable by the account used by the Netscape service (i.e., the Everyone group).
- Check the Error log of the FastTrack server if your CGI script does not work.
- Because Netscape servers run as services, you need to make sure that files and environment variables are available to them.
- Since Netscape uses associations to run scripts, and POST'ed data is sent through the `stdin` data stream to a program, this may be related to the problems of redirection in handling Posted data with Ch programs.

1.3.4 Configure Other Web Servers to Support Ch CGI

If other Web servers are used, check the server's documentation on how to set up a CGI interpreter.

In general, set up a directory where executable scripts go, and put your Ch script there. Make sure that the user account that the Web server uses can read the script, the files may need to be readable for the Everyone

group.

Because most Web servers run as services, you need to take special steps to make sure that files and environment variables are available to them.

Chapter 2

System Administration in Unix

This chapter describes the system requrements, installation, and administration for the SoftIntegration CGI Toolkit in Unix.

2.1 System Requirements for Unix

The Ch language environment shall be installed before Ch CGI Toolkit can be installed. One of Ch Standard, Professional, and Student Editions can be used for CGI programming. The CGI Toolkit is supported in the following Unix operating systems.

- Solaris 2.6 or above
- HP-UX 10.20 or above
- Linux on Intel architectuer (kernel 2.4.20-8 or above)
- LinuxPPC on Power architecture (kernel 2.6.10-1 or above)
- Mac OS X 10.3 or above
- FreeBSD 5.1 or above

The hardware requirement for the Intel Linux platform is

- Pentium/90Mhz or above
- A minimum of 16 Megabytes of RAM
- Disk space of 2 Mb.

The CGI Toolkit has been tested with the following Web servers: Netscape Enterprise Server (3.0 or above) and Apache Web Server (1.2.6 or above).

2.2 Install Apache in Unix

The Apache HTTP Server is an open-source secure, efficient and extensible HTTP server that provides HTTP services in sync with the current HTTP standards. More information concerning the Apache HTTP Server can be found on the Web at http://httpd.apache.org.

The installation procedures and the location of the configuration files and webserver root directory are Linux distribution dependent. Listed below are installation guidelines for a few common Linux distributions. Please refer to your specific distribution's website and the Apache website for more information on installing and trouble shooting Apache.

2.2.1 Installing Apache under Ubuntu

Login your system and run the Synaptic Package Manager which is a graphical program for installing packages. You can launch Synaptic from **System -¿ Administration -¿ Synaptic Package Manager**. The **System** menu should be the third menu on the menubar. Synaptic will ask for an adminstrative password. Scroll the right window pane which shows a listing of available packabes until you find 'apache2'. Right click on the package and select 'mark for installation' and then click 'Apply' at the top of the program. It will show you a listing of other required packages that need to be installed. Click 'Okay.' Once all of the packages have been installed, close Synaptic and open a terminal. Start Apache by running

```
/etc/init.d/apache2 start
```

Open a Web browser and type

```
http://127.0.0.1
```

into the URL. A page either showing the Apache logo and other information or a page saying "It works" should be displayed.

By default, the configuration files are located under **/etc/apache/**. The webserver root directory is under **/var/www/** and the cgi-bin directory is under **/usr/lib/cgi-bin**.

2.2.2 Installing Apache under Fedora

Login your system and run the Fedora package manager. Under the Fedora menu, select the "Add/Remove Software." Then type in your root password when prompted. The package manager window should pop up. Then click on the 'Search' button and search for apache. Select the Apache 2 package by checking the box located to the left of the package name. Then click the 'Apply' button at the bottom right of the program. The package manager will ask for a confimation to install the packages and any dependencies. Select 'Continue'. Once the installation of the packages are complete, exit out of the package manager. Open a terminal and start the Apache server with

```
/etc/init.d/httpd start
```

Open a Web browser and type in

```
http://127.0.0.1
```

into the URL. A page either showing the Apache logo and other information or a page saying "It works" should be displayed.

For Fedora 5.0, by default, the configuration files are located under **/etc/httpd/**. The webserver document root directory is under **/var/www/html** and the cgi-bin directory is under **/var/www/cgi-bin**.

2.2.3 Installing Apache under Gentoo

The Gentoo Wiki has a nice installation page for Apache on the Web at http://gentoo-wiki.com/Apache2_Install.

Open up a terminal. Either log in as root, su into root or use sudo and run **emerge -av apache**. Once portage has finished compiling and installing Apache, start it with

```
/etc/init.d/apache2 start
```

Open a Web browser and type in

```
http://127.0.0.1
```

into the URL. A page either showing the Apache logo and other information or a page saying "It works" should be displayed.

The configuration files are located under **/etc/apache2/**. The webserver document root directory is under **/var/www/localhost/html** and the cgi-bin directory is under **/var/www/localhost/cgi-bin**.

2.3 Install and Uninstall CGI Toolkit in Unix

2.3.1 Install CGI Toolkit in Unix

If you have older version installed, uninstall that version from the system first..
If you have the CD with you, install the CGI Toolkit in the following steps.

1. Log in as root
2. Insert the Ch setup CD into the CD-ROM drive. Depending on how your operating system is configured, your CD drive may be mounted automatically. If the CD drive is not mounted, you must mount it before continuing.
3. Go to your CD-ROM directory where the CD-ROM is mounted.
4. Run the following command.

```
ch ./install.ch
```

2.3.2 Uninstall CGI Toolkit in Unix

Remove all components of CGI Toolkit from the CHHOME directory where you installed Ch. **Note that CHHOME is not the string "CHHOME". Rather, it is the Unix filesystem path under which Ch is installed**. Under Unix, the default directory for installing Ch version 2.0 is **/usr/local/ch2.0**, and the symbolic /usr/local/ch or /usr/ch will be created, and CHHOME will be set to either /usr/local/ch or /usr/ch. More specifically, remove files `libcgi.dl` and `libwcgi.dl` in the directory `CHHOME/toolkit/dl`, `cgi.h` and `wcgi.h` in `CHHOME/toolkit/include`, and directories `CHHOME/toolkit/lib/cgi` and `CHHOME/toolkit/demos/CGI`.

2.3.3 Install CGI Toolkit in Max OS X

1. Download the compressed file from the SoftIntegration website.
2. Your Mac OS X shuttle will uncompress the file and create a directory chcgi-3.5.1.macosx in your Desktop. If not, you can untar and decompress the downloaded file with the command below.

```
gzip -cd  chcgi-3.5.1.macosx.tgz | tar xvf -
```

3. Goto *chcgi-3.5.1.macosx Folder* from Mac OS X Desktop, then double click chcgi-3.5.1.pkg and you can follow the instructions to install.

2.3.4 Uninstall CGI Toolkit in Mac OS X

You will have to be the root user for uninstalling Ch.

- ```
  remove /usr/local/ch/docs/chcgi.pdf
  remove /usr/local/ch/toolkit/include/cgi.h
  remove /usr/local/ch/toolkit/include/wcgi.h
  remove /usr/local/ch/toolkit/dl/libcgi.dl
  remove /usr/local/ch/toolkit/dl/libwcgi.dl
  remove /usr/local/ch/toolkit/lib/cgi
  remove /usr/local/ch/toolkit/demos/CGI
  remove /usr/local/ch/toolkit/release/release_CGI
  remove /usr/local/ch/toolkit/license/license_CGI
  ```
- remove the package receipt file ”/Library/Receipts/chcgi-3.5.1.pkg”

2.4 Configuration and Setup of Web Browsers in Unix

Ch is denoted by a specific file extension, with **.ch** as the default Ch file extension, **.chs** as the safe Ch file extension. Both the Web browser and server can be configured to take advantage of internet computing.

1. Copy the file **CHHOME/config/.mime.types** to your home directory or append the following to your existing file ˜**/.mime.types** in the user’s home directory

```
# handle CH language environment
application/x-chs              chs
```

2. Then, copy the file **CHHOME/config/.mailcap** to your home directory or append the following to your existing file ˜**/.mailcap** in the user’s home directory.

```
#handle CH language environment
application/x-chs; ch -S %s
```

When file ˜**/.mailcap** in user’s home directory is changed, the Web browser needs to be restarted to make it effective.
```

2.5 Configuration and Setup of Web Servers

This section addresses the issues related to the setup and system administration of the CGI Toolkit in Unix, Mac OS X, and Gumstix.

2.5.1 Apache 1.0 Web Servers

For the Apache Web server, add the following line to file **server_home_dir/conf/mime.types**

```
application/x-chs          chs
```

If you want to enable CGI scripts based on an extension, such as .ch, you need to add the following line to either httpd.conf or srm.conf or apache2.conf:

```
AddHandler cgi-script .ch
```

To use dynamically linked libraries in */usr/ch/extern/lib* and */usr/local/lib*, add the following line to either httpd.conf or srm.conf or apache2.conf:

```
SetEnv LD_LIBRARY_PATH /usr/ch/extern/lib:/usr/local/lib
```

After the above changes, the Web server needs to restart for the changes to be effective.

To test Ch CGI in a Apache web server in Mac OS X, copy CHHOME/toolkit/demos/CGI/chhtml directory to /Library/WebServer/Documents/chhtml, and CHHOME/toolkit/demos/CGI/chcgi directory to /Library/WebServer/CGI-Executables/chcgi. Then, click hyperlinks in your web site `http://your.MacMachine.website.com/chhtml/`.

2.5.2 Apache 2.0 Web Servers

No configuration modifications are required.

2.5.3 BOA Web Servers on the Gumstix

By default, a BOA Web server is installed and run on Gumxstix. No configuration modifications are required. By default, the configuration file is located under **/etc/boa/**. The webserver root directory is **/var/www/** and the cgi-bin directory is **/usr/lib/cgi-bin**.

2.5.4 Netscape Web Server

If you administer a Web server and want to generate a Ch applet from a Web server, add the following line to the Netscape WWW server configuration file **mime.types** located in server_home_dir/https-80_or_http/config/mime.types

```
type=application/x-chs          exts=chs
```

2.6 Testing Ch CGI Scripts and demos

To run the sample code of Ch from the Web sever, create a `chhtml` directory in the Web server document root directory where your Web documents are located, and create a `chcgi` directory in your Web server cgi-bin directory.

2.6.1 Hardcopying the Ch CGI Scripts and Demos

Then, copy all files including the subdirectories from CHHOME/toolkit/demos/CGI/chhtml to web-server-document-root/ and all files from CHHOME/toolkit/demos/CGI/chcgi to webserver-cgi-bin-directory/. Then you will have the directory web-server-document-root/chhtml and webserver-cgi-bin-directory/chcgi.

For example, under Redhat, copy /usr/local/ch/toolkit/demos/CGI/chhtml to /var/www/html and /usr/local/ch/toolkit/demos/CGI/chcgi to /var/www/cgi-bin using

```
cp -r /usr/local/ch/toolkit/demos/CGI/chhtml /var/www/html
cp -r /usr/local/ch/toolkit/demos/CGI/chcgi  /var/www/cgi-bin
```

2.6.2 Symbolic Linking the Ch CGI Scripts and Demos

Instead of copying files, a symbolic link can be created as follows

```
ln -s /usr/local/ch/toolkit/demos/CGI/chhtml /var/www/html/chhtml
ln -s /usr/local/ch/toolkit/demos/CGI/chcgi  /var/www/cgi-bin/chcgi
```

Some Web servers, such as Apache 2.0, require a modification to the configuration file in order to allow the system to follow symbolic links. Please refer to your Web server documentation.

2.6.3 Setting Up the Correct Permissions

In order to make sure the permissions are setup correctly, run the following

```
chmod -R 755 /var/www/html/chhtml
chmod -R 755 /var/www/cgi-bin/chcgi
```

2.6.4 Trying the Demos

Try the following URLs in your Web browser:

```
http://127.0.0.1/chhtml/ or
http://localhost/chhtml or
http://YourComputerName.YourDomain/chhtml or
http://YourComputerName/chhtml
```

Chapter 3

Common Gateway Interface

Common Gateway Interface (CGI) is a standard that specifies how external programs interface with a Web server. One of the most important applications of CGI is handling fill-out forms. A CGI program located on a host machine's Web server can accept a user's input through a fill-out form and generate Web pages dynamically.

3.1 Common Gateway Interface in Ch

A CGI program can be written in any language that allows it to be executed on a host computer where a Web server is located. The most commonly used languages for CGI at present are C and Perl. CGI programs written in C or C++ normally have to be compiled. These compiled programs are difficult to modify and maintain. Therefore, many people prefer to write CGI programs in Perl which is interpretive and resembles C language. However, it is difficult to develop and maintain large programs in Perl. Ch can be used for common gateway interface directly without compilation, so as to speed up the development process greatly. More importantly, it can leverage a large body of existing C programs.

There are many security features built into the Ch language environment for internet computing. A CGI program is normally run in a regular Ch shell. A webmaster can examine all CGI programs by looking at the source code directly. The execution environment for Ch shell can be controlled by modifying the startup file **.chrc** in Unix and **_chrc** in Windows. A Ch script starting with

```
#!/bin/ch
```

at the first line of the program can be executed as a regular Ch program without any modification by a Web server in both Unix and Windows.

The path for commands executable by the Ch shell is controlled by system-wide startup file **CHHOME/config/chrc** which includes **.chrc** in Unix and **_chrc** in Windows at the home directory of the account that executes CGI programs. The startup file **.chrc** can be modified to restrict the programs executable by the CGI program.

3.2 Classes for Common Gateway Interface

The enterprise edition of the Ch language environment is capable of common gateway interface. It contains several classes and demonstrations CGI programs. The CGI programs can be found in the directory CHHOME/toolkit/demos/CGI. On-line documentation and demos of CGI in Ch are available on the Web. Header file **cgi.h** contains the definition of classes such as **CResponse**, **CRequest**, **CServer**, and **CCookie**

classes, and their member functions as well as defined constants. These classes provide convenient mechanisms for common gateway interface. Member functions of these classes are listed in Tables 3.1 to 3.4. The **CResponse** class is typically used with the following syntax:

```
#include <cgi.h>
/* ... */
class CResponse Response;
/* ... */
Response.setContentType(content);
Response.begin();
/* ... */
Response.end();
```

Two generic data types **chchar** and **chstrarray** are typedefed in the header file **cgi.h** as follows.

```
typedef char   chchar;
typedef char** chstrarray;
```

When the Unicode is used, these two generic data types, **chchar** and **chstrarray**, are typedefed in the header file **wcgi.h** as follows.

```
typedef wchar_t   chchar;
typedef wchar_t** chstrarray;
```

The content type delivered to the Web browser is handled by function **CResponse**::**setContentType**(). Any **CResponse::set*()** member function should be called before **CResponse**::**begin**() is called. Before the program ends, the function **CResponse**::**end**() should be called. Detailed description of each function can be found in the chapter about common gateway interface in *The Ch Language Environment — Reference Guide*.

3.3 Processing Fill-Out Forms

A simple CGI Ch program for handling forms will be presented by an example of ordering a pizza through cyberspace as shown on the Web page in Figure 3.1, whose HTML source code is shown in Program 3.1. Here, the street address and telephone number of a customer are entered through the default text input of a fill-out form. The choice of topings for the pizza is entered through the check boxes. More than one toping can be selected. There are two different request methods to handle a fill-out form: one is called POST, the other is GET. In the default request method of GET, the encoded fill-out form contents are appended to the URL as if they were a normal query through the environmental variable `QUERY_STRING`. On the other hand, the fill-out form contents in the method of POST are sent to the server through stdin rather than as a part of a URL. When the request method of POST is used, the environment variable `CONTENT_LENGTH` can be used to determine how much data shall be read from stdin. In this example, the request method of POST is used as shown near the beginning of Program 3.1. The CGI program form.ch located in the default Ch/CGI program directory cgi-bin/ch of a Web server to process this fill-out form is indicated by the field ACTION in Program 3.1. The role of CGI in this example is to obtain the customer's street address, phone number, and selection of topings. If an automatic pizza making machine is used, pizzas can be made and delivered automatically.

The simple CGI Ch program form.ch in Program 3.2 can obtain the input submitted through a fill-out form. As explained before, to run this CGI program as a Ch script in a Web server, the following line of code

Table 3.1: Member functions of class **CResponse**.

Function	Description
addCookie()	adds a specified cookie with attributes.
addHeader()	adds an HTTP header to the HTTP response.
begin()	begins to send output. **Mandatory in CGI**.
end()	ends standard output. **For CGI only**.
exit()	causes the server to stop processing a script and return.
flush()	sends buffered HTML output immediately.
getBuffer()	retrieves the value of the **Buffer** property.
getCacheControl()	retrieves the value of the **CacheControl** property.
getCharSet()	retrieves the value of the **CharSet** property.
getContentType()	retrieves the value of the **ContentType** property.
getExpires()	retrieves the value of the **Expires** property.
getExpiresAbsolute()	retrieves the value of the **ExpiresAbsolute** property.
getStatus()	retrieves the value of the **Status** property.
PICS()	adds a value to the PICS label field of the header.
redirect()	causes the browser to attempt to connect to a different URL.
setBuffer()	sets the value of the **Buffer** property.
setCacheControl()	sets the value of the **CacheControl** property.
setCharSet()	sets the value of the **CharSet** property.
setContentType()	sets the value of the **ContentType** property.
setExpires()	sets the value of the **Expires** property.
setExpiresAbsolute()	sets the value of the **ExpiresAbsolute** property.
setStatus()	sets the value of the **Status** property.
title()	sets the title of an HTML page. **For CGI only**.

Table 3.2: Member functions of class **CRequest**.

Function	Description
binaryRead()	retrieves the bytes that were read by an HTTP Post and place it into a buffer.
getCookie()	retrieves a cookie.
getCookies()	retrieves all cookies.
getForm()	retrieves a value of the specified name which was read by POST or GET method.
getForms()	retrieves all values of a specified name which were read by POST or GET method.
getFormNameValue()	retrieves all pairs of name and value that were read by POST or GET method.
getServerVariable()	retrieves the value of a specified ServerVariable.
getTotalBytes()	retrieves the size of the current request in bytes.

Table 3.3: Member functions of class **CServer**.

Function	Description
HTMLEncode()	applies HTML encoding to the specified string.
URLEncode()	applies URL encoding rules, including escape characters, to the specified string.
mapPath()	maps the specified relative or virtual path to the corresponding physical directory on the server.

Table 3.4: Member functions of class **CCookie**.

Function	**Description**
addPort()	adds a new port into the portlist of the cookie. For version 1 only.
getComment()	retrieves the **Comment** attribute of the cookie. For version 1 only.
getCommentURL()	retrieves the **CommentURL** attribute of the cookie. For version 1 only.
getDiscard()	retrieves the **Discard** attribute of the cookie . For version 1 only.
getDomain()	retrieves the **Domain** attribute of the cookie.
getMaxAge()	retrieves maximum age of the cookie.
getName()	retrieves the name of the cookie.
getPath()	retrieves the path on the server to which browser returns the cookie.
getPorts()	retrieves all ports in the portlist of the cookie. For version 1 only.
getSecure()	determines if the browser is sending the cookie only over a secure protocol.
getValue()	retrieves the value of the cookie.
getVersion()	retrieves the version of the protocol the cookie complies with.
setComment()	sets the **Comment** attribute of the cookie. For version 1 only.
setCommentURL()	sets the **CommentURL** attribute of the cookie. For version 1 only.
setDiscard()	sets the **Discard** attribute of the cookie. For version 1 only.
setDomain()	sets the **Domain** attribute of the cookie.
setMaxAge()	sets maximum age of the cookie.
setName()	sets the name of the cookie.
setPath()	sets the path on the server to which browser returns the cookie.
setSecure()	sets the **Secure** attribute of the cookie.
setValue()	sets the value of the cookie.
setVersion()	sets the version of the protocol the cookie complies with.

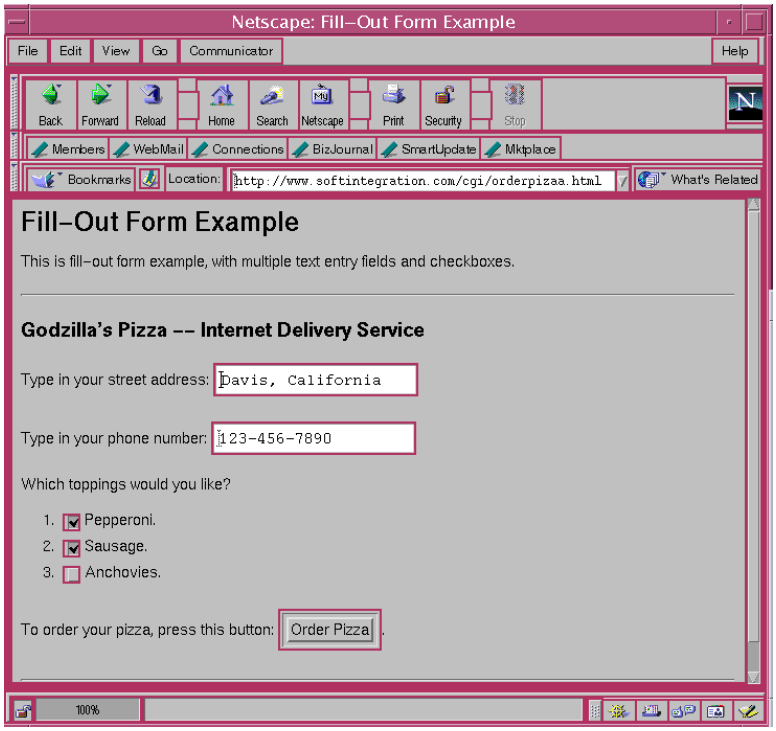


Figure 3.1: Part of the fill-out form for ordering a pizza through the cyberspace.

```
<HTML>
<TITLE>Fill-Out Form Example </TITLE>
<BODY>
<H1>Fill-Out Form Example </H1>

This is a fill-out form example, with multiple text entry
fields and checkboxes. <P>
<HR>

<FORM METHOD="POST" ACTION="/cgi-bin/ch/form.ch">
<H2>Godzilla's Pizza -- Internet Delivery Service</H2>
Type in your street address: <INPUT NAME="address"> <P>
Type in your phone number: <INPUT NAME="phone"> <P>
Which toppings would you like? <P>
<OL>
<LI> <INPUT TYPE="checkbox" NAME="topping" VALUE="pepperoni"> Pepperoni.
<LI> <INPUT TYPE="checkbox" NAME="topping" VALUE="sausage"> Sausage.
<LI> <INPUT TYPE="checkbox" NAME="topping" VALUE="anchovies"> Anchovies.
</OL>
To order your pizza, press this button: <INPUT TYPE="submit"
VALUE="Order Pizza">. <P>
</FORM>

<HR>
</BODY>
</HTML>
```

Program 3.1: Fill-out form in html file for ordering pizza.

```
#!/bin/ch
#include <cgi.h>

int main() {
  chstrarray values;
  class CResponse Response;
  class CRequest Request;

  Response.setContentType("text/html");
  Response.begin();
  Response.title("CGI FORM results");
  printf("<H1>CGI FORM test script reports:</H1>\n");

  printf("The following 4 name/value pairs are submitted<p>\n");
  printf("<ul>\n");
  printf("<li> <code>address = %s </code>\n", Request.getForm("address"));
  printf("<li> <code>phone = %s </code>\n", Request.getForm("phone"));
  Request.getForms("topping", values);
  printf("<li> <code>topping = %s </code>\n", value[0]);
  printf("<li> <code>topping = %s </code>\n", value[1]);
  printf("</ul>\n");
  Response.end();
}
```

Program 3.2: A CGI program form.ch for processing a fill-out form.

`#!/bin/ch`

shall be added at the beginning of the program. For an illustrative purpose, in this example, program form.ch only decodes the user's input, and generates a dynamic Web page in HTML format. The programming statement

```
Response.setContentType("text/html");
```

indicates that the content type of the output is a text in HTML format. Other content types such as plain text or graphics can also be generated dynamically. By default, the content type is "`text/html`" from Ch-CGI. In Ch-CGI, all properties of a CResponse object such as content type must be set up before the function **CResponse**::**begin**() is called, which starts output. For the content type of html text, member function **CResponse**::**title**() can be used conveniently to add title and body tags. The member function **CResponse**::**title**() is implemented with the following source code.

```
void CResponse::title(char* titleName) {
   printf("<html>\n");
   if (titleName != NULL)
      printf("<head> <title> %s </title></head>\n", titleName);
    printf("<body bgcolor=\"#FFFFFF\">\n");
}
```

If more elaborative output is desired, the user may write header information directly using function **printf**() instead of using function **CResponse**::**title**(). If the content type is not "`text/html`", member function **CResponse**::**title**() shall not be used.

The names and values submitted in a fill-out form can be obtained in the CGI program by member functions **CResponse**::**getForm**(), **CResponse**::**getForms**(), or **CResponse**::**getFormNameValue**(). Because

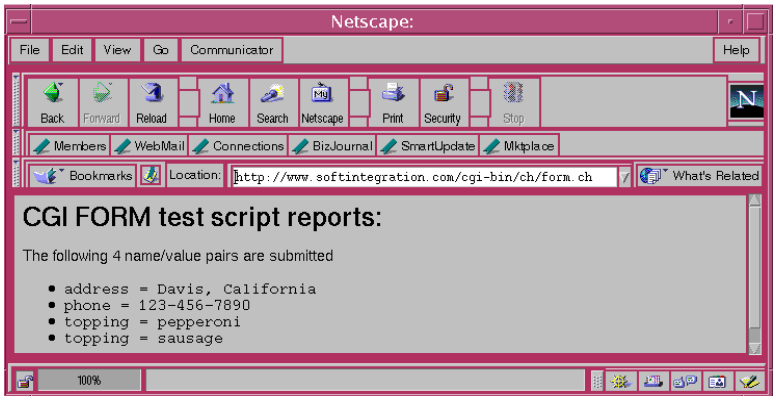


Figure 3.2: Dynamic Web page in html format generated by CGI program form.ch.

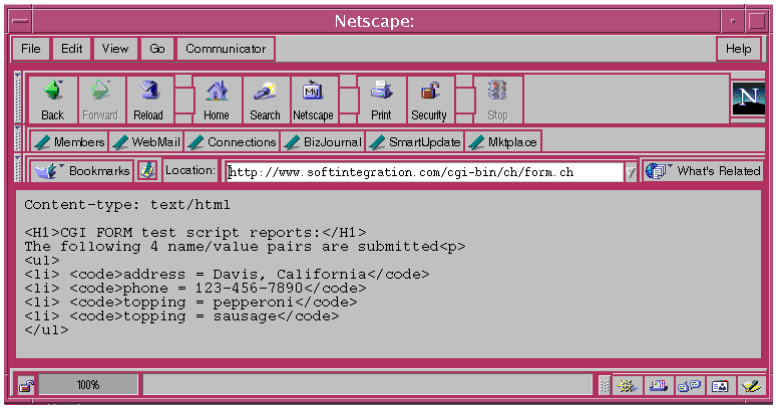


Figure 3.3: Dynamic Web page with plain text generated by CGI program form.ch with option `-g`.

there are multiple values associated with name `topping`, member function **CResponse**::**getForms**() is used in the program `form.ch`. A CResponse object will be terminated by the function **CResponse**::**end**(). The Web page in html file format generated dynamically by the CGI program form.ch is shown in Figure 3.2.

A CGI program with -g option turns the Web browser into a text console. The CGI program will print output in plain text on the Web browser for the convenience of debugging. For example, option `-g` can be used in the first line of CGI program **form.ch** in Program 3.2 by changing

```
#!/bin/ch
```

to

```
#!/bin/ch -g
```

Although the content type is text/html, the output is displayed as a plain text as shown in Figure 3.3 for the output of the modified CGI program with option `-g`.

Note: The latest version of IIS in Windows will display error messages in the browser correctly without using option -g for CGI.

3.4 Verbatim Output Blocks Using fprintf

A block of the verbatim output can be achieved using the feature of function **fprintf**. The syntax for a block of verbatim output is

`fprintf` *stream* << *TERMINATOR*
`...`
TERMINATOR

or

`fprintf` *stream* << "*TERMINATOR*"
`...`
TERMINATOR

where *stream* is a valid file stream and terminator *TERMINATOR* is a valid identifier that have not been used as a keyword or variable name in the program. Macro names, such as "END", can be used as the terminator, since they are processed verbatim without macro expansion in this case. It is recommended that an identifier of all capital letters is used. The verbatim block output using **fprintf** has the following constraints.

- White spaces and comments can follow the first terminator.
- White spaces can precede the second terminator.
- The second terminator shall be terminated with a new line character. No character, even a white space, is allowed to appear after the second terminator.
- All characters, including white characters and comments, between the first and last lines are processed verbatim.
- The first terminator can be enclosed in double-quotes, whereas the second shall not. If the first terminator is enclosed in double-quotes, the dollar sign '$' within the enclosing block will be treated verbatim. Otherwise, the single dollar sign '$' is used for variable or expression substitution. Two syntaxes of

  $*var* and ${*var*}

  can be used for variable substitution. The variable name or symbol to be expanded may be enclosed in braces, which are optional but serve to protect the variable to be expanded from characters immediately following it which could be interpreted as part of the name.

  The variable in a variable substitution could be a predefined identifier; a user-defined variable of string, pointer to char, integral, floating-point, or complex data type; an environment variable; or undefined symbol. For a variable substitution, the Ch shell will first search the Ch name space for the variable name according to its scope rule. If the variable is not defined, then it searches the environment variables of the current process. If no variable with the specified name is found either in Ch space or environment space, no substitution will take place and the variable is ignored.
- Expression substitution in the form of

  ```
  $(expression)
  ```

  can be used to substitute the valid Ch expression with its result. *The expression* shall be an expression of string, pointer to char, integral, floating-point, or complex data type.
- The variable or expression substitution can be prevented by preceding the '$' with a '\'. A '$' is passed unchanged if followed by a blank, tab, or end-of-line.

Program 3.3: Generating an html file.

Program 3.4: Using fprintf for a block output.

- A value through variable substitution or expression substitution will be printed out using a default format control string for its data type.

For example, if the program `verbat.ch` consists of the following programming statements,

```
#include <stdio.h>
int sum = 2
fprintf stdout << END /* This is a comment */
  /* this is verbatim output */
  sum = \$$sum
  sum + 1 = \$$(sum+1)
 END
```

The result from executing `verb.ch` is shown as follows.

```
> verbat.ch
    /* this is verbatim output */
    sum = $2
    sum + 1 = $3
>
```

In command

```
sum = \$$sum
```

the escape character '`\`' is used to print out as a single dollar sign, and the symbol `$sum` is substituted with the value of `sum`, i.e. 2. In the next command

```
sum + 1 = \$$(sum+1)
```

the symbol `$(sum+1)` indicates an expression substitution. It is replaced by the result of the expression `sum+1`, i.e. 3. The comment following the first terminator `END` and the white spaces preceding the second `END` are ignored. But, the comment inside the block is printed out verbatim.

By default, a variable of double type is printed out with four digits after the decimal point whereas a variable of float type is printed out with two digits after the decimal point. To print out a variable of double, one may cast it to float before printing it out if the value is within the representable range of float type. For example, $((float)d) can be printed out with two digits after decimal point, $((int)d) with integral part only.

Often time, a block of HTML code needs to be sent as a standard output stream in a CGI program. For example, Program 3.3 will generate the code below,
which displays the text `Hello, world` in a web browser. According to the HTTP protocal, the line

```
Content-Type: text/html
```

must start without any white space, and there must be only an empty line without white space following it. Using the verbatim output feature, the above Ch CGI program can be simplied as Program 3.4. Note that the value of `hello` is retrieved by using the dollar sign $ inside the verbatim output block,

As another example, the function `sendApplet()` below generates a C program.

```
void sendApplet(char *x, char *y, char *expr) {
    fprintf(stdout, "#include<stdio.h>\n");
    fprintf(stdout, "int main() {\n");
    fprintf(stdout, "  double x = %s;\n", x);
    fprintf(stdout, "  double y = %s;\n", y);
    fprintf(stdout, "  printf(\"x = %%f, \", x);\n");
    fprintf(stdout, "  printf(\"y = %%f \\n\", y);\n");
    fprintf(stdout, "  printf(\"%s = %%f\\n\", %s);\n", expr, expr);
    fprintf(stdout, "}\n");
}
```

This function `sendApplet()` can be rewritten in Ch as follows.

```
void sendApplet(char *x, char *y, char *expr) {
    fprintf stdout << ENDFILE
      #include<stdio.h>
      int main() {
        double x = $x;
        double y = $y;
        printf("x = %f", x);
        printf("y = %f\n", y);
        printf("$expr = %f\n", $expr);
      }
    ENDFILE
}
```

where the values of variables x, y and $expr$ are obtained using operator $.

3.5 Dynamic Web Plotting

Plotting through CGI programs is very useful for many Web-based applications. With Ch Professional Edition and CGI toolkit, plots can be very easily generated dynamically on-line. How to generate a dynamic plot will be presented in this section. We will also describe how data is encoded and decoded for transferring among the browser, Web server, and CGI programs.

In a Web-based plotting, the parameters for plotting are submitted from a Web browser, shown in Figure 3.4, with its corresponding HTML file in Program 3.5 and encoded by the browser. The parameters as name-value pairs are decoded by member function **CRequest**::**getFormNameValue**() in first CGI program **webplot1.ch** shown in Program 3.6. They are then passed as query strings to the second CGI program **webplot2.ch** shown in Program 3.7. These parameters are obtained again using member function **CRequest**::**getFormNameValue**(). The plot generated as a PNG file and displayed through a Web browser is shown in Figure 3.5 .

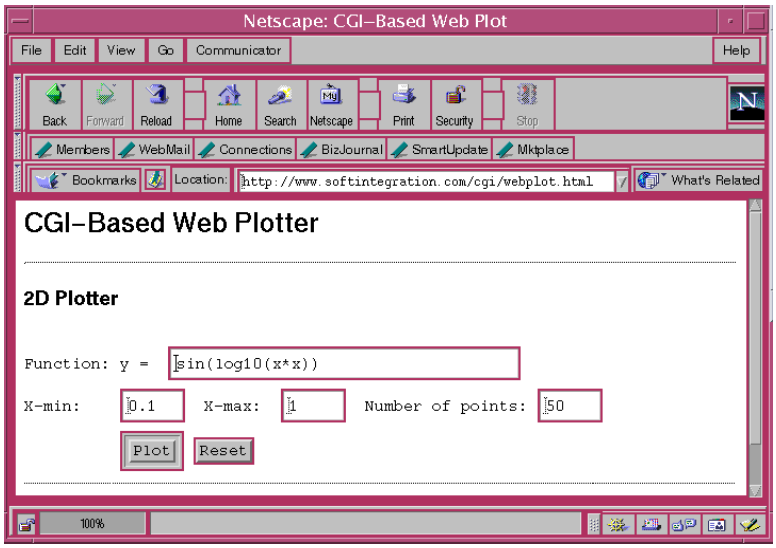


Figure 3.4: A Web-plotter based on the fill-out form.

```
<HTML>
<HEAD>
<TITLE>
CGI-Based Web Plot
</TITLE>
</HEAD>
<BODY bgcolor="#FFFFFF" text="#000000" vlink="#FF0000">
<H1>
CGI-Based Web Plotter
</H1>

<HR>
<H2>2D Plotter</H2>
<PRE>
<FORM method="post" action="/cgi-bin/chcgi/toolkit/demos/sample/webplot1.ch">
Function: y =  <INPUT name="expression" value="sin(log10(x*x))" size=35>
X-min:    <INPUT name="xMin" value="0.1" size=5>  X-max:  <INPUT name="xMax"
value="1" size=5>  Number of points: <INPUT name="numpoints" value="50" size=5>
          <INPUT type="submit" value="Plot"> <INPUT type="reset" value="Reset">
<HR>
</BODY>
</HTML>
```

Program 3.5: HTML file for submitting plotting parameters.

```
#!/bin/ch
#include <cgi.h>

int main() {
   int i, num;
   chstrarray name, value;
   class CResponse Response;
   class CRequest Request;
   class CServer Server;

   num = Request.getFormNameValue(name, value);
   Response.setContentType("text/html");
   Response.begin();
   Response.title("Web Plot");
   printf("<center>\n");
   printf("<img src=\"/cgi-bin/chcgi/toolkit/demos/sample/webplot2.ch");
   for (i=0; i<num; i++){
      putc(i == 0 ? '?' : '&', stdout);
      fputs(Server.URLEncode(name[i]),stdout);
      putc('=', stdout);
      fputs(Server.URLEncode(value[i]),stdout);
   }
   printf("\">\n");
   printf("</center>\n");
   Response.end();
}
```

Program 3.6: CGI program webplot1.ch

```
#!/bin/ch
#include <cgi.h>
#include <chplot.h>

int main() {
   double MinX, MaxX, Step, x, y;
   int pointsX, pointsY, i;
   chstrarray name, value;
   class CResponse Response;
   class CRequest Request;
   class CPlot plot;

   Request.getFormNameValue(name, value);
   MinX = atof(value[1]);
   MaxX = atof(value[2]);
   pointsX = atoi(value[3]);
   double x1[pointsX], y1[pointsX];

   Step = (MaxX - MinX)/(pointsX-1);
   for(i=0;i<pointsX;i++) {
     x = MinX + (i*Step);
     y = streval(value[0]);
     x1[i] = x;
     y1[i] = y;
   }

   Response.setContentType("image/png");
   Response.begin();
   plotxy(x1, y1, value[0], "X", "Y", &plot);
   /* output plot in color png file format */
   plot.outputType(PLOT_OUTPUTTYPE_STREAM, "png");
   plot.plotting();
   Response.end();
}
```

Program 3.7: CGI program webplot2.ch

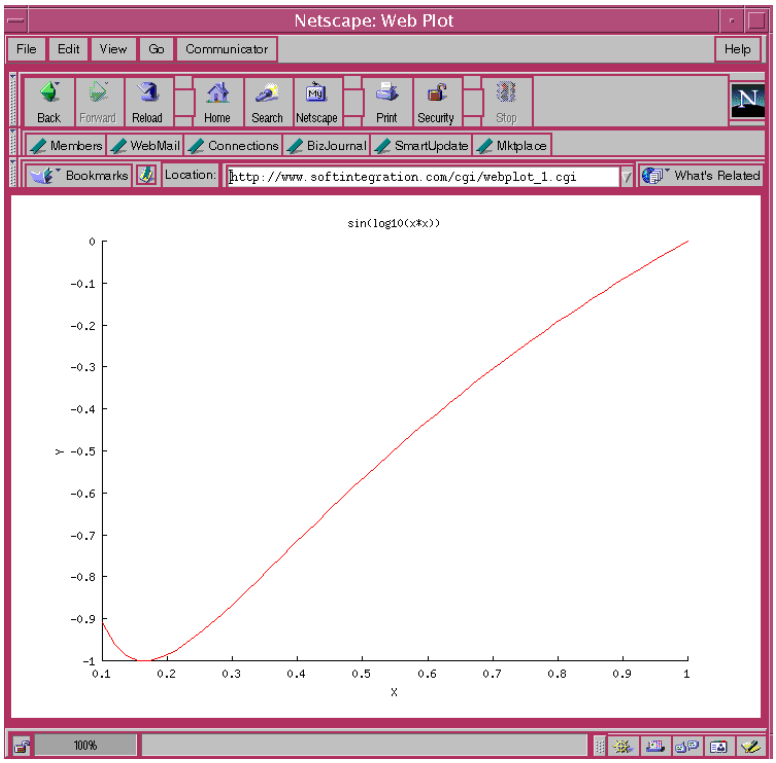


Figure 3.5: Plot generated through the Web plotting.

3.6 Uploading Files to a Web Server

Many applications need to upload files to a Web server. For example, files can be uploaded and attached to an email, then sent through a Web server. Images can be uploaded to a Web server for image processing. Papers can be submitted to a conference or journal through a Web browser. How to upload files to a Web server from the client machine using a Web browser using Ch CGI will be illustrated by an example. In this example, we assume that the user is asked to submit other information besides uploading a file as shown in Figure 3.6.

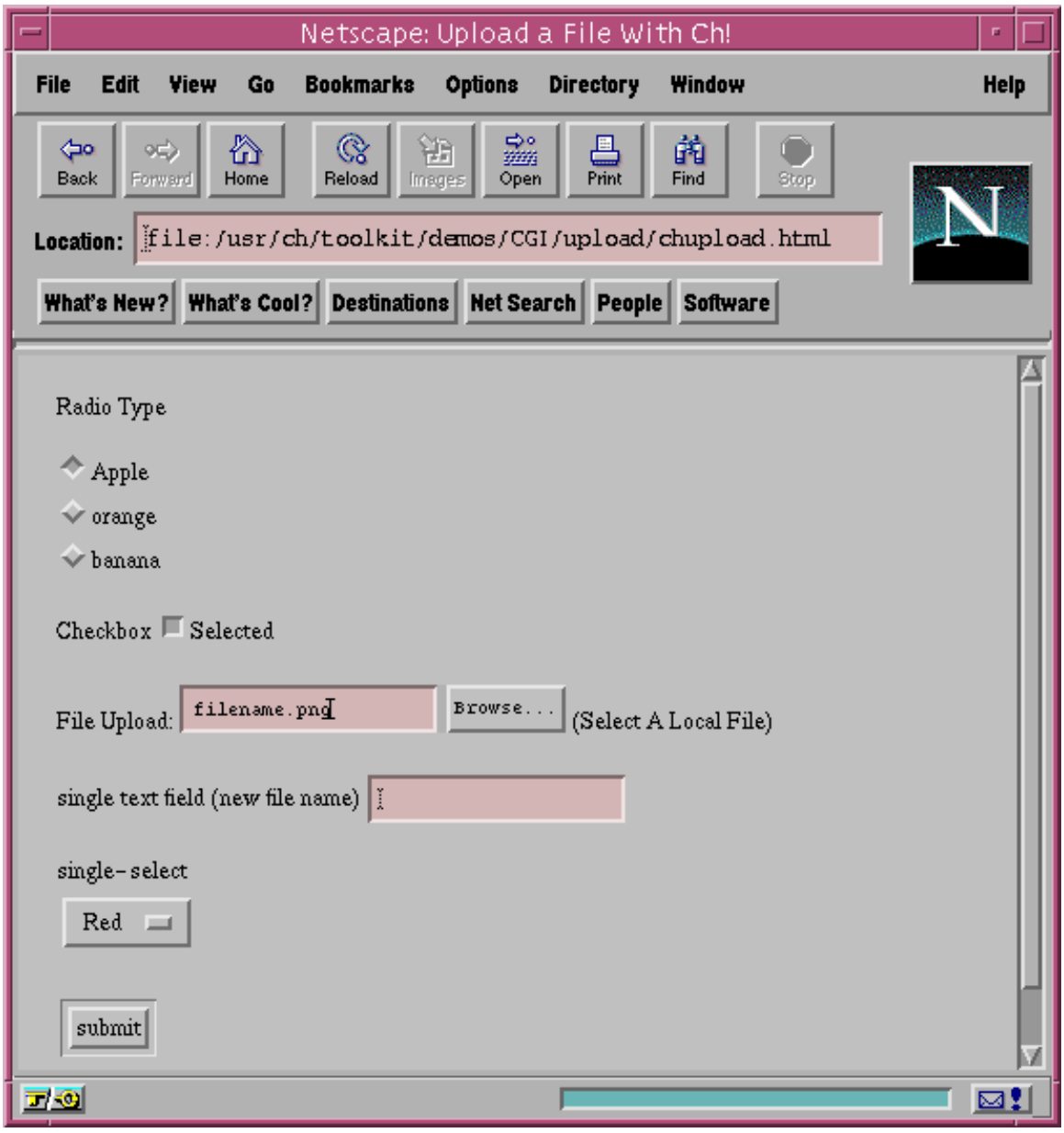


Figure 3.6: A fill-out form for uploading a file.

In this example, it is assumed that the user will upload a PNG image file named *filename.png* typed in the fill-out form. The user can also select and upload other files by browsing the file system in his computer. The corresponding HTML file for the user interface shown in Figure 3.6 is given in Program 3.8.

```
<html>
<head>
<title>Upload a File With Ch!</title>
</head>
<body>
<form action="/cgi-bin/upload/chupload.ch" enctype="multipart/form-data" method="post">
<p>
Radio Type
<p>
<input type="radio" name="food" value="apple" checked> Apple<br>
<input type="radio" name="food" value="orange"> orange<br>
<input type="radio" name="food" value="banana"> banana<br>
<p>
Checkbox <input type="checkbox" name="selected" checked> Selected
<p>File Upload:
<input type="file" name="file" value=""> (Select A Local File)
<p>
single text field (new file name) <input type="text" name="newfile" value="">
<p>
single-select
<br>
<select name="colors">
<option value="Red">Red
<option value="Green">Green
<option value="Blue">Blue
</select>
<P>
<input type="submit" value="submit" />
</form>
</body>
</html>
```

Program 3.8: The HTML file for uploading a file.

Depending on the size of file to be uploaded as well as the network speed and traffic, it may take a while to load a large file. In such a case, it is desirable that the Web brower will display an informative message that shows that the file uploading is in progress. This can be accomplished by using the HTML File shown in Program 3.9, which uses a Javascript to display the waiting message while the web browser is uploading a file to the Web server.

```
<html>
<head>
<title>Upload a File With Ch!</title>
<noscript>
<META HTTP-EQUIV=Refresh CONTENT="0; URL=/no_javascript.html">
</noscript>

<script>
function KeyPress() {
   if (event.keyCode == 13) {
       BeginAttach();
   }
}
function BeginAttach() {
   document.all.before_attach.style.display = "none";
   document.all.middle_attach.style.display = "inline";
   document.attachfile.submit();
}
</script>
</head>
<body>
<form name="attachfile" enctype="multipart/form-data" method=POST
action="/cgi-bin/upload/chupload.ch">

<span id=middle_attach style="display:none">

<table cellpadding=4 cellspacing=0 border=0 width="100%">
<tr class=frmt><td><b>Attaching file...</b></td></tr>
<tr class=frmb><td>Please wait while we attach the file to your message.</td></tr>
</table>
</span>
<span id=before_attach>
<p>
Radio Type
<p>
<input type="radio" name="food" value="apple" checked> Apple<br>
<input type="radio" name="food" value="orange"> orange<br>
<input type="radio" name="food" value="banana"> banana<br>
<p>
Checkbox <input type="checkbox" name="selected" checked> Selected
<p>File Upload:
<input type="file" name="file" value="" onkeypress="KeyPress()"> (Select A Local File)
<p>
single text field (new file name) <input type="text" name="newfile" value="">
<p>
single-select
<br>
<select name="colors">
<option value="Red">Red
<option value="Green">Green
<option value="Blue">Blue
</select>
<P>
<input type="submit" value="submit" onclick="BeginAttach()"/>
</form>
</span>
</body>
</html>
```


Program 3.9: HTML file for uploading a file with a waiting message.

```
#!/bin/ch
/* This CGI program will process fill-out form and upload a file, say 'filename.png'.
   The file will be loaded at C:/filename in Windows, and /tmp/filename.png in Unix */

#include <cgi.h>
class CResponse Response;
class CRequest Request;
char* getNameValue(char *infoStr, char* fileField);
char* getBoundary();
#define RETURN_VAL "\r\n\r\n"

int main() {
    Response.begin();
    Response.title("Test of Upload");

    size_t total = Request.getTotalBytes();
    char *binData = Request.binaryRead(&total);
    char* boundary = getBoundary();

   /* seperate uploaded file data from header and tail with posted info */
   char *headInfo=binData;
   char *head = strstr(binData, RETURN_VAL);
   char* binDataHead;

   /* make headInfo contains posted head, and binDataHead the start of uploaded file */
   for (head=strstr(binData, RETURN_VAL); head!= NULL;
        head = strstr(head, RETURN_VAL))
   {
      head = head + strlen(RETURN_VAL);
      if ( strstr(head, "\"; filename=\"") == NULL)
      {
           binDataHead = head;//binData contains data only
           *(head-1)='\0';  // make headInfo having data of header only
           break;
      }
   }
```

Program 3.10: CGI program chupload.ch for uploading files to a Web server.

```
int boundaryLength = strlen(boundary);
int bmove = 0;
char *tempBinHead = binDataHead;
int fileLen=0;
/* get the uploaded file length in fileLen */
while (1) {
      if ( *tempBinHead == boundary[bmove]) {
          // matched the end of boundary
          bmove++;
          tempBinHead++;
          if (bmove == boundaryLength) {
              // get rid of additional tailer ahead of boundary
              fileLen -= 4;
              break;
          }
      } else if (bmove > 0) {
          /* the match may start in the middle */
          tempBinHead = tempBinHead - bmove +1 ;
          fileLen++;
          bmove = 0;
      } else {
          tempBinHead++;
          fileLen++;
      }
}

// all data after binDataHead+fileLen belongs to posted tail data
//  take out of file and combins head and tail together
string_t  totalHead;

//int i= sprintf(totalHead, "%s%s", headInfo, tempBinHead);
strcpy(totalHead, headInfo);
strcat(totalHead, tempBinHead);
printf("totalhead:string: %s  <br>", totalHead);
```

Program 3.10: CGI program chupload.ch for uploading files to a Web server (continued).

```
    char* userAgent = Request.getServerVariable("HTTP_USER_AGENT");
    // get the uploaded file path and name
    char* filename = getNameValue(totalHead, "filename=");
    char* sName;
    if ( filename != NULL)
    {
        printf("uploaded filename:[%s]<br>\n", filename);
        printf("file size:[%d]<br>\n", fileLen);

        if (strstr(userAgent, "Win") != NULL) // from windows
        {
            sName=strrchr(filename, '\\');
            if (sName  != NULL)
                sName++;
            else //Netscape 6.0 or up
                sName = filename;
        }
        else
        {
        // from "Mac" or "Unix", just be careful some Mac char is illegal
            sName=filename;
        }

        printf("sName:[%s]<br>\n",sName);
        #ifdef _WIN32_
           string_t fullPath= stradd("c:\\", sName);
        #else
           string_t fullPath=stradd("/tmp/", sName);
        #endif
        printf("fullPath:[%s]<br>\n",fullPath);

       FILE* fp=fopen(fullPath, "wb");
       fwrite(binDataHead, fileLen, sizeof(char), fp);
       fclose(fp);
   }

   char* value = getNameValue(totalHead, "Content-Type");
   printf("Content Type::[%s]<br>\n",  value);
   value= getNameValue(totalHead, "food");
   printf("food:[%s]<br>\n", value);
   value= getNameValue(totalHead, "selected");
   if ((value != NULL) && (strcmp(value, "on") == 0))
      printf("selected success:[%s]<br>\n", value);
   else
      printf(" nont selected <br>");
   value= getNameValue(totalHead, "colors");
   printf("colors:[%s]<br>\n", value);
   value= getNameValue(totalHead, "newfile");
   printf("new filename:[%s]<br>\n", value);
   Response.end();
}
```

Program 3.10: CGI program chupload.ch for uploading files to a Web server (continued).

```
/* infoStr: contains the input of string containing pair
   fileField: contains name
   return:  the value of fileField name
*/
char* getNameValue(char *infoStr, char* fileField)
{
    int len, resultLen, extraLen;
    char  *startPos, *endPos;
    char * result;

    len = strlen(fileField);

    for (startPos = strstr(infoStr, fileField);
         startPos !=NULL;
         startPos = strstr(startPos+1, fileField))
    {
     // guess for handling name/value pair
        if ( strncmp( (startPos + len-1), "=\"", 2) == 0)
        {
            extraLen= 1;
            break;
        }
        else if ( strncmp( (startPos + len), ": ", 2) == 0)
        {
            extraLen= 2;
            break;
        }
        else if (strncmp( (startPos + len), "\"\r\n\r\n", 5) == 0)
     // guess for handling name/value pair
        {
            extraLen= 5;
            break;
        }
     }

     if (startPos != NULL)
     {
         endPos = startPos + len + extraLen;
         resultLen =0;
       // code needs to modify if handling multi-line text or multi-select
         while ((*endPos !='"') && ( *endPos !='\r') && (*endPos != '\n'))
         {
               resultLen++;
               endPos++;
          }
         if ( resultLen !=0 )
         {
            result= malloc(sizeof(char)*(resultLen+1));
            strncpy(result, startPos+len+extraLen, resultLen);
            result[resultLen]='\0';
            return result;
         }
     }
    return NULL;
}
```

Program 3.11: CGI program chupload.ch for uploading files to a Web server (continued).

```
char* getBoundary()
{
    char* contentType = Request.getServerVariable("CONTENT_TYPE");
    char* bound;
    char* startPos;

    for ( startPos=strchr(contentType, ';'); startPos!=NULL;
          startPos= strchr(startPos, ';'))
    {
        *startPos = '\0';
        startPos++;
        if ( (strstr(startPos, "boundary="))!= NULL ) {
            char *str;
            startPos = strstr(startPos, "boundary=");
            bound = startPos + strlen("boundary=");
            str = bound;
            while ((*str) && (!isspace(*str))) {
                str++;
            }
            *str = '\0';
            break;
        }
    }
   return bound;
}
```

Program 3.12: CGI program chupload.ch for uploading files to a Web server (continued).

The user interface for both Programs 3.8 and 3.9 is the same as shown in Figure 3.6. They both use the same CGI program shown in Program 3.10. Program 3.10 will process the names and their corresponding values submitted through a fill-out form, and print them out as text in HTML file format. A file uploaded from the Web browser will be saved with the same file name in the directories C:/ and /tmp for Windows and Unix, respectively.

3.7 Cookies for Personalized Content

3.7.1 What Is Cookie

Cookie is a general mechanism by which the server side of the connection can both store and retrieve information on the client side. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications.

A server, when returning an HTTP object to a client, may also send a piece of state information which the client will store. Included in that state object is a description of the range of URLs for which that state is valid. Any future HTTP requests made by the client which fall in that range will include a transmittal of the current value of the state object from the client back to the server. The state object is called a cookie.

This simple mechanism provides a powerful new tool which enables a host of new types of applications to be written for Web-based environments. Shopping applications can now store information about the currently selected items, for fee services it can send back registration information and free the client from retyping a user-id on next connection. Sites can store per-user preferences on the client, and have the client supply those preferences every time that site is connected to.

3.7.2 Properties of a Cookie

A cookie is introduced to the client by including a Set-Cookie header as part of an HTTP response. The Ch cookie class supports both the Version 0 (by Netscape) and Version 1 (by RFC 2965 which obsoletes RFC 2109). By default, cookies uses version 0. Since RFC 2965 is released on October 2000, and most browsers might not support the RFC 2965, the users are encouraged to use Version 0 features. All samples we discussed in this section cover the version 0 only.

Typically the syntax of the Set-Cookie HTTP Response Header which includes some important properties of the cookie is:

```
Set-Cookie: NAME=VALUE; expires=DATE; path=PATH;
domain=DOMAIN_NAME; secure
```

`NAME=VALUE`
This string is a sequence of characters excluding semi-colons, commas and white spaces. If there is a need to place such data in the name or value, some encoding method such as URL style encoding is recommended, though no encoding is defined or required. This is the only required property on the Set-Cookie header.

`expires=DATE`
The expires attribute specifies a date string that defines the valid life time of that cookie. Once the expiration date has been reached, the cookie will no longer be stored or given out. The date string is formatted as:

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

This is based on RFC 822, RFC 850, RFC 1036, and RFC 1123 with the variations that the only legal time zone is GMT and the separators between the elements of the date must be dashes. Expires is an optional attribute. If not specified, the cookie will expire when the user's session ends.

In Ch-CGI, for the convenience of users, the property of MaxAge instead of expires is used.

Note that there is a bug in Netscape Navigator version 1.1 and earlier. Only cookies whose path attribute is set explicitly to "/" will be properly saved between sessions if they have an expires attribute.

`domain=DOMAIN_NAME`
When searching the cookie list for valid cookies, a comparison of the domain attributes of the cookie is made with the Internet domain name of the host from which the URL will be fetched. If there is a tail match, then the cookie will go through path matching to see if it should be sent. "Tail matching" means that domain attribute is matched against the tail of the fully qualified domain name of the host. A domain attribute of "softintegration.com" would match the host name "www.softintegration.com".

Only hosts within the specified domain can set a cookie for a domain and domains must have at least two (2) or three (3) periods in them to prevent domains of the form: ".com", ".edu", and "va.us". Any domain that falls within one of the seven special top level domains listed below only require two periods. Any other domain requires at least three. The seven special top level domains are: "COM", "EDU", "NET", "ORG", "GOV", "MIL", and "INT".

The default value of domain is the host name of the server which generated the cookie response.

`path=PATH`
The path attribute is used to specify the subset of URLs in a domain for which the cookie is valid. If a cookie has already passed domain matching, then the pathname component of the URL is compared with the path attribute, and if there is a match, the cookie is considered valid and is sent along with the URL request. The path "/foo" would match "/foobar" and "/foo/bar.html". The path "/" is the most general path. If the path is not specified, it as assumed to be the same path as the document being described by the header which

contains the cookie.

`secure`
If a cookie is marked secure, it will only be transmitted if the communications channel with the host is a secure one. Currently this means that secure cookies will only be sent to HTTPS (HTTP over SSL) servers. If secure is not specified, a cookie is considered safe to be sent over unsecured channels.

In addition, properties of Port, Discard, Version, Comment and CommentURL are added for the cookies of version 1. In Ch-CGI, some of these properties can be set and retrieved by member functions of class CCookie.

3.7.3 How to Set a Cookie

In Ch-CGI, CCookie class is designed for setting and getting properties of a cookie. In Program 3.13, member functions of **setName()**, **setValue()**, **setMaxAge()**, **setDomain()**, **setPath()** and **setSecure()** are used to set properties of cookies. After these properties are set, **CResponse.addCookie()** sends cookie with these properties to the client.

3.7.4 How to Get Cookies

In Program 3.14, **CRequest::getCookies()** retrieves all cookies with properties from the client. Member functions **getName()** and **getValue()** are used to get names and values of cookies.

If you know the name of the cookie which you want to get from the client in advance, the function of **CRequest::getCookie()** can be used. It retrieves the cookie from the client by the name. The syntax is shown below.

```
chchar *cookieValue = Request.getCookie(cookieName);
```

The example of using **CRequest::getCookie()** is shown in Program 3.15

3.8 Tips for Debugging CGI Programs

When you debug and run a CGI program, you will encounter some error messages. In general, it is more difficult to debug a CGI program than a regular program because a CGI program is run in the account of a Web server, instead of regular user account. For example, you may see an error message such as the following one from Netscape browser:

Server Error

```
This server has encountered an internal error which prevents it
from fulfilling your request.  The most likely cause is a
misconfiguration.  Please ask the administrator to look for messages
in the server's error log.
```

The messages in the Web server's error log file may appear as follows:

```
[06/Feb/1996:16:38:08] failure:  for host your.host.id.num.  trying
to GET /your/path/your.ch, cgi-parse-output reports:  the CGI program
/your_absolute_cgi_path/your.ch did not produce a valid header (program
terminated without a valid CGI header.  Check for core dump or other
abnormal termination)
```

```
#!/bin/ch
#include <stdio.h>
#include <cgi.h>
int main()
{
  class CCookie ck, ck2;
  class CResponse Response;

  // put Cookie with name of testCookie, key of CookieKey and value of Cookievalue
  ck.setName("testCookie");
  ck.setPath("/");
  ck.setValue("CookieValue");
  ck.setMaxAge(600);
  ck.setDomain("edu");
  ck.setSecure(false);
/*
Because there is no browser to support the following properties as yet,
it is not recommended to use them.

  ck.setVersion(1);
  ck.setMaxAge(600);
  ck.addPort(8080);
  ck.setComment("This cookie is for test");
  ck.setCommentURL("mailto:someone@softintegration.com");
  ck.setDiscard(false);
  ck.setDomain("softintegration.com");
*/
  Response.addCookie(&ck);

  ck2.setName("testCookie2");
  ck2.setPath("/");
  ck2.setValue("CookieValue2");
  Response.addCookie(&ck2);

  ck2.setName("cookie name");
  ck2.setPath("/");
  ck2.setValue("cookie value");
  Response.addCookie(&ck2);
  Response.begin();

  printf("Cookie: name=testCookie, value=CookieValue   has been added<br>\n");
  printf("Cookie: name=testCookie2, value=CookieValue2 has been added<br>\n");
  printf("Cookie: name=cookie name, value=cookie value has been added\n");

  Response.end();
}
```

Program 3.13: Add a cookie to client.

```
#!/bin/ch
#include <cgi.h>

int main()
{
  class CCookie *pck;
  class CResponse Response;
  class CRequest Request;
  int i, count;

  Response.begin();
  Response.title("Test of Request.getCookies");
  printf("<H1> Test of Request.getCookies </H1><hr>\n");

  //get Cookies
  count = Request.getCookies(&pck);
  for(i=0; i < count; i++)
    printf("%s = %s <br>\n", pck[i].getName(), pck[i].getValue());

  Response.end();
}
```

Program 3.14: Get cookies from client.

```
#!/bin/ch
#include <cgi.h>

int main()
{
  class CResponse Response;
  class CRequest Request;
  chchar name[] = "testCookie";
  chchar *value;

  Response.begin();
  Response.title("Test of Request.getCookie");
  printf("<H1> Test of Request.getCookie </H1><hr> \n");

  //get Cookie
  value = Request.getCookie(name);
  printf("%s = %s <br>\n", name, value);
  value = Request.getCookie("cookie name");
  printf("%s = %s <br>\n", "cookie name", value);
  Response.end();
}
```

Program 3.15: Get a cookie from client by name.

This error message basically indicates that your CGI program did not produce a valid CGI header such as

```
Content-type: text/html
```

or

```
Content-type: text/plain
```

To debug your Ch CGI program, the following steps shall be taken.

- Make sure the first output statement from the CGI program will produce a valid CGI header such as the one described above.
- Make sure your program can run successfully from your terminal prompt by just typing the program name, say, `your_program.ch`. Sometimes, your program may terminate in the middle of execution if the flow of the program depends on some environment variables passed from the Web server. It is fine if this is the case. For example, member function CRequest::getFormNameValue() depends on the environment variables REQUEST_METHOD, CONTENT_LENGTH, and QUERY_STRING. If your CGI program is not invoked by the Web server, these environment variables are not set. The values supposedly passed from a FORM will be NULL, and the program may be terminated prematurely if run from a terminal. But, the program can be run successfully from the Common Gateway Interface.
- Since your CGI program is executed by the Web server, which uses a different user account from yours, make sure your program is readable and executable by other users in the system. In Unix, you can change the permission of your program by the following commands

  ```
  chmod 755 your_program.ch
  ```

  If Ch is not installed by a superuser or administrator account, make sure CHHOME/bin/ch is executable by other users. Subdirectories and files in the Ch home directory should also be accessible by others so that relevant modules such as header file **cgi.h** can be accessed by the Web server. The environment variable CHHOME can be setup at the time before the Ch CGI code is executed by the command below.

  ```
  #!/bin/sh
  env CHHOME=/your/Ch/home/dir /path/to/cgi-bin/ch_program.ch
  ```

  The following CGI code can be used to find the user name and home directory where the startup file **.chrc** in Unix and **_chrc** in Windows is loated for running the web server.

  ```
  #!/bin/ch
  printf("Content-Type: text/plain\n\n");
  printf("The user running the Web server = %s\n", _user);
  printf("The home directory of the user running the Web server = %s\n", _home);
  ```

- In Windows, even though the current directory is specified in the search paths for header files and function files in system variables **_ipath** and **_fpath**, respectively, in the web server's startup file **_chrc**. A CGI program may still not be able to find the header file and functions in the current directory. In this case, the directory for the header file and functions need to be specified explicityly in the startup file. For example, assume the home directory for IIS in Windows is located at `C:/inetpub`. To setup Web-based Ch Control System Toolkit, the following two programming statements may need to be added

```
_ipath = stradd(_ipath, "C:/inetpub/cgi-bin/chcgi/toolkit/control;");
_fpath = stradd(_fpath, "C:/inetpub/cgi-bin/chcgi/toolkit/control;");
```

in the startup file **_chrc** in the home directory of the web server account.

- If your CGI program reads and writes a file, make sure the file is readable and writable by the account of the Web server. Often, you may need to create a temporary file using the function tmpnam() in Ch, if a temporary file is needed in the CGI program.
- If you can login as the Web server, test your CGI program from the account of the Web server. You may need to get permission from your system administrator to do so.
- By default, only CGI programs located in the server's cgi-bin can be executed by the Web server. If your CGI program is not located in cgi-bin and your Web server is not configured with a file association to recognize CGI programs with file extension `.ch`, your Ch program will fail under the Common Gateway Interface. Check with your system administrator about the setup of your Web server.
- If your CGI program invokes other Ch programs, you may need to add the following function call

  ```
  setbuf(stdout,NULL);
  ```

  or

  ```
  setvbuf(stdout, NULL, _IONBF, 0);
  ```

  before the first printing statement of your CGI program. This function will cause output to be flushed immediately instead of being buffered so that the output will be sent in a proper sequence. By default, the member function **CResponse**::**begin**() executes this function automatically.
- For Web servers in Unix or Apache Web server in Windows, if you use a C program as your CGI program, make sure to add the following line as the first statement of the program.

  ```
  #!/bin/ch
  ```

  For Web servers in Windows such as Microsoft Personal Web Server, Microsoft Information Server, or Netscape Web Server, the above line is not necessary. For portability, it is recommended that the above line be added for all Ch CGI programs.
- Unknown Commands in a CGI Program. A Ch CGI program is normally run in regular Ch shell. The first line

  ```
  #!/bin/ch
  ```

  of your CGI program indicates that the program will be executed in a regular Ch shell. The path for commands executable by the Ch shell is controlled by the system-wide startup file **CHHOME/config/chrc**, which includes startup file **.chrc** in the home directory of the Web server (not the home directory of your regular user account)
- Turn your Web browser into a display console by adding the debug option -g at the first line of your CGI code as

```
#!/bin/ch -g
```

This is one of the most useful features for debugging Ch CGI code.

- A CGI code for Ch plotting may produce a HTML output similar to what is shown below

```
<img src="/cgi-bin/chcgi/toolkit/demos/sample/plot2_1.ch">
```

The above code shall display an image generated by the CGI program `plot2_1.ch`. In case, it does not display a plot, you can add

```
#!/bin/ch -g
```

at the beginning of the program `plot2_1.ch` and test it directly using the following URL address

```
http://your_web_server_address.com/cgi-bin/chcgi/toolkit/demos/sample/plot2_1.ch
```

If there any additional arguments following the program `plot2_1.ch`, it should also be typed as part of the URL address such as.

```
http://your_web_server_address.com/cgi-bin/chcgi/toolkit/demos/
         sample/webplot2.ch?expression=x*x&xMin=0.1&xMax=6&npoints=50
```

- Read the on-line tutorial on the WWW at `http://www.softintegration.com` about how to write Ch CGI programs.

Chapter 4

References for CGI Classes

Common Gateway Interface (CGI) is a standard that specifies how external programs interface with a web server. One of the most important applications of CGI is handling fill-out form. A CGI program located on a host machine's web server can accept user's input through a fill-out form and generate web pages dynamically.

Header file **cgi.h** contains the definition of the **CResponse**, **CRequest**, and **CServer** classes, and their member functions as well as defined constants. These classes contain several utility member functions for common gateway interface.

Two generic data types **chchar** and **chstrarray** are also typedefed in the header file **cgi.h** as follows.

```
typedef char   chchar;
typedef char** chstrarray;
```

When the Unicode is used, this two generic data types **chchar** and **chstrarray** are typedefed in the header file **wcgi.h** as follows.

```
typedef wchar_t   chchar;
typedef wchar_t** chstrarray;
```

4.1 CResponse Class

When a browser requests data from a web server, the server responds, either with a redirect message, the requested data, or an error. The **CResponse** class contains several utility functions for sending information to the client.

Public Data
None.

Differences Between Ch-CGI and Ch-ASP
Member functions **CResponse**::**begin**(), **CResponse**::**end**(), and **CResponse**::**title**() are available in Ch-CGI only. They are not valid in Ch-ASP.

Note :
1. **CResponse**::**begin**() is mandatory in Ch-CGI for programs to run successfully.

2. The content type of the output of Ch-CGI is text/html by default.
3. The output is not buffered in Ch-CGI by default.

Public Member Functions

Function	Description
addCookie()	adds a specified cookie with attributes.
addHeader()	adds an HTTP header to the HTTP response.
begin()	begins to send output. **Mandatory in CGI**.
end()	ends standard output. **For CGI only**.
exit()	causes the server to stop processing a script and return.
flush()	sends buffered HTML output immediately.
getBuffer()	retrieves the value of the **Buffer** property.
getCacheControl()	retrieves the value of the **CacheControl** property.
getCharSet()	retrieves the value of the **CharSet** property.
getContentType()	retrieves the value of the **ContentType** property.
getExpires()	retrieves the value of the **Expires** property.
getExpiresAbsolute()	retrieves the value of the **ExpiresAbsolute** property.
getStatus()	retrieves the value of the **Status** property.
PICS()	adds a value to the PICS label field of the header.
redirect()	causes the browser to attempt to connect to a different URL.
setBuffer()	sets the value of the **Buffer** property.
setCacheControl()	sets the value of the **CacheControl** property.
setCharSet()	sets the value of the **CharSet** property.
setContentType()	sets the value of the **ContentType** property.
setExpires()	sets the value of the **Expires** property.
setExpiresAbsolute()	sets the value of the **ExpiresAbsolute** property.
setStatus()	sets the value of the **Status** property.
title()	sets the title of an HTML page. **For CGI only**.

CResponse::addCookie

Synopsis
int addCookie(**CCookie *** *cookie*);

Purpose
Add a specified cookie with attributes, such as MaxAge, Path, Domain and Secure, to a client.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
cookie A pointer to an object of CCookie which contains the information of the cookie to be added.

Description
The function **addCookie**() adds a specified cookie with attributes such as MaxAge, Path, Domain and Secure to a client. The **CRequest**::**getCookie**() function retrieve a cookie by name, while the **CRequest**::**getCookies**() function retrieves all of cookies.

Note: Properties of Name, Value, MaxAge, Path, Domain and Secure are supported by cookies which comply with Netscape Cookie specifications version 0 and RFC 2965 new version 1. Properties of Discard, Comment, CommentURL, portList and version are supported only by cookies which comply with RFC 2965 new version 1. Because there is no browser to support the cookie complying with RFC 2965 new version 1 as yet, these properties are not recommended to use.

Differences Between Ch-CGI and Ch-ASP
In Ch-ASP, **addCookie**() can add a cookie with properties of only Name, Value, Path, Domain, MaxAge and Secure.

In Ch-CGI, it can add a cookie with all properties of Name, Value, Path, Domain, MaxAge, Secure, Discard, Version, Comment and CommentURL and portList. Member function **addCookie**() shall be called before member function **begin**() is called.

Example 1

The program below adds cookies to the client.

```
#!/bin/ch
#include <cgi.h>

int main() {
  class CCookie cookie1, cookie2;
  class CResponse Response;
  class CRequest Request;

  Response.setContentType("text/html");

  cookie1.setName("name1");
  cookie1.setValue("value1");
  cookie1.setDomain("iel.ucdavis.edu");
```

```
  cookie1.setMaxAge(3600000);
  cookie1.setPath("/foo");
  cookie1.setSecure(true);
  Response.addCookie(&cookie1);

  cookie2.setName("name2");
  cookie2.setValue("value2");
  cookie2.setPath("/");
  cookie2.setComment("This cookie is for test");
  Response.addCookie(&cookie2);

  Response.begin();
  Response.title("CGI Cookie results");
  printf("<H1>CGI Cookie test script reports:</H1>\n");

  printf("two cookies have been sent to the client<br>\n");
  Response.end();
}
```

Example 2

The program below retrieves cookies which are added in Example 1.

```
#!/bin/ch
#include <cgi.h>

int main() {
  int i, num;
  class CCookie  *cookie;
  class CResponse Response;
  class CRequest Request;
  int *portlist, portnum, j;

  Response.begin();
  num = Request.getCookies(&cookie);
  if(num == 0) {
    printf("No cookie has been retrieved<p>\n");
    exit(0);
  }
  else if(num < 0) {
    printf("Error: in Request.getCookies() <p>\n");
    exit(0);
  }
  printf("The following %d Cookies are retrieved<p>\n",num);
  printf("<ul>\n");
  for(i=0; i < num; i++) {
    printf("<li> <code>%s = ",cookie[i].getName());
    printf("%s; ",cookie[i].getValue());
    printf("</code>\n");
  }
  printf("</ul>\n");
  Response.end();
}
```

This example will display:

```
The following 2 Cookies are retrieved
name1 = value1;
name2 = value2;
```

See Also
CRequest::**getCookies**(), **CRequest**::**getCookie**(), **CCookie**.

CResponse::addHeader

Synopsis
int addHeader(**chchar *** *headerName*, **chchar *** *headerValue*);

Purpose
Add an HTTP header to the response.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
headerName A string containing the name of the HTTP header.
headerValue A string containing the value of the HTTP header.

Description
The function **addHeader**() adds an HTTP header to the HTTP response. It always adds a new HTTP header to the response. It will not replace an existing header of the same name. Once a header has been added, it cannot be removed.

Example

```
char *headerName = "CHHEADER";
char *headerValue = "CHHEADERVAL";
Response.addHeader(headerName, headerValue);
```

This example will add the following header to the client:

```
CHHEADER:CHHEADERVAL
```

See Also
None.

CResponse::begin

Synopsis
int begin();

Purpose
Begins to send output. It is mandatory in Ch-CGI and for Ch-CGI only.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
None.

Description
The function **begin**() starts processing all headers set in **CResponse::set*()** and **CResponse::add*()**. It is mandatory in **Ch-CGI**. Any **CResponse::set*()** and **CResponse::add*()** member functions including **CResponse**::**setContentType**(), **CResponse**::**addCookie**(), and **CResponse**::**addHeader**() should be called before this function is called.

Differences Between Ch-CGI and Ch-ASP
This member function is mandatory in Ch-CGI. It is not valid in Ch-ASP.

Example
See **CRequest**::**getFormNameValue**().

See Also
CResponse::**end**(), **CResponse**::**title**().

CResponse::end

Synopsis
void end();

Purpose
End the standard output.

Return Value
None.

Parameters
None.

Description
The function **end**() will flush the buffer if the the **Buffering** is true and print out `</body>` and `</html>` tags to end an HTML page if its content type is text/html.

Differences Between Ch-CGI and Ch-ASP
This member function works in Ch-CGI only. It is not valid in Ch-ASP.

Example
See **CRequest**::**getFormNameValue**().

See Also
CResponse::**begin**(), **CResponse**::**title**().

CResponse::exit

Synopsis
void exit();

Purpose
Cause the server to stop processing a script and return.

Return Value
None.

Parameters
None.

Description
The function **exit**() causes the server to stop processing a script and return the current response. When this function is called, the remaining contents of the file are not processed, and the buffer are flushed if the **Buffering** is true.

Example

```
printf("Output before Response.exit()\n");
Response.exit();
printf("Output after Response.exit()\n");
```

The example above will only print out the string of

```
Output before Response.exit()
```

to the client, and then exit.

See Also
CResponse::**getBuffer**(), **CResponse**::**setBuffer**(), **CResponse**::**flush**().

CResponse::flush

Synopsis
int flush();

Purpose
Send buffered output immediately.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
None.

Description
The function **flush**() sends buffered output immediately. This function will cause a run-time error or be ignored if the **Buffer** property has not been set to **TRUE**.

In Ch-CGI, the output is not buffered by default. In Ch-ASP, the output is buffered by default.

In Ch-ASP If **Keep-Alives** is set in a web server, the server will maintain **Keep-Alive** requests made by the client, unless the **CResponse::flush()** is called.

HTTP **Keep-Alives** are an optimizing feature of servers and browsers; an HTTP **Keep-Alive** maintains a client connection after the initial request is satisfied. HTTP **Keep-Alives** are part of the HTTP version 1.1 specification.

When user set the **Buffer** property to **TRUE** in a script and do not call the **CResponse::flush()** method in the same script, the server will maintain **Keep-Alive** requests made by the client. The benefit of writing scripts in this manner is that server performance is improved because the server does not have to create a new connection for each client request (assuming that the server, client, and any proxies all support bf Keep-Alive requests).

However, a potential drawback to this approach is that buffering prevents any of the response from being displayed to the user until the server has finished all script processing for the current .asp file. For long involved scripts, the user might be forced to wait a considerable amount of time before the script is processed.

If this function is called, the server does not honor Keep-Alive requests for that page.

Example

```
bool buffer_cur;
buffer_cur = Response.getBuffer();
if(buffer_cur)
  printf("The current buffering is ture\n");
else {
  printf("The current buffering is false and will be set to ture\n");
  Response.setBuffer(ture);
}
printf("Output before Response.flush()\n");
Response.flush();
```

The example above will print out string of

```
Output before Response.flush()
```

to the client.
See Also
CResponse::**getBuffer**(), **CResponse**::**setBuffer**().

CResponse::getBuffer

Synopsis
bool getBuffer();

Purpose
Retrieve the current value of the **Buffer** property.

Return Value
A boolean data type. If page output is buffered, true is returned. Otherwise, false is returned.

Parameters
None.

Description
The function **getBuffer**() retrieves the current value of the **Buffer** property of the object. When page output is buffered, the server does not send a response to the client until all of the server scripts on the current page have been processed, or until the **CResponse::flush()**, **CResponse::end()** or **CResponse::exit()** function has been called.

The function of **CResponse::setBuffer** can set the current value of the **Buffer** prope rty of the object.

The **Buffer** property cannot be set after the server has sent output to the client. For this reason, the call to **CResponse::setBuffer** should be done before the **CResponse::begin()** function is invoked.

Example
See **CResponse**::**flush**().

See Also
CResponse::**setBuffer**(), **CResponse**::**flush**().

CResponse::getCacheControl

Synopsis
chchar * getCacheControl();

Purpose
Retrieve a value of the **CacheControl** property.

Return Value
Upon successful completion, a string which contains the CacheControl value is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getCacheControl**() retrieves a value of the **CacheControl** property. The **CResponse**::**setCacheControl**function can be used to override the default value which is Private. By setting the value to Public, proxy servers will be able to cache output from pages; no-cache, the Response message MUST NOT be cached anywhere.

Example
See **CResponse**::**setCacheControl**().

See Also
CResponse::**setCacheControl**().

CResponse::getCharSet

Synopsis
chchar * getCharSet();

Purpose
Retrieve a character set to append to the content type header.

Return Value
Upon successful completion, a string which contains a character set is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getCharSet**() retrieves a character set to append to the content type header. The **CResponse**::**setCharSet** function can be used to set the character set when displaying the current object.

Example
See **CResponse**::**setCharSet**().

See Also
CResponse::**setCharSet**().

CResponse::getContentType

Synopsis
chchar * getContentType();

Purpose
Retrieve the current value of the **ContentType** property.

Return Value
Upon successful completion, a string which contains the ContentType value is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getContentType**() retrieves the current value of the **ContentType** property of the object and the **CResponse::setContentType** function can set the content type. The content type of the output of **Ch-CGI** and **Ch-ASP** is text/html by default.

Example
See **CResponse**::**setContentType**().

See Also
CResponse::**setContentType**().

CResponse::getExpires

Synopsis
int getExpires();

Purpose
Retrieve the current value of the **Expires** property.

Return Value
Upon successful completion, an integer that indicates the minutes of expires is returned. Otherwise, a negative value is returned.

Parameters
None.

Description
The function **getExpires**() retrieves the current value of the **Expires** property of the object. If the user returns to the same page before it expires, the cached version is displayed. If this property is set more than once on a page, the shortest time is used.

If the property is never set before it is called, **INT_MAX** will be returned. In this case, the return value does not make sense.

The **CResponse::setExpires** function can set a new value to the **Expires** property.

Example
See **CResponse**::**setExpires**().

See Also
CResponse::**setExpires**().

CResponse::getExpiresAbsolute

Synopsis
chchar * getExpiresAbsolute();

Purpose
Retrieve the current value of the **ExpiresAbsolute** property.

Return Value

Upon successful completion, a string which contains the value of the **ExpiresAbsolute** property is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getExpiresAbsolute**() retrieves the current value of the **ExpiresAbsolute** property of the object. If the user returns to the same page before the set date and time, the cached version is displayed. If this property is set more than once on a page, the earliest expiration date or time is used. If the expiration date and time is not set before this function is called, NULL is returned.

The date string is formatted as:

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

This is based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is GMT(Greenwich Mean Time) and the separators between the elements of the date must be dashes.

The **CReponse::setExpiresAbsolute** function can be used to set this property.

Example
See **CResponse**::**setExpiresAbsolute**().

See Also
CResponse::**setExpiresAbsolute**().

CResponse::getStatus

Synopsis
chchar * getStatus();

Purpose
Retrieve the current value of the **Status** property.

Return Value
Upon successful completion, a pointer that points to a string which contains the value of the **Status** property is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getStatus**() retrieves the current value of the **Status** property of the object. The **CResponse::setStatus** function can be used to modify the status line. Status values are defined in the HTTP1.1 RFC 2068.

Example
See **CResponse**::**setStatus**().

See Also
CResponse::**setStatus**().

CResponse::PICS

Synopsis
int PICS(**chchar *** *headerValue*);

Purpose
Add a value to the **PICS** label field of the header.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
headerValue A string containing the new **PICS** value.

Description
The function **PICS**() adds a value to the **PICS** label field of the header. It inserts any string in the header, whether or not it represents a valid **PICS** label.

If a single page includes multiple tags containing **CResponse::PICS** function, each instance will replace the **PICS** label set by the previous one. As a result, the **PICS** label will be set to the value specified by the last instance of **CResponse::PICS** in the page.

Because **PICS** labels contain quotes, the author must add a backslash before each quote.

For more details on the **PICS** standard, see http://www.w3.org/Pics/.

Example

```
Response.PICS("(PICS-1.1 <http://www.rsac.org/ratingv01.html> labels on
\"1997.01.05T08:15-0500\" until \"1999.12.31T23:59-0000\" ratings (v 0 s
0 l 0 n 0))");
```

This example will add the following header to the client:

```
PICS-label:(PICS-1.1 <http://www.rsac.org/ratingv01.html> labels on
"1997.01.05T08:15-0500" until "1999.12.31T23:59-0000" ratings (v 0 s
0 l 0 n 0))
```

See Also
None.

CResponse::redirect

Synopsis
int redirect(**chchar *** *URL*);

Purpose
Stop the server from processing the current script and then causes the browser to attempt to connect to a different URL.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
URL A string containing the URL.

Description
The function **redirect**() stops the server from processing the current script and then causes the browser to attempt to connect to a different URL.

If you have set any response body content in the page, it will be ignored. However, this function does send to the client other HTTP headers set by this page. An automatic response body containing the redirect URL as a link is generated. This function sends the following explicit header,

```
HTTP 1.0 302 Object Moved
Location: URL
```

where URL is the value passed to the function.
If buffering is set to false and your component attempts to call this function after any body has been sent to the client, the server will generate an error.

Example

```
char *URL = "iel.ucdavis.edu"
Response.redirect(URL);
```

The example above will redirect user to primary web site of IEL.

See Also
CResponse::**getStatus**().

CResponse::setBuffer

Synopsis
int setBuffer(**bool** *buffering*);

Purpose
Set the value of the **Buffer** property.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters

buffering A boolean value that contains the new Buffer value.

Description
The function **setBuffer**() sets the current value of the **Buffer** property of the object. When page output is buffered, the server does not send a response to the client until all of the server scripts on the current page have been processed, or until the **CResponse::flush**, **CResponse::end** or **CResponse::exit** function has been called.

The **Buffer** property cannot be set after the server has sent output to the client. For this reason, the call to **CResponse::setBuffer** should be the first line of the script file.

The function of **CResponse::getBuffer** can retrieve the current value of the **Buffer** property of the object.

Note:
Under **Ch-CGI** debug mode with the first line of a **Ch-CGI** code as `#!/bin/ch -g`, this setting has no effect.
Differences Between Ch-CGI and Ch-ASP
In Ch-CGI, the output is not buffered by default.
In Ch-ASP, the output is buffered by default.

Example
See **CResponse**::**flush**().

See Also
CResponse::**getBuffer**(), **CResponse**::**flush**(), **CResponse**::**exit**().

CResponse::setCacheControl

Synopsis
int setCacheControl(**chchar *** *cacheControl*);

Purpose
Set the value of the **CacheControl** property.

Return Value
Upon successful completion, zero is returned. Otherwise, a value of non-zero is returned.

Parameters
cacheControl A string containing the new CacheControl value which could be Private(by default), Public or no-cache. no-cache is only supported by HTTP/1.1 protocal.

Description
The function **setCacheControl**() sets the value of the **CacheControl** property of the object. This function can be used to override the default value which is Private. By setting the value to Public, proxy servers will be able to cache output from pages; no-cache, the Response message MUST NOT be cached anywhere. The **CResponse::getCacheControl** function can retrieve a value for the **CacheControl** property.

Differences Between Ch-CGI and Ch-ASP
In Ch-ASP, **setCacheControl**() does not support the parameter of no-cache.

Example

```
char *cacheControl = "public";
Response.setCacheControl(cacheControl);
printf("The current value of cache control is %s\n",
       Response.getCacheControl());
```

The example above will set the current value of **CacheControl** property to ”public” and then print out:

```
The current value of cache control is public
```

See Also
CResponse::**getCacheControl**().

CResponse::setCharSet

Synopsis
int setCharSet(**chchar *** *charSet*);

Purpose
Set the value of the **CharSet** property.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
charSet A string containing the new CharSet value.

Description
The function **setCharSet**() sets the value of the **CharSet** property of the object. The **CResponse**::**getCharSet**() function can retrieve the character set of the current **HTML** page.

Example

```
char *charSet = "ISO-LATIN-1";
Response.setCharSet(charSet);
printf("The current value of character set is %s\n",
       Response.getCharSet());
```

The example above will set current value of **CharSet** property to ”ISO-LATIN-1” and then print out:

```
The current value of character set is ISO-LATIN-1
```

See Also
CResponse::**getCharSet**().

CResponse::setContentType

Synopsis
int setContentType(**chchar *** *type*);

Purpose
Set the value of the **ContentType** property.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
type A string containing the new ContentType value. Valid ContentType include `text/plain`, `text/html`, `image/gif`, etc.

Description
The function **setContentType**() sets the value of the **ContentType** property of the object. The content type of the output of Ch-CGI and Ch-ASP is text/html by default. The **CResponse**::**getContentType**function can be used to get the **ContentType** property of the current object.

If the content type is plain text set by the member function call of `Response.setContentType("text/plain")`, the member function `Response.title(title)` which generates a title in HTML format shall not be called.

In Ch-CGI, this function and other **CResponse::set*()** member functions, should be called before **CResponse**::**begin**() is called.

Example

```
char *type = "image/JPEG";
Response.setContentType(type);
printf("The current content type is %s\n", Response.getContentType());
```

The example above will set the current content type to ”image/JPEG” and then print out:

```
The current content type is "image/JPEG"
```

See Also
CResponse::**getContentType**().

CResponse::setExpires

Synopsis
int setExpires(**int** *expiresMinutes*);

Purpose
Set the current value of the **Expires** property.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
expiresMinutes An integer that contains the minutes of new Expires value.

Description
The function **setExpires**() sets the current value of the **Expires** property of the object. If the user returns to the same page before it expires, the cached version is displayed. If this property is set more than once on a page, the shortest time is used.

CResponse::**getExpires**() gets the current value of the **Expires** property of the object. If the property is never set before it is called, **INT_MAX** will be returned by the function of getExpires(). In this case, the return value does not make sense.

Example

```
int expiresMinutes = 10;
Response.setExpires(expiresMinutes);
printf("The current value of expires is %d minutes\n",
       Response.getExpires());
```

The example above will set the current value of **Expires** property to 10 minutes and then print out:

```
The current value of expires is 10 minutes
```

See Also
CResponse::**getExpires**().

CResponse::setExpiresAbsolute

Synopsis
int setExpiresAbsolute(**chchar *** *expires*);

Purpose
Set the value of the ExpiresAbsolute property.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
expires A string containing the new ExpiresAbsolute value.

Description
The function **setExpiresAbsolute**() sets the value of the ExpiresAbsolute property of the object. If the user returns to the same page before the set date and time, the cached version is displayed. If this function is called more than once on a page, the earliest expiration date or time is used.

The date string is formatted as:

```
Wdy, DD-Mon-YYYY HH:MM:SS GMT
```

This is based on RFC 822, RFC 850, RFC 1036, and RFC 1123, with the variations that the only legal time zone is GMT(Greenwich Mean Time) and the separators between the elements of the date must be dashes.

The **CReponse::getExpiresAbsolute()** function can be used to get the date and time. If the expiration date and time is not set before this function is called, NULL is returned.

Example

```
char *expires = "Monday, 17-Dec-2001 14:02:40 GMT";
Response.setExpiresAbsolute(expires);
printf("The current value of expires is \n",
       Response.getExpiresAbsolute());
```

The example above will set the current value of **ExpiresAbsolute** property to ”Monday, 17-Dec-2001 14:02:40 GMT” and then print out:

```
The current value of expires is Monday, 17-Dec-2001 14:02:40 GMT
```

See Also
CResponse::**getExpiresAbsolute**().

CResponse::setStatus

Synopsis
int setStatus(**chchar *** *status*);

Purpose
Set the value of the Status property.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
status A string containing the new Status value.

Description
The function **setStatus**() sets the value of the Status property of the object. The **CResponse**::**getStatus**() function can be used to obtain the status line returned by the server.

Status values are defined in the HTTP1.1 RFC 2068.

Example

```
char *status = "401 Unauthorized";
Response.setStatus(status);
printf("The current status line is %s\n", Response.getStatus());
```

The example above will set current status line to ”401 Unauthorized” and then print out:

```
The current status line is 401 Unauthorized
```

See Also
CResponse::**getStatus**().

CResponse::title

Synopsis
void title(**chchar *** *title*);

Purpose
Set the title of an HTML page.

Return Value
None.

Parameters
title A string containing title of the HTML page.

Description
The function **title**() sets the title by adding `<head>` and `<title>` tags into an HTML page. If content type is `text/html`, tags of `<html>` and `<body bgcolor="#FFFFFF">` are added by this function as well, the **CResponse**::**end**() function will add `</body>` and `</html>` tags at the end of this HTML page correspondingly.
It is equivalent to the code below:

```
void CResponse::title(char* titleName)
{
   printf("<html>\n");
   if (titleName != NULL)
       printf("<head> <title> %s </title></head>\n", titleName);
    printf("<body bgcolor=\"#FFFFFF\">\n");
}
```

Differences Between Ch-CGI and Ch-ASP
This member function works in Ch-CGI only. It is not valid in Ch-ASP.

Example
See **CRequest**::**getFormNameValue**().

See Also
CResponse::**begin**(), **CResponse**::**end**().

4.2 CRequest Class

The **CRequest** class contains several utility functions for receiving the content from a browser. One of the most important applications of **CRequest**is handling fill-out forms. The program located on a host machine's web server can accept user's input through a fill-out form and generate web pages dynamically.

Public Data
None.

Differences Between Ch-CGI and Ch-ASP
By default, there is no difference in **CRequest** between Ch-CGI and Ch-ASP.

Public member functions.

Function	Description
binaryRead()	retrieves the bytes that were read by an HTTP Post and place it into a buffer.
getCookie()	retrieves a cookie.
getCookies()	retrieves all cookies.
getForm()	retrieves a value of the specified name which was read by POST or GET method.
getForms()	retrieves all values of a specified name which were read by POST or GET method.
getFormNameValue()	retrieves all pairs of name and value that were read by POST or GET method.
getServerVariable()	retrieves the value of a specified ServerVariable.
getTotalBytes()	retrieves the size of the current request in bytes.

CRequest::binaryRead

Synopsis
char * binaryRead(**size_t ***count*);

Purpose
Retrieve the bytes that were read by an HTTP Post and places it into a buffer.

Return Value
Upon successful completion, a pointer that points to a buffer which contains the retrieved bytes is returned. Otherwise, NULL is returned.

Parameters

count A pointer that points to a value of type size_t. Before execution, this value specifies how many bytes to read from the client. After this function returns, count will contain the number of bytes successfully read from the client. The total number of bytes that will actually be read is less than or equal to the value returned by the **CRequest::getTotalBytes** function.

Description
The function **binaryRead**() retrieves the bytes that were read by an HTTP Post and places it into a buffer. This function is used to read the raw data sent by the client as part of a POST request and it is used for low-level access to this data, as opposed to, for example, using the **CRequest::getForm** function view form data sent in a POST request.

Once this funtion has been called, any execution of **CRequest::getForm**, **CRequest::getForms** or **CRequest::getFormNameValue** function will cause an error. Conversely, once these **CRequest::getForm*** functions have been called, calling this function will cause an error.

Example

```
size_t count;
char * buffer;
count = Request.getTotalBytes();
printf("The size of the current Request in bytes is : %d\n", count);
buffer = Request.binaryRead(&count);
printf("The raw data is : %s\n", buffer);
printf("The actually read bytes is : %d\n", count);
```

The example above will print out the size of the current request , the raw data which is actually read and its size.

See Also
CRequest::**getTotalBytes**(), **CRequest**::**getForm**(), **CRequest**::**getForms**(), **CRequest**::**getFormNameValue**().

CRequest::getCookie

Synopsis
chchar * getCookie(**chchar*** *cookieName*);

Purpose
Retrieve the value of a cookie by its name.

Return Value
Upon successful completion, a string which contains the value of a cookie is returned. Otherwise, NULL is returned.

Parameters
cookieName A string which contains the name of a cookie.

Description
The function **getCookie**() retrieves the value of a cookie by its name. The **CRequest**::**getCookies**() function can get all cookies. The **CResponse**::**addCookie**() function adds a cookie to a client.

Differences Between Ch-CGI and Ch-ASP
In Ch-ASP, the cookies added by addCookie() can be retrieved by getCookie() or getCookes() in the same program. In Ch-CGI, the cookies added by addCookie() cannot be retrieved by getCookie() or getCookies() in the same program.

Example
Program 1:

```
class CCookie cookie1;
char *cookieName = "name1";
char *cookieValue = "value1";
cookie1.setName(cookieName);
cookie1.setValue(cookieValue);
Response.addCookie(&cookie1);
```

Program 2:

```
printf("The value of %s is %s \n", cookieName,
       Request.getCookie(cookieName));
```

The program 2 of this example will print out :

```
The value of name1 is value1
```

See Also
CResponse::**addCookie**(), **CRequest**:**getCookies**(), **CCookie**.

CRequest::getCookies

Synopsis
int getCookies(**CCookie**** *cookies*);

Purpose
Retrieve all of cookies.

Return Value
Upon successful completion, the number of cookies actually retrieved is returned. Otherwise, a negative value is returned.

Parameters
cookies A pointer to an array of CCookie objects which contains the retrieved cookies.

Description
The function **getCookies**() retrieves all of cookies. The **CRequest**::**getCookie**() function retrieves the value of a cookie by its name. The **CResponse**::**addCookie**() function adds a cookie to a client.

Differences Between Ch-CGI and Ch-ASP
In Ch-ASP, the cookies added by addCookie() can be retrieved by getCookie() or getCookes() in the same program. In Ch-CGI, the cookies added by addCookie() cannot be retrieved by getCookie() or getCookies() in the same program in Ch-CGI.

Example
See **CResponse**::**addCookie**().

See Also
CResponse::**addCookie**(), **CRequest**::**getCookie**(), **CCookie**.

CRequest::getForm

Synopsis
chchar * getForm(**chchar *** *name*);

Purpose
Retrieve a value of the specified name which was read by POST or GET method.

Return Value
Upon successful completion, a value for the specified name read by POST or GET method is returned. Otherwise, NULL is returned.

Parameters
name A string which contains the name of the specified item.

Description
The function **getForm**() retrieves a value of the specified name which was read by POST or GET method. If there are multiple values for the specified name, the first one is returned. The **CRequest::getForms** function retrieves all values of items with the specified name, and the **CRequest::getFormNameValue** function retrieves all pairs of name and value.

Example

```
char *name = "favorite";
char *value;
value = Request.getForm(name);
if(value == NULL)
    printf("No item named %s has been submitted\n", name);
else
    printf("The value of items named %s is %s\n", name, value);
```

The example above will retrieve all unparsed values of items with name of ”favorite” which is read by POST or GET methods, then print these values out as a single string.

See Also
CRequest::**getForms**(), **CRequest**::**getFormNameValue**().

CRequest::getForms

Synopsis
int getForms(**chchar *** *name*, **chstrarray &** *values*);

Purpose
Retrieve all values of the specified name which were read by POST or GET method.

Return Value
Upon successful completion, number of repeated values of the specified name is returned. Otherwise, zero is returned.

Parameters
name A string which contains the name of the specified item.
value A reference of an array which contains all values of the specified name.

Description
The function **getForms**() retrieves all values of the specified name which were read by POST or GET method. The **CRequest::getForm** function retrieves the first value of the specified name, and the **CRequest::getFormNameValue** function retrieves all pairs of name and value.

Example

```
int num;
char *name = "favorite";
chstrarray value;
num = Request.getForms(name, value);
if(num == 0)
  printf("No item named %s has been submitted\n", name);
else if (num < 0)
  printf("Error: in Request.getForms()\n");
else {
  printf("The following %d values are submitted with the name of %s\n",
         num, name);
  printf("<ul>\n");
```

```
  for(i=0; i < num; i++)
    if(value[i])
      printf("<li> <code> %s </code>\n",value[i]);
  printf("</ul>\n");
}
```

The example above will retrieve all values of items with name of ”favorite” which is read by POST or GET, then print these values out seperately.

See Also
CRequest::**getForm**(), **CRequest**::**getFormNameValue**().

CRequest::getFormNameValue

Synopsis
int getFormNameValue(**chstrarray &** *names*, **chstrarray &** *values*);

Purpose
Retrieve all name/value pairs that were read by POST or GET method.

Return Value
Upon successful completion, the number of name/value pairs is returned. Otherwise, -1 is returned.

Parameters
names A reference of an array that contains names of the pairs.
values A reference of an array that contains values of the pairs.

Description
The function **getFormNameValue**() retrieves all pairs of name and value that were read by POST or GET method. The name and value in one pair will be saved in corresponding positions of the retrieved arrays. **CRequest::getForms** function retrieves all values of items with the specified name, while **CRequest::getForm** function retrieves the first value.

Characters like &, %, and $ typed into the text entry field in a fill-out form are automatically escaped into hex form — a percent sign followed by a two-digit hex value corresponding to the ASCII value of the character when the query is constructed in a Web browser. For example, string ”&%$” becomes ”%26%25%24”.

Example

The program below can be used to obtain all names and values from a fill-out form, and print them out in content type of text in HTML file format.

```
#!/bin/ch
#include <cgi.h>

int main() {
  int i, num;
  chstrarray name, value;
  class CResponse Response;
  class CRequest Request;
```

```
  Response.setContentType("text/html");
  Response.begin();
  Response.title("CGI FORM results");
  printf("<H1>CGI FORM test script reports:</H1>\n");

  num = Request.getFormNameValue(name, value);
  if(num == 0) {
    printf("No name/value has been submitted<p>\n");
    Response.exit();
  }
  else if(num < 0) {
    printf("Error: in Request.getFormNameValue() <p>\n");
    Response.exit();
  }
  printf("The following %d name/value pairs are submitted<p>\n",num);
  printf("<ul>\n");
  for(i=0; i < num; i++) {
    printf("<li> <code>%s = ",name[i]);
    if(value[i])
      printf("%s",value[i]);
    printf("</code>\n");
  }
  printf("</ul>\n");
  Response.end();
}
```

See Also
CRequest::**getForm**(), **CRequest**::**getForms**().

CRequest::getServerVariable

Synopsis
chchar * getServerVariable(**chchar *** *variableName*);

Purpose
Retrieve the value of a specified server variable.

Return Value
Upon successful completion, a string which contains the value of a specified server variable. Otherwise, NULL is returned.

Parameters
variableName A string which contains the name of a specified server variable.

Description
The function **getServerVariable**() retrieves a specified ServerVariable value. If a client sends a header other than those specified in the table below, the value of that header can be retrieved by prefixing the header name with HTTP_ in the call to this function. For example, if the client sent the header

```
SomeNewHeader:SomeNewValue
```

SomeNewValue can be retrieved by using the following syntax:

```
headerValue = Request.getServerVariable("HTTP_SomeNewHeader");
```

Server Variables

Variable	Description
ALL_HTTP	All HTTP headers sent by the client.
ALL_RAW	All headers in raw form.
APPL_MD_PATH	The metabase path for the Application for the ISAPI DLL.
APPL_PHYSICAL_PATH	The physical path corresponding to the metabase path.
AUTH_PASSWORD	The value entered in the client's authentication dialog.
AUTH_TYPE	The authentication method used by server.
AUTH_USER	Raw authenticated user name.
CERT_COOKIE	Unique ID for client certificate.
CERT_FLAGS	Determine if the client certificate is present.
CERT_ISSUER	Issuer field of the client certificate.
CERT_KEYSIZE	Number of bits in Secure Sockets Layer connection key size.
CERT_SECRETKEYSIZE	Number of bits in server certificate private key.
CERT_SERIALNUMBER	Serial number field of the client certificate.
CERT_SERVER_ISSUER	Issuer field of the server certificate.
CERT_SERVER_SUBJECT	Subject field of the server certificate.
CERT_SUBJECT	Subject field of the client certificate.
CONTENT_LENGTH	The length of the content as given by the client.
CONTENT_TYPE	The data type of the content.
GATEWAY_INTERFACE	The revision of the CGI specification used by the server.
`HTTP_<HeaderName>`	The value stored in the header HeaderName.
HTTP_ACCEPT	The value of the Accept header.
HTTP_ACCEPT_LANGUAGE	A string describing the language to use for displaying content.
HTTP_USER_AGENT	A string describing the browser that sent the request.
HTTP_COOKIE	The cookie string that was included with the request.
HTTP_REFERER	A string containing the URL of the page that referred the request to the current page, but does not include redirect requests.

Server Variables (Contd.)

Variable	Description
HTTPS	ON if the request came in through secure channel (SSL) or OFF if the request is for a non-secure channel.
HTTPS_KEYSIZE	Number of bits in Secure Sockets Layer connection key size.
HTTPS_SECRETKEYSIZE	Number of bits in server certificate private key.
HTTPS_SERVER_ISSUER	Issuer field of the server certificate.
HTTPS_SERVER_SUBJECT	Subject field of the server certificate.
INSTANCE_ID	The ID for the IIS instance in textual format.
INSTANCE_META_PATH	The metabase path for the instance of response.
LOCAL_ADDR	The Server Address on which the request came in.
LOGON_USER	The Windows account that the user is logged into.
PATH_INFO	Extra path information as given by the client.
PATH_TRANSLATED	A translated version of PATH_INFO.
QUERY_STRING	Query information stored in the string following the question mark (?) in the HTTP request.
REMOTE_ADDR	The IP address of the remote host making the request.
REMOTE_HOST	The name of the host making the request.
REMOTE_USER	Unmapped user-name string sent in by the user.
REQUEST_METHOD	The method used to make the request.
SCRIPT_NAME	A virtual path to the script being executed.
SERVER_NAME	The server's host name.
SERVER_PORT	The port number to which the request was sent.
SERVER_PORT_SECURE	A string that contains either 0 or 1.
SERVER_PROTOCOL	The name and revision of the request information protocol.
SERVER_SOFTWARE	The name and version of the server software.
URL	Gives the base portion of the URL.

Example

```
char *variableName = "SERVER_NAME";
char *variableValue;
variableValue = Request.getServerVariable(variableName);
printf("The hostname of the web server is : %s\n", variableValue);
```

The example above will print out the hostname of the web server.

See Also
None.

CRequest::getTotalBytes

Synopsis
size_t getTotalBytes();

Purpose
Retrieve the size of the current request in bytes.

Return Value
Upon successful completion, a value of type of size_t that contains the size of the current request in bytes is returned. Otherwise, -1 is returned.

Parameters
None.

Description
The function **getTotalBytes**() retrieves the size of the current request in bytes.

Example
See **CRequest**::**binaryRead**().

See Also
CRequest::**binaryRead**().

4.3 CServer Class

The **CServer** class contains several utility functions for high-level access to the web server.

Public Data
None.

Differences Between Ch-CGI and Ch-ASP
By default, there is no difference in **CServer** between Ch-CGI and Ch-ASP.

Public member functions.

Function	Description
HTMLEncode()	applies HTML encoding to the specified string.
URLEncode()	applies URL encoding rules, including escape characters, to the specified string.
mapPath()	maps the specified relative or virtual path to the corresponding physical directory on the server.

CServer::HTMLEncode

Synopsis
chchar * HTMLEncode(**chchar *** *in*);

Purpose
Apply HTML encoding to the specified string.

Return Value
Upon successful completion, a string which contains the HTML encoded text is returned. Otherwise, NULL is returned.

Parameters
in A string containing the text to be HTML encoded.

Description
The function **HTMLEncode**() applies HTML encoding to the specified string. If a browser get a encoded text, it will display it in HTML format, rather than in plain text. For example, if the parameter contains a string with symbols of < >, the returned value would contain the HTML code for those characters as < >. A browser would display these two symbols as < >.

Example

```
char *in = "< >";
char *result;
result = Server.HTMLEncode(in);
printf("The string \"< >\" is encoded by %s\n", result);
```

The example above will print out:

```
The string "< >" is encoded by &lt; &gt;
```

to the client. But the user will actually see:

```
The string "< >" is encoded by < >
```

in the HTML page. .

See Also
CServer::**URLEncode**().

CServer::URLEncode

Synopsis
chchar * URLEncode(**chchar *** *in*);

Purpose
Apply URL encoding rules, including escape characters, to the specified string.

Return Value
Upon successful completion, a string which contains the URL encoded text is returned. Otherwise, NULL is returned.

Parameters
in A string containing the text to be URL encoded.

Description
The function **URLEncode**() applies URL encoding rules, including escape characters, to the specified string.

Example

```
char *in = "x*sin(x)";
char *result;
result = Server.URLEncode(in);
printf("The string \"x*sin(x)\" is encoded by %s\n", result);
```

The example above will print out:

```
The string "x*sin(x)" is encoded by x%2Asin%28x%29
```

to the client.

See Also
CServer::**HTMLEncode**().

CServer::mapPath

Synopsis
chchar * mapPath(**chchar *** *path*);

Purpose
Map the specified relative or virtual path to the corresponding physical directory on the server.

Return Value
Upon successful completion, a string which receives the physical path is returned. Otherwise, NULL is returned.

Parameters
path A string containing relative or virtual path.

Description
The function **mapPath**() maps the specified relative or virtual path to the corresponding physical directory on the server. This function does not check whether the path it returns is valid or exists on the server. Because it maps a path regardless of whether the specified directories currently exist, the user can use it to map a path to a physical directory structure, and then pass that path to a component that creates the specified directory or file on the server.

Example

```
char *path = ".";
char *result;
result = Server.mapPath(path);
printf("The virtual path of .\\ is mapped to : %s\n", result);
```

The example above will print out the current directory of the web server.

See Also
None.

4.4 CCookie Class

A cookie is a small amount of information sent by a web server to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management.

The **CCookie** class contains several utility functions for setting or getting the name and value of a cookie, as well as optional attributes. The **CCookie** class supports both the Version 0 (by Netscape) and Version 1 (by RFC 2965 which obsoletes RFC 2109). By default, Ch cookies uses version 0. Since RFC 2965 is released on October 2000, most browsers might not support RFC 2965, the users are encouraged to use Version 0 features.
Note: Properties of Name, Value, MaxAge, Path, Domain, Secure and Version are supported by cookies which comply with both of Netscape Cookie specifications version 0 and RFC 2965 new version 1. Properties of Discard, Comment, CommentURL, and portList are supported only by cookies which comply with RFC 2965 new version 1. Because there is no browser to support the cookie complying with RFC 2965 new version 1 as yet, these properties are not recommended to use.

Public Data
None.

Differences Between of Ch-CGI and Ch-ASP
By default, there is no difference in **CCookie** between Ch-CGI and Ch-ASP.

Public Member Functions

Function	Description
addPort()	adds a new port into the portlist of the cookie. For version 1 only.
getComment()	retrieves the **Comment** attribute of the cookie. For version 1 only.
getCommentURL()	retrieves the **CommentURL** attribute of the cookie. For version 1 only.
getDiscard()	retrieves the **Discard** attribute of the cookie . For version 1 only.
getDomain()	retrieves the **Domain** attribute of the cookie.
getMaxAge()	retrieves maximum age of the cookie.
getName()	retrieves the name of the cookie.
getPath()	retrieves the path on the server to which browser returns the cookie.
getPorts()	retrieves all ports in the portlist of the cookie. For version 1 only.
getSecure()	determines if the browser is sending the cookie only over a secure protocol.
getValue()	retrieves the value of the cookie.
getVersion()	retrieves the version of the protocol the cookie complies with.
setComment()	sets the **Comment** attribute of the cookie. For version 1 only.
setCommentURL()	sets the **CommentURL** attribute of the cookie. For version 1 only.
setDiscard()	sets the **Discard** attribute of the cookie. For version 1 only.
setDomain()	sets the **Domain** attribute of the cookie.
setMaxAge()	sets maximum age of the cookie.
setName()	sets the name of the cookie.
setPath()	sets the path on the server to which browser returns the cookie.
setSecure()	sets the **Secure** attribute of the cookie.
setValue()	sets the value of the cookie.
setVersion()	sets the version of the protocol the cookie complies with.

CCookie::addPort

Synopsis
int addPort(int *portNum*);

Purpose
Add a new port into the portlist of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
portNum An integer which indicates the new port to be added.

Description
The function **addPort**() adds a new port into the portlist of the cookie. This portlist contains some ports to which a cookie may be returned in a Cookie request header.

The **CCookie**::**getPorts**() function can be used to retrieve this portlist.

Example

The program below sets and gets properties of a cookie locally.

```
#!/bin/ch
#include <cgi.h>

int main() {
  class CCookie cookie1;
  class CResponse Response;
  int *portlist, portnum, j;

  Response.setContentType("text/html");
  Response.begin();
  Response.title("CGI Cookie results");
  printf("<H1>CGI Cookie test script reports:</H1>\n");

// set properties of cookie
  cookie1.setName("name1");
  cookie1.setValue("value1");
  cookie1.setVersion(1);
  cookie1.addPort(8080);
  cookie1.addPort(8081);
  cookie1.setComment("This cookie is for test");
  cookie1.setCommentURL("");
  cookie1.setDiscard(false);
  cookie1.setDomain("iel.ucdavis.edu");
  cookie1.setMaxAge(3600000);
  cookie1.setPath("/foo");
  cookie1.setSecure(true);

// get properties of cookie
  printf("<li> <code>%s = ",cookie1.getName());
```

```
  printf("%s; ",cookie1.getValue());
  printf("Version = %d; ",cookie1.getVersion());
  printf("Comment = %s; ",cookie1.getComment());
  printf("CommentURL = %s; ",cookie1.getCommentURL());
  printf("Discard = %d; ",cookie1.getDiscard());
  printf("Domain = %s; ",cookie1.getDomain());
  printf("MaxAge = %d; ",cookie1.getMaxAge());
  printf("Path = %s; ",cookie1.getPath());
  printf("Secure = %d; ",cookie1.getSecure());
  portnum = cookie1.getPorts(portlist);
  if(portnum > 0)
     for(j = 0; j < portnum; j++)
       printf("port[%d] = %d; ", j, portlist[j]);
  printf("</code>\n");
  printf("</ul>\n");
  Response.end();
}
```

See Also
CCookie::**getPorts**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getComment

Synopsis
chchar * getComment();

Purpose
Retrieve the **Comment** attribute describing the purpose of this cookie.

Return Value
Upon successful completion, a string which contains the value of the **Comment** attribute of the cookie is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getComment**() retrieves the **Comment** attribute describing the purpose of this cookie. Because cookies can be used to derive or store private information about a user, the value of the Comment attribute allows an origin server to document how it intends to use the cookie. The user can inspect the information to decide whether to initiate or continue a session with this cookie. Characters in value MUST be in UTF-8 encoding. [RFC2279]

Property of **Comment** are not supported by Netscape Version 0 cookies.

The **CCookie**::**setComment**() function can be used to set the **Comment** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also

CCookie::**setComment**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getCommentURL

Synopsis
chchar * getCommentURL();

Purpose
Retrieve the **CommentURL** attribute of the cookie.

Return Value
Upon successful completion, a string which contains the value of the **CommentURL** attribute of the cookie is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getCommentURL**() retrieves the **CommentURL** attribute of the cookie. Because cookies can be used to derive or store private information about a user, the **CommentURL** attribute allows an origin server to document how it intends to use the cookie. The user can inspect the information identified by the URL to decide whether to initiate or continue a session with this cookie.

The **CCookie**::**setCommentURL**() function can be used to set the **CommentURL** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setCommentURL**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getDiscard

Synopsis
bool getDiscard();

Purpose
Retrieve the **Discard** attribute of the cookie.

Return Value
A boolean value which contains the **Discard** attribute of the cookie is returned.

Parameters
None.

Description
The function **getDiscard**() retrieves the **Discard** attribute of the cookie. The Discard attribute instructs the

user agent to discard the cookie unconditionally when the user agent terminates.

The **CCookie**::**setDiscard**() function can be used to set the **Discard** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setDiscard**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getDomain

Synopsis
chchar * getDomain();

Purpose
Retrieve the **Domain** attribute of the cookie.

Return Value
Upon successful completion, a string which contains the value of the **Domain** attribute of the cookie is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getDomain**() retrieves the **Domain** attribute of the cookie. The value of the **Domain** attribute specifies the domain for which the cookie is valid. If an explicitly specified value does not start with a dot, the user agent supplies a leading dot.

The **CCookie**::**setDomain**() function can be used to set the **Domain** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setDomain**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getMaxAge

Synopsis
int getMaxAge();

Purpose
Retrieve the maximum age of the cookie.

Return Value

Upon successful completion, an integer which indicates the maximum age of the cookie in seconds is returned. Otherwise, a negative value is returned.

Parameters
None.

Description
The function **getMaxAge**() retrieves the maximum age of the cookie. The value of the maximum age is the lifetime of the cookie in seconds. It is a decimal non-negative integer. To handle cached cookies correctly, a client should calculate the age of the cookie according to the age calculation rules in the HTTP/1.1 specification [RFC2616]. When the age is greater than delta-seconds seconds, the client should discard the cookie. A value of zero means the cookie should be discarded immediately.

The **CCookie**::**setMaxAge**() function can be used to set the maximum age of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setMaxAge**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getName

Synopsis
chchar * getName();

Purpose
Retrieve the name of the cookie.

Return Value
Upon successful completion, a string which contains the name of the cookie is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getName**() retrieves the name of the cookie. The **CCookie**::**setName**() function can be used to set the name of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**setName**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getPath

Synopsis
chchar * getPath();

Purpose
Retrieve the **Path** attribute of the cookie.

Return Value
Upon successful completion, a string which contains the value of the **Path** attribute of the cookie is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getPath**() retrieves the path on the server to which browser returns the cookie. The cookie is visible to all the pages in the directory you specify, and all the pages in that directory's subdirectories.

Consult RFC 2109 (available on the Internet) for more information on setting path names for cookies.
The **CCookie**::**setPath**() function can be used to set the **Path** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setPath**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getPorts

Synopsis
int getPorts(**int **** *portList*);

Purpose
Retrieve all ports in the portlist of the cookie.

Return Value
Upon successful completion, an integer which indicates the number of the ports in the portlist is returned. Otherwise, a negative value is returned.

Parameters
portList An integer array which contains all ports in the portlist.

Description
The function **getPorts**() retrieves all ports in the portlist of the cookie. This portlist contains the ports to which a cookie may be returned in a Cookie request header.

The **CCookie**::**addPort**() function can be used to add a new port into the portlist of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**addPort**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getSecure

Synopsis
bool getSecure();

Purpose
Retrieve the **Secure** attribute of the cookie.

Return Value
A boolean value which contains the **Secure** attribute of the cookie is returned.

Parameters
None.

Description
The function **getSecure**() retrieves the **Secure** attribute of the cookie. It indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.

The default value of **Secure** attribute is false and the cookie is sent from the browser to the server using any protocol. If the value of **Secure** attribute is set to true, sent on only a secure protocol.

The **CCookie**::**setSecure**() function can be used to set the **Secure** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setSecure**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getValue

Synopsis
chchar * getValue();

Purpose
Retrieve the value of the cookie.

Return Value
Upon successful completion, a string which contains the value of the cookie is returned. Otherwise, NULL is returned.

Parameters
None.

Description
The function **getValue**() retrieves the value of the cookie. The **CCookie**::**setValue**() function can be used to set the value of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**setValue**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::getVersion

Synopsis
int getVersion();

Purpose
Retrieve the **Version** attribute of the cookie.

Return Value
Upon successful completion, an integer which indicates the version of the cookie is returned. Otherwise, a negative value is returned.

Parameters
None.

Description
The function **getVersion**() retrieves the version of the cookie. Version 1 complies with RFC 2965, and version 0 complies with the original Netscape Cookie Specification. Cookies provided by a browser use and identify the browser's cookie version. By default, the value of Version is 0.

The **CCookie**::**setVersion**() function can be used to set the version of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**setVersion**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setComment

Synopsis
int setComment(**chchar *** *comment*);

Purpose

Set the **Comment** attribute which describes the purpose of this cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
comment A string containing the value of the **Comment** attribute of the cookie.

Description
The function **setComment**() sets the **Comment** attribute which describes the purpose of this cookie. Because cookies can be used to derive or store private information about a user, the value of the Comment attribute allows an origin server to document how it intends to use the cookie. The user can inspect the information to decide whether to initiate or continue a session with this cookie. Characters in value MUST be in UTF-8 encoding. [RFC2279]

Comments are not supported by Netscape Version 0 cookies.

The **CCookie**::**getComment**() function can be used to retrieve the **Comment** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**getComment**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setCommentURL

Synopsis
int setCommentURL(**chchar *** *commentURL*);

Purpose
Set the **CommentURL** attribute of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
commentURL A string containing the value of the **CommentURL** attribute of the cookie.

Description
The function **setCommentURL**() sets the **CommentURL** attribute of the cookie. Because cookies can be used to derive or store private information about a user, the **CommentURL** attribute allows an origin server to document how it intends to use the cookie. The user can inspect the information identified by the URL to decide whether to initiate or continue a session with this cookie.

The **CCookie**::**getCommentURL**() function can be used to retrieve the **CommentURL** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**getCommentURL**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setDiscard

Synopsis
int setDiscard(**bool** *discard*);

Purpose
Set the **Discard** attribute of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
Parameters
discard A boolean value that contains the new value of the **Discard** attribute.

Description
The function **setDiscard**() sets the **Discard** attribute of the cookie. The Discard attribute instructs the user agent to discard the cookie unconditionally when the user agent terminates.

The **CCookie**::**getDiscard**() function can be used to retrieve the **Discard** attribute of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**getDiscard**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setDomain

Synopsis
int setDomain(**chchar *** *domain*);

Purpose
Set the **Domain** attribute of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
domain A string containing the value of the **Domain** attribute of the cookie.

Description
The function **setDomain**() sets the **Domain** attribute of the cookie. The value of the **Domain** attribute specifies the domain for which the cookie is valid. If an explicitly specified value does not start with a dot, the user agent supplies a leading dot.

The **CCookie**::**getDomain**() function can be used to retrieve the **Domain** attribute of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**getDomain**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setMaxAge

Synopsis
int setMaxAge(**int** *maxAge*);

Purpose
Set the maximum age of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
maxAge An integer specifying the maximum age of the cookie in seconds;

Description
The function **setMaxAge**() sets the maximum age of the cookie. The value of the maxmimum age is the lifetime of the cookie in seconds. It is a decimal non-negative integer. To handle cached cookies correctly, a client should calculate the age of the cookie according to the age calculation rules in the HTTP/1.1 specification [RFC2616]. When the age is greater than delta-seconds seconds, the client should discard the cookie. A value of zero means the cookie should be discarded immediately.

The **CCookie**::**getMaxAge**() function can be used to retrieve the maximum age of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**getMaxAge**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setName

Synopsis
int setName(**chchar *** *name*);

Purpose
Set the name of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
name A string containing the name of the cookie.

Description
The function **setName**() sets the name of the cookie. The **CCookie**::**getName**() function can be used to retrieve the name of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**getName**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setPath

Synopsis
int setPath(**chchar *** *path*);

Purpose
Set the **Path** attribute of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
path A string specifying a path.

Description
The function **setPath**() sets the the path on the server to which browser returns the cookie. The cookie is visible to all the pages in the directory you specify, and all the pages in that directory's subdirectories.

Consult RFC 2109 (available on the Internet) for more information on setting path names for cookies.
The **CCookie**::**getPath**() function can be used to retrieve the **Path** attribute of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**getPath**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setSecure

Synopsis
int setSecure(**bool** *secure*);

Purpose
Set the **Secure** attribute of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters

secure A boolean value that contains the new value of the **Secure** attribute.

Description
The function **setSecure**() sets the **Secure** attribute of the cookie. It indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.

The default value of **Secure** attribute is false and the cookie is sent from the browser to the server using any protocol. If the value of **Secure** attribute is set to true, sent on only a secure protocol.

The **CCookie**::**getSecure**() function can be used to retrieve the **Secure** attribute of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**getSecure**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setValue

Synopsis
int setValue(**chchar *** *value*);

Purpose
Set the value of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
value A string containing the value of the cookie.

Description
The function **setValue**() sets the value of the cookie. The **CCookie**::**getValue**() function can be used to retrieve the value of the cookie.

Example
See **CResponse**::**addCookie**().

See Also
CCookie::**getValue**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

CCookie::setVersion

Synopsis
int setVersion(**int** *version*);

Purpose
Set the version of the cookie.

Return Value
Upon successful completion, zero is returned. Otherwise, non-zero value is returned.

Parameters
version An integer indicates the version of the cookie.

Description
The function **setVersion**() sets the version of the cookie. Version 1 complies with RFC 2965, and version 0 complies with the original Netscape Cookie Specification. Cookies provided by a browser use and identify the browser's cookie version. By default, the value of Version is 0.

The **CCookie**::**getVersion**() function can be used to retrieve the version of the cookie.

Example
See **CCookie**::**addPort**().

See Also
CCookie::**getVersion**(), **CResponse**::**addCookie**(), **CRequest**::**getCookies**().

Index