UNIT IV

COMMON GATEWAY INTERFACE

Overview

This topic provides information about CGI. Common Gateway Interface (CGI) is a standard, supported by almost all web servers, that defines how information is exchanged between a web server and an external program (CGI program).

The CGI specification dictates how CGI programs get their input and how they produce any output. CGI programs process data that is received from browser clients. For example, the client fills out a form and sends the information back to the server. Then the server runs the CGI program.

Programs that are called by the server must conform to the server CGI interface in order to run properly. We will describe this in further detail later in this chapter.

The administrator controls which CGI programs the system can run by using the server directives. The server recognizes a URL that contains a request for a CGI program, commonly called a CGI script. Depending on the server directives, the server calls that program on behalf of the client browser.

The server supports CGI programs that are written in C++, REXX, Java™, ILE C, ILE RPG, and ILE COBOL. It also supports multi-thread CGI programs in all of these languages capable of multiple threads.

You need to compile programs that are written in programming languages. Compiled programs typically run faster than programs that are written in scripting languages. On the other hand, those programs that are written in scripting languages tend to be easier to write, maintain, and debug.

The functions and tasks that CGI programs can perform range from the simple to the very advanced. In general, we call those that perform the simple tasks CGI scripts because you do not compile them. We often call those that perform complex tasks gateway programs. In this manual, we refer to both types as *CGI programs*.

Given the wide choice of languages and the variety of functions, the possible uses for CGI programs seem almost endless. How you use them is up to you. Once you understand the CGI specification, you will know how servers pass input to CGI programs and how servers expect output.

There are many uses for CGI programs. Basically, you should design them to handle dynamic information. Dynamic in this context refers to temporary information that is created for a one-time use and not stored as a static Web page. This information may be a document, an e-mail message, or the results of a conversion program.

Objectives::

- Describe how CGI has become more popular.
- Explain CGI functions.
- Explain HTML tags emulation
- Describe how interface programmings are used.
- Describe how server- browser communication is made

4.1 HTML FORMS

A form is an area that can contain form elements.

Form elements are elements that allow the user to enter information (like text fields, text area fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.

A form is defined with the <form> tag.

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

Input

The most used form tag is the <input> tag. The type of input is specified with the type attribute. The most commonly used input types are explained below.

Text Fields

Text fields are used when you want the user to type letters, numbers, etc. in a form.

```
<form>
First name:
<input type="text" name="firstname">
<br>
Last name:
<input type="text" name="lastname">
</form>
```

How it looks in a browser:

First name:

Last name:

Note that the form itself is not visible. Also note that in most browsers, the width of the text field is 20 characters by default.

Radio Buttons

Radio Buttons are used when you want the user to select one of a limited number of choices.

```
<form>
<input type="radio" name="sex" value="male"> Male
<br>
<input type="radio" name="sex" value="female"> Female
</form>
```

How it looks in a browser:

Male

Female

Note that only one option can be chosen.

Checkboxes

Checkboxes are used when you want the user to select one or more options of a limited number of choices.

```
<form>
I have a bike:
<input type="checkbox" name="vehicle" value="Bike">
<br>
I have a car:
<input type="checkbox" name="vehicle" value="Car">
<br>
I have an airplane:
<input type="checkbox" name="vehicle" value="Airplane">
</form>
```

How it looks in a browser:

I have a bike:

I have a car:

I have an airplane:

The Form's Action Attribute and the Submit Button

When the user clicks on the "Submit" button, the content of the form is sent to another file. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input.

```
<form name="input" action="html_form_action.asp"
method="get">
Username:
<input type="text" name="user">
<input type="submit" value="Submit">
</form>
```

How it looks in a browser:

Username: Submit

If you type some characters in the text field above, and click the "Submit" button, you will send your input to a page called "html_form_action.asp". That page will show you the received input.

More Examples

Checkboxes

This example demonstrates how to create check-boxes on an HTML page. A user can select or unselect a checkbox.

Radio buttons

This example demonstrates how to create radio-buttons on an HTML page.

Simple drop down box

This example demonstrates how to create a simple drop-down box on an HTML page. A drop-down box is a selectable list.

Another drop down box

This example demonstrates how to create a simple drop-down box with a pre-selected value.

Textarea

This example demonstrates how to create a text-area (a multi-line text input control). A user can write text in the text-area. In a text-area you can write an unlimited number of characters.

Create a button

This example demonstrates how to create a button. On the button you can define your own text.

Fieldset around data

This example demonstrates how to draw a border with a caption around your data.

Form Examples

Form with input fields and a submit button

This example demonstrates how to add a form to a page. The form contains two input fields and a submit button.

Form with checkboxes

This form contains three checkboxes, and a submit button.

Form with radio buttons

This form contains two radio buttons, and a submit button.

Send e-mail from a form

This example demonstrates how to send e-mail from a form.

Form Tags

Tag	Description
<form>	Defines a form for user input
<input>	Defines an input field
<textarea>	Defines a text-area (a multi-line text input control)
<label>	Defines a label to a control
<fieldset>	Defines a fieldset
<legend>	Defines a caption for a fieldset
<select>	Defines a selectable list (a drop-down box)
<optgroup>	Defines an option group
<option>	Defines an option in the drop-down box
<button>	Defines a push button
<isindex>	Deprecated. Use <input> instea

4.2 CGI CONCEPTS

What is CGI?

The common gateway interface is a standard that specifies the interaction between external applications and a web server during a URL request. CGIs allow the manipulation and presentation of information in real time. Two examples are processing forms or presenting dynamic web pages. In this tutorial we are concerned with processing forms, but the standard governs any request that is handled by an application other than the server.

When a browser requests information in a URL that needs action by an external CGI application, the web server will send the data coming from the browser to the designated CGI application. In our case Frontier is the CGI application and grabs the data from the server, takes appropriate action, and then returns a reply. Figure 1 depicts the CGI interface graphically.

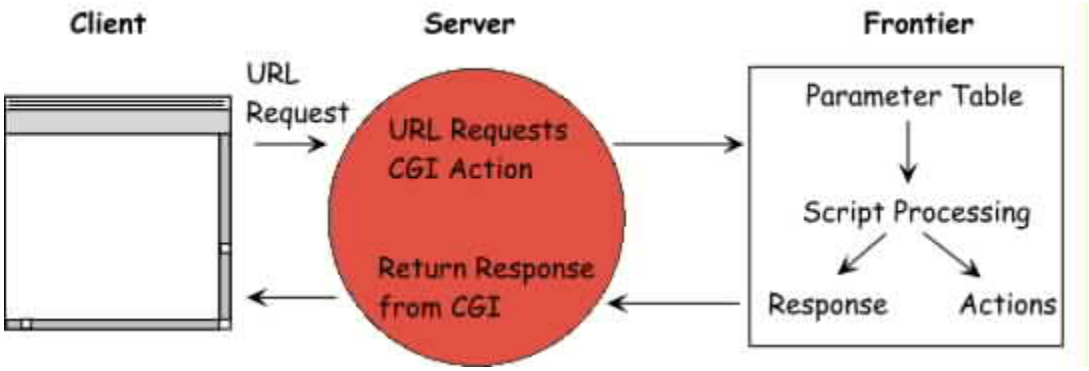


Figure 2.1 - The CGI interface

In most cases the reply will be an HTML document giving feedback to the user on the data that they sent, but any legal MIME type can be sent. CGI applications can send back HTML, plain text, images or even audio files. It is totally under the control of the CGI application, but the type of data must be speicifed by a MIME type in the header of

the document. Below is an example of a simple HTML document sent back with its corresponding MIME type. The MIME type is in line 1.

Listing 1 - An example of text that might be sent back from a CGI application

```
1:  Content-type: text/html
2:  <HTML><HEAD>
3:  <TITLE>output of HTML from CGI script</TITLE>
4:  </HEAD><BODY>
5:  <H1>Sample output</H1>
6:  What do you think of <STRONG>this?</STRONG>
7:  </BODY></HTML>
```

Don't worry about MIME types, since Frontier will handle much of the dirty work for you.

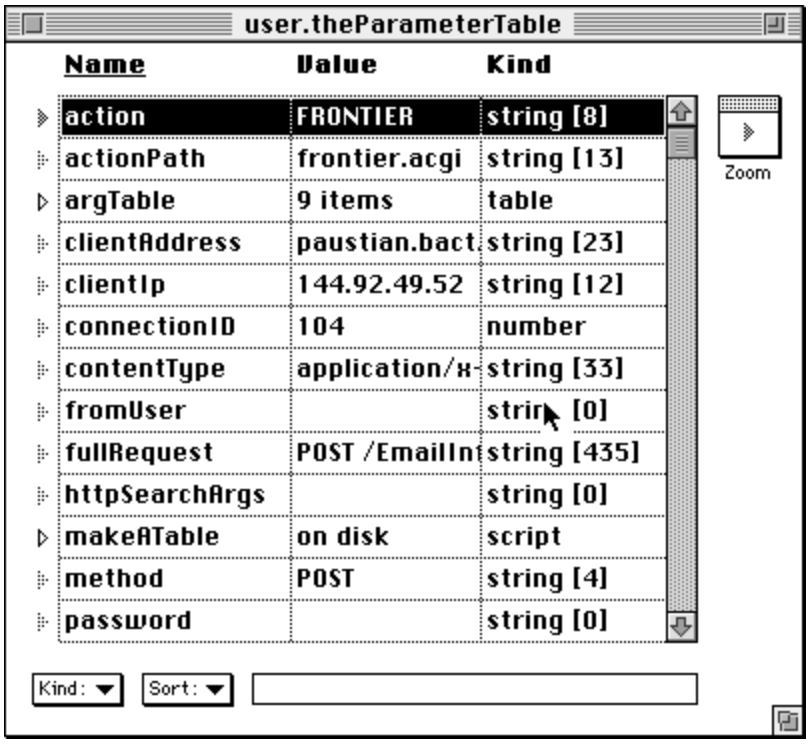
The CGI interface to Frontier

Web servers on the Macintosh communicate with Frontier through Apple Events. During a CGI request, a web server will send an 'sdoc' Apple Event to Frontier, containing the following data from the browser that activated the CGI.

path_args	The data in the URL following the "$" character
http_search_args	Data passed when using the GET method. Normally you will not want to use this with Frontier
post_args	Data passed when using the POST method
method	Identifies the method being used. Normally POST
client_address	The IP address of the client making the request
username	The username of the client (only if using the security features)
password	The password of the client (only if using the security features)
from_user	More information about the user, typically an e-mail address

server_name	Name of the requesting server
server_port	Port the server is running on
script_name	The URL that was sent to the server from the client
referer	The URL that the client was viewing before sending the request
user_agent	The name of the client software
content_type	MIME content type of post_args.
action	The action that caused the CGI. In our case, it should be FRONTIER
action_path	Path to file from the base server folder. (Normally frontier.fcgi)
client_ip	Contains the IP address of the client
full_request	Complete, unmodified text of the request from the client

The only really important thing for these tutorials is the post_args parameter which contains the data from the form. The 'sdoc' Apple Event is actually received by the webserver suite, which is a collection of Frontier scripts for handling CGI requests. This suite parses the incoming information from the server into separate items and builds a table in Frontier containing the data. I will refer to this table as the **parameter** table. A sub-table of the parameter table (argTable) is built from post_args and contains all the information from the form. CGI scripts that are written to handle this data, get passed a variable containing a pointer to the parameter table. The job of our script is to extract the necessary data from the parameter table and take appropriate action.



4.3 HTML TAGS EMULATION

Many folks are attempting to use the HTTPClient to mimick a browser submitting form data. While the transformation from html to code is really quite straightforward, it assumes a minimal knowledge of html. So, the best place to start is get an understanding of html - the specs are available from w3.org:

- HTML Home Page at W3,
- HTML-3.2,
- HTML-4.01,
- Forms in HTML-4.01,

The HTML-4.01 section on forms gives a good introduction.

In order to make this easier however, this doc attempts to explain more specifically how to create the necessary code using the HTTPClient in order duplicate what a browser would send when you submit a given form. What follows here is not meant to be exhaustive - if you see something in your form that is not explained here, then please refer to the html specs.

For the rest of this document, I'm going to assume you've got a page under the URL http://www.some.org/some/form.html which contains an html form whose submission you want to emulate in java code. Html tag and attribute names are case insensitive, so while I'll always use uppercase for these (i.e. FORM or NAME=), they may appear in a different case in the actual html you're aiming at emulating (but note that the values are not case insensitive, i.e. if you see a NAME=UsEr then you must send the string "UsEr" using that exact case).

Finding and Interpreting the FORM tag

The first thing to do is to find the beginning and the end of the form you're interested in. The beginning is marked with a `<FORM ...>`, and the end with a `</FORM>`. There may be multiple forms in the document, so make sure you pick the one you're actually interested in (this can be achieved either by counting the forms and picking the n-th one, or by looking at which input fields the form contains and making sure it contains the ones you're interested in).

The start tag has the general form

```
<FORM ACTION="some/url" METHOD="method" ENCTYPE="enc-type">
```

There may be additional attributes (but they're not of interest here), and the METHOD and/or ENCTYPE may be missing.

The ACTION attribute specifies the URL to which to send the form data; it is taken relative to the document's URL. If you are unsure how to contruct the resulting URL, you can use the URI class to do so:

```
URI doc_url  = new URI("http://www.some.org/some/form.html");
URI form_url = new URI(doc_url, "some/url");
```

The METHOD attribute describes the method to use; it can take the values GET or POST. If it is missing, GET is assumed. As the name implies, this is the method you must use when posting the form data.

The ENCTYPE attribute specifies the content-type (and therefore the encoding) for the data to be sent. If not present, the default is "application/x-www-form-urlencoded", which is what will be used if you create an array of NVPair's and pass them to the Get or Post method of HTTPConnection. The other popular value is "multipart/form-data" - if you see that, you can use the Codecs.mpFormDataEncode() to correctly encode the form data.

The INPUT tags

All the actual data to be entered and submitted is specified by INPUT tags. These have the form

```
<INPUT TYPE="type" NAME="name" VALUE="value">
```

Again, there may be other attributes, but in general they're not of interest here; the VALUE attribute is often missing. For each INPUT tag you need to decide whether to create a corresponding NVPair, and if so, what name and value to put into it. The name for the NVPair is always what's given in the NAME attribute; the value is either what you enter in that field or what is in the VALUE attribute.

The type attribute specifies the what kind of element is displayed, e.g. an input field, a button, or nothing at all; if missing, the type is assumed to be "text". This also determines whether you need to create a corresponding NVPair and what to put in its value:

TYPE=text
: A simple text field is displayed. An NVPair must always be created for each such input tag, and the NVPair's value is whatever you'd enter in that field (which may be the empty string if nothing is entered).

TYPE=password
: A simple password field is displayed - treat this exactly like a TYPE=text.

TYPE=checkbox
: A checkbox is displayed. If the box is supposed to be checked, then create an NVPair with the value given in the VALUE attribute; otherwise skip this field. Note that you need to create an NVPair for each checkbox that is selected, even if some of them have the same NAME and/or VALUE attributes.

TYPE=radio
: A radio button is displayed. There may be multiple of these with the same name, which creates a group (in which only one of the buttons can be selected at any time). Create an NVPair with value given in the VALUE attribute of that button which is checked; if no buttons are checked, don't create an NVPair for this tag.

TYPE=hidden
: Nothing is displayed. Create an NVPair with the name and value as given in the NAME and VALUE attributes.

TYPE=submit
: A submit button is displayed. If the tag contains a NAME attribute, then create an NVPair with the name and value as given in the NAME and VALUE attributes; if no NAME attribute is specified, skip this.

TYPE=reset
: A reset button is displayed. Ignore this.

The SELECT and OPTION tags

These are used to provide a drop down selection box. An NVPair must be created with the name from the NAME attribute of the SELECT tag and the value of the VALUE attribute of the selected OPTION tag.

The TEXTAREA tag

This displays a multiline text entry area. An NVPair must created with the name from the NAME attribute of this tag, and the value must be whatever was entered in the textarea (which may be the empty string if nothing was entered).

A Complete Example

Following is an example form that uses all of the above described elements.

```
<FORM ACTION="/cgi-bin/test" METHOD="POST">
Text: <INPUT NAME="text1">
<BR><INPUT type=checkbox NAME="ok" VALUE="yes">Ok?
<BR><INPUT type=checkbox NAME="ok" VALUE="totally">Really Ok?
<BR><INPUT type=radio NAME="when" VALUE="yesterday">yesterday
   <INPUT type=radio NAME="when" VALUE="today">today
   <INPUT type=radio NAME="when" VALUE="tomorrow">tomorrow
<INPUT type=hidden NAME="secret" VALUE="data">
<INPUT type=hidden NAME="secret" VALUE="more data">
<INPUT type=hidden NAME="also-hidden" VALUE="data">
<BR><SELECT NAME="aChoice">
        <OPTION VALUE="one">1
        <OPTION VALUE="two">2
        <OPTION VALUE="three">3
        </SELECT>
<BR><TEXTAREA NAME="comments" ROWS=10 COLS=80>
        </TEXTAREA>
<BR><INPUT type=submit VALUE="Doit">
</FORM>
```

Assuming we enter the text "Hell hath no fury..." for the text, we check the second checkbox, choose today, select the third option, and give no comment, then the resulting code to duplicate this would look like the following:

```
// set up the uri according to the action attribute
URI form_uri = new URI(doc_uri, "/cgi-bin/test");
HTTPConnection con = new HTTPConnection(form_uri);

// create the NVPair's for the form data to be submitted
NVPair[] form_data =
        new NVPair[] {
           new NVPair("text1", "Hell hath no fury...");
           new NVPair("ok", "totally");
           new NVPair("when", "today");
           new NVPair("secret", "data");
           new NVPair("secret", "more data");
           new NVPair("also-hidden", "data");
           new NVPair("aChoice", "three");
        };

// POST the form data, as indicated by the method attribute
HTTPResponse rsp = con.Post(form_uri.getPathAndQuery(), form_data);
```

An Example Using File-Upload

Following is an example form that uses file-upload.

```
<FORM ACTION="/cgi-bin/test-upload" ENCTYPE="multipart/form-data"
  METHOD="POST">
<INPUT type=file NAME="upload_file">
<INPUT type=hidden NAME="secret" VALUE="data">
<BR><INPUT type=submit VALUE="Doit">
</FORM>
```

Assuming we select the file "hhgttg.txt", then the resulting code to duplicate this would look like the following:

```
// set up the uri according to the action attribute
URI form_uri = new URI(doc_uri, "/cgi-bin/test-upload");
HTTPConnection con = new HTTPConnection(form_uri);

// create the NVPair's for the form data to be submitted and do the
// encoding
NVPair[] files = { new NVPair("upload_file", "hhgttg.txt") };
NVPair[] opts  = { new NVPair("secret", "data") };
NVPair[] hdrs  = new NVPair[1];
byte[] form_data = Codecs.mpFormDataEncode(opts, files, hdrs);

// POST the form data, as indicated by the method attribute
HTTPResponse rsp = con.Post(form_uri.getPathAndQuery(), form_data, hdrs);
```

4.4 SERVER- BROWSER COMMUNICATION

Up to this point, we have spent a lot of time talking about the mechanics of scripting within our applications, but we have only provided hints as to how the scripting technology can be used within the web environment. ASP provides seven objects within the scripting engine that your scripts can use to work within the context of an application and to facilitate the scripts communication with the web browser that is calling the pages. These objects are known as the ASP intrinsic objects.

Although in the previous chapters of the book, we have been providing most of our code examples in both languages, we will avoid doing that in this chapter since the ASP objects behavior is identical in both languages. So for the purposes of readability, we will standardize on VBScript in this chapter.

ASP Intrinsic Objects

At the end of the last chapter, we discussed how ASP is dependent on a series of component objects. These intrinsic ASP objects fall into the following categories:

- **Communicating between Browser and Server**

  There are two objects that work in tandem within ASP to facilitate communication. The Request object is used to retrieve information that is submitted to the web server by the users web browser. The Response object allows scripts on the server to insert dynamic content into the stream of information flowing back to the user.
- **Creating a Context on the Server**

  The Application object creates a common context for all of the ASP pages contained within a web application as defined in IIS. The Session object provides a memory space for each user who is entering the application, allowing them to keep user-specific content in a persistent state. These objects will be covered in the next chapter, Creating a Context for ASP.
- **Controlling the Server Environment**

  The Server object provides a number of utilities for retrieving information about the server environment. This object will be covered in the next chapter, Creating a Context for ASP.

  Handling Errors

  The ASPError object, which is new in IIS 5, provides a persistent state for error information so that is may be retrieved from another page context.

  Maintaining Transactional Integrity

  The ObjectContext object permits us to establish transactional integrity within our web page.

The ASPError and ObjectContext objects will be covered in the chapter, Handling Errors in ASP.

4.5 E-MAIL GENERATION

Acoupleofpossibilitiesare:

1.ShellExecutea"mailto"command.

2.UseSimpleMAPI -see"SendingMessageswithSimpleMAPI"inMSDN.

You can test the capabilities of the mailto syntax using the Start, Run dialog. It can't automatically send the email, or add attachments.

4.6 CGI CLIENT SIDE APPLETS

CGI and Java are two totally different animals. CGI is a specification that can be used by any programming language. CGI applications are run on a Web server. Java is a programming language that is run on the client side.

CGI applications should be designed to take advantage of the centralized nature of a Web server. They are great for searching databases, processing HTML form data, and other applications that require limited interaction with a user.

Java applications are good when you need a high degree of interaction with users: for example, games or animation.

Java programs need to be kept relatively small because they are transmitted through the Internet to the client. CGI applications, on the other hand, can be as large as needed because they reside and are executed on the Web server.

You can design your Web site to use both Java and CGI applications. For example, you might want to use Java on the client side to do field validation when collecting information on a form. Then once the input has been validated, the Java application can send the information to a CGI application on the Web server where the database resides.

CGI Client-Side Examples

These examples are taken from chapter 17. The chapter also includes complete coverage of all the HTML FORM elements, ISINDEX, ISMAP, and methods for sending GET and POST data from Java applets. Chapter 16 covers HTTP, cookies, and public-key

cryptography. Chapters 19 and 20 cover JavaScript, with specific examples on using JavaScript to validate FORM values before they are submitted, processing cookies on the client instead of on the server, calling Java from JavaScript, and calling JavaScript from Java. See the source code archive for example code.

Java Source	Description	On-Line Example
SearchYahoo.java. Requires SearchService.java.	A simple example of an applet as a CGI client that works like an HTML form, sending GET data and having the browser display the results. This one talks to the Yahoo! search engine.	SearchYahoo.html
SearchExcite.java. Requires SearchService.java.	Another example of an applet as a CGI client. This one talks to the Excite search engine, demonstrating one advantage of using Java instead of forms: you can more easily reuse code in other applications (this one extends the same class that the Yahoo applet did).	SearchExcite.html
ShowFile.java	An applet that sends data via GET and then reads the results itself (rather than having the browser display results).	ShowFile.html
Weather.java. The client-side (applet) requires CityChooser.java and WeatherPanel.java. The server-side is answered by the WeatherInfo script, which then invokes WeatherInfo.java.	An applet that sends data via POST and then reads the result.	Weather.html

4.7 CGI SERVER SIDE APPLETS

These examples come from chapter 18. The chapter also includes complete coverage of all the CGI environment variables, a URL decoder, CGI parser, and Cookie parser in Java, server-side Java and the Servlet API, and a quick overview of non-CGI

alternatives such as NSAPI, ISAPI, LiveWire, and JDBC. JDBC is covered in depth in chapter 15. Servlets are a particularly good alternative for people who are installing/customizing their own server, or whose employer's or ISP's server already supports them. You should **very** seriously consider using them instead of "standard" CGI with Java if you fall in this category. Please see my tutorial on Java servlets and JSP 1.0 for more detail.

Shell Script Interface (Unix Specific)	**Java Source (Portable)**	**Description**	**On-Line Example**
CgiHello	N/A	An extremely simple CGI Script that outputs "Hello, World".	CgiHello
ShowData	N/A	A simple CGI script that shows any attached data.	ShowData
CgiGet	CgiGet.java. Requires CgiShow.java. Note that some browsers will try to interpret the HTML strings in the print statements and the result may be formatted strangely when viewed in the browser. But you can save the file to disk to edit it normally.	A simple script that passes the query data from the QUERY_STRING variable to a Java program that builds a page showing the data supplied.	CgiGet
CgiCommandLine	CgiCommandLine.java	A simple script that passes the command-line data to a Java program that builds a page showing the data supplied. Arguments are separated by plus signs ("+") and cannot contain an equals sign ("=").	CgiCommandLine
IsIndex	IsIndex.java	A script that passes the data to a Java program that builds an HTML document	IsIndex

		that uses ISINDEX. Data-entry page or results page is built depending on whether any data is supplied.	
CgiPost	CgiPost.java	A script that invokes a Java program without passing any data to it. The program reads data from standard input.	CgiPost.html
ShowParse	ShowParse.java. Requires QueryStringParser.java, CgiParser.java, LookupTable.java, URLDecoder.java, and StringVector.java.	A script that passes data to a Java program that separates the various attached variables, URL decodes their values, and produces a table of the results. Can accept GET or POST.	ShowParse version using GET. See below for a version that sends POST data.
ShowParse	ShowParse.java. Requires QueryStringParser.java, CgiParser.java, LookupTable.java, URLDecoder.java, and StringVector.java.	A script that passes data to a Java program that separates the various attached variables, URL decodes their values, and produces a table of the results. Can accept GET or POST.	AdBuilder.html; sends POST data to ShowParse. See above for a version that sends GET data.
CssTest	CssTest.java. Requires CssChoices.java, CookieParser.java, CgiParser.java, LookupTable.java, and URLDecoder.java.	A script that passes data to a Java program that builds a page to let you test out cascading style sheet properties. Properties selected are stored as cookies, which are parsed and used as defaults in later sessions.	CssTest

4.8 AUTHORIZATION & SECURITY

Authentication, Identification, and Authorization

Authentication is the process by which a person or other entity (such as a server) proves that it is who (or what) it says it is. Authentication is achieved through presenting something that you know, something that you have, some unique identifying feature, or some combination of these. A common example is the way you authenticate yourself in order to use a teller machine: you insert your ATM card (something you have) and enter your personal identification number (PIN, something you know). Unique identifying features include such things as fingerprints, retina patterns, and voice prints.

It is always desirable to authenticate the person or server with which you are dealing before transferring something valuable, such as information or money. Authentication, however, is time consuming and can inconvenience users. For example, once having shown your photo ID card to enter the Apple Worldwide Developer's Conference, you would not want to get it out again every time you walked into one of the conference rooms. To make situations like this more convenient and efficient, many systems use some method of **identification**, which verifies that the person or entity is the same one you communicated with last time. The means of identification can be through the use of a ticket or token issued when authentication is done. For example, the conference badge you are given to wear during the Developer's Conference identifies you as a legitimate attendee who was authenticated when you first came in.

In general, authentication or identification is not sufficient to gain access to information or code. For that, the entity requesting access must have **authorization**. Authorization requires first a determination that the authenticated entity has the appropriate **permissions**—that is, the right to the specific type of access (such as read, write, or execute) requested—and then the actual granting of that access. For example, the mere possession of a conference badge does not grant you the right to enter a restricted area, such as the speakers' preparation room. You must have permission to enter this area

(indicated, in this case, by the color of your badge), and you must be granted access by the guard at the door.

Authentication and Identification methods available in Mac OS X are described in “Authentication and Identification Methods.” Permissions in BSD and Mac OS X are described in “Permissions.” Authorization is discussed in more detail in “Authorization.”

Aspects of Security

The fundamental purpose of security is to control who has access to valuable property, whether physical or intellectual. This is the reason we have locks on the doors of our houses, why the military encrypts classified information, and why Mac OS X enables you to require a password every time someone logs on to your computer.

Security features on a personal computer can be classified into two general groups: those designed to protect programs and data on the computer from unauthorized access by users on the system (“local security”); and those designed to protect the system, programs, and data from unauthorized access over a network or other transport medium, such as removable disks (“remote transport security”).

When considering local security, you must be aware of whether access is being controlled by the operating system or by the application itself.

In this section:

Local Security
Remote Transport Security
System-Restricted or Self-Restricted Access Local Security

Local security is important when a computer is being shared, such as in libraries or schools; or when an unauthorized person might get access to the computer, such as a computer kept in an open cubicle in a large office. Security features useful in such environments include the password protection offered by the Finder, encryption of data

provided by FileVault, BSD access permissions, and access permissions added to applications through use of Authorization Services.

Remote Transport Security

Remote transport security is important to all users, and especially to users whose computers are connected to a LAN or to the Internet. Web browsers, for example, use secure transport protocols (“Protocols for Secure Communication”) to protect data from interception while in transit, digital signatures (“Digital Signatures”) to ensure data integrity, and digital certificates (“Digital Certificates”) to verify the identity of people or servers trying to get access to data. Many of the security APIs provided by Mac OS X are useful in this regard, including the secure networking APIs (Secure Transport, CFNetwork, and URL Loading System), Keychain Services (used to store certificates, passwords, and encryption keys), and Certificate, Key, and Trust Services.

System-Restricted or Self-Restricted Access

It is important to understand that certain forms of access permission are enforced by the operating system, whereas others are enforced by individual applications. BSD permissions (“BSD”) control who can execute a program or open a file, and are built into the operating system. On the other hand, if you want finer-grained control over access, such as restricting certain operations to a subset of users, you must enforce these restrictions yourself. Authorization Services provides functions you can use to implement such restrictions, and you can make the restrictions optional so that they operate only when your application is being used in an environment where they are necessary. For example, you might want to restrict access to some application preferences to administrators on a shared computer but not require a password when the computer is not shared. See Authorization for Everyone, Technical Note TN2095, for techniques and sample code for implementing self-restricted access permissions.

Summary

- The Common Gateway Interface is the protocol by which programs interact with Web servers. The versatility of CGI gives programmers the opportunity to write

gateway programs in almost any language, although there are many trade-offs associated with different languages. Without this ability, making interactive Web pages would be difficult at best, requiring modifications to the server and putting interactivity out of the reach of most programmers who are not also site administrators.

Key words ::

- CGI,HTML,HTTPS
- Windows DNA
- Java servlet API ,Bean ,jar
- NSAPI, ISAPI, LiveWire, and JDBC.
- Caveats

Key term quiz::

1. CGI stands for ----------.

2. Caveats mean ----------.

3. ---------- is the process by which a person or other entity (such as a server) proves that it is who (or what) it says it is.

4. ----------is a specification that can be used by any programming language

5. CGI specifies which information is communicated between ---------- and such a ----------, and how.

Multiple Choice::

1. The lifecycle of the servlet is defined in the _____interface
    a. javax.servlet.Servlet
    b. javax.servlet.ServletContext
    c. javax.servlet.ServletConfig
    d. javax.servlet.GenericServlet

2. A submit action on the html form invokes the _____method on the servlet.
    a. doGet()
    b. doPost()
    c. doDelete()
    d. method mentioned in the method attribute of the Form Tag

3. The init method is invoked
    a. For every client request
    b. Only once during entire life cycle
    c. For every session
    d. Every time the service method is invoked

4. Servlets can be made to execute a single request thread by implementing
    a. javax.servlet.Servlet
    b. java.lang.Runnable
    c. javax.servlet.SingleThread
    d. None of the above

5. To retrieve all the initialization parameter names from within a servlet, one of the following has to be performed
    a. the ServletContext object is used to invoke getServletInfo();
    b. the ServletConfig object is used to invoke getInitParameterNames();
    c. the ServletRequest object is used to invoke getParameter();
    d. none of the above

Review Questions

In your own words briefly answer the following

1. What are servlets?
2. Life cycle of a servlet?
3. Threading Model of a servlet. Is it single Threaded or Multithreaded
4. What is a single threaded model servlet? What is its effect on performance?
5. How to make a servlet single Threaded?
6. Session management in servlet. How are sessions created in a servlet?

7. Does a session created in a servlet shared by a JSP called by that servlet?
8. Difference between Generic and HttpServlet
9. How to forward a request from a servlet? List the different ways of doing the same.
10. How to redirect a request in HTTP Servlet

Lab Excerise

Complete the following excersies

1. Write a java program to access the information from Database using JDBC Connectivity.

2. Develop a server side program using Servlet to perform Order Processing

3. Develop a server side program using JSP to perform Order Processing