VSS University of Technology

---------- BURLA ----------

DEPARTMENT OF COMPUTER APPLICATIONS (MCA)

MCA-304 Enterprise Web-based Computing with Java 5TH Semester

Veer Surendra Sai University of Technology, Burla
(A UGC & AICTE affiliated Unitary Technical University)
Sambalpur-768018, Odisha
INDIA

www.vssut.ac.in

MCA-304 Enterprise Web-based Computing with Java L-T-P: 3-1-0

Prerequisite: Familiarity with java, HTML, web technology.

UNIT – I (8 Hours)

Introduction to Networking Basics Of Networking, Overview Of The OSI Model, Socket Programming, Client Sockets And Server Socket, Multicast Sockets

Javabeans – Javabean Architecture, Bean Properties, Methods And Events, Bean Introspection

UNIT – II (10 Hours)

Java Database Connectivity [JDBC] DBMS Concepts, RDBMS & Understanding basic database design, SQL, Introduction To SQL, DDL, DML, Joins, JDBC, Basics Of Database Connectivity, Introduction To JDBC, JDBC Architecture, JDBC Interfaces, JDBC Exceptions, Prepared Statement, Callable Statement, Stored Procedure And Functions, Triggers

UNIT – III (12 Hours)

Servlets Introduction To Web Application Development, Introduction of a 2 & 3 Tier Architecture, Server Side Programming, Introduction To Servlets, Comparing Servlets With CGI, Servlet Lifecycle, Servlet With Html, Server Side Includes, Servlet Chaining, HTTP Tunneling, Session Management, Servlets With JDBC, Inter Servlet Communication, Deployment Descriptor (web.XML), Servlet Context & Config Objects, ,Event Handling in Servlet, Jasper Report generation & Calling Using Servlet.

UNIT – IV (8 Hours)

. Enterprise Java Beans Architecture, Introduction To Session Beans, Characteristics, How To Write & Call Session Beans, Understanding EJB Security, Introduction To Entity Beans & its features, Example Server, Example Client, Transactions - Need, benefits, model, isolation

Text Books:

1. Ivan Bayross, Web Technologies Part-I, BPB Publications, P/705

2. Ivan Bayross, Web Technologies Part-II, BPB Publications, P/922

3. OracleTM - JAVA Tutorial (Web), W3School Tutorial

4. Java Server Programming J2EE 1.3 Edition

5. N.P.Gopalan and J.Akilandeswari, Web Technologies – A developer's Perspective, PHI

6. R. Bangia, Multimedia and Web Technology, Fire Wall Media, New Delhi.

Reference Books:

1. Core Java Part 2 Advanced Features – Sun Microsystems press

2. J2EE™ Tutorial, The, 2nd Edition By Eric Armstrong, Jennifer Ball, Stephanie Bodoff, Stephanie Bodoff, Stephanie Bodoff, Debbie Carson, Ian Evans, Dale Green, Kim Haase,

Eric Jendrock. Published by Addison Wesley

Course Outcomes:

1. To learn the graphics and animation on the web pages, using Java Applets

2. To learn and design a full set of Event driven UI widgets and other components, including windows, menus, buttons, checkboxes, text fields, scrollbars and scrolling lists, using Abstract Windowing Toolkit (AWT) & Swings

3. To learn Java Data Base Connectivity (JDBC) so as to retrieve and manipulate the information on any relational database through Java programs.

4. To learn the server side programming using Servlets and JSP.

5. To learn Java Bean so as to make the reusable software components

6. To learn the invocation of the remote methods in an application using RMI

7. To learn the development of Enterprise based applications, using EJB: Stateful, Stateless and Entity Beans.

8. To make the students familiar with Struts frameworks, which gives the opportunity to reuse the codes for quick development.

9. To learn Hibernate for the mapping of Java classes and objects associations to the relational database tables.

DISCLAIMER

This document does not claim any originally and cannot be used as a substitute for prescribed textbooks. The information presented here is merely a collection of knowledge base by the committee members for their respective teaching assignments. Various online/offline sources as mentioned at the end of the document as well as freely available material from internet were helpful for preparing this document. The ownership of the information lies with the respective authors/institution/publisher. Further, this study material is not intended to be used for commercial purpose and the committee members make no representations or warranties with respect to the accuracy or completeness of the information contents of this document and specially disclaim any implied warranties of merchantability or fitness for a particular purpose. The committee members shall not be liable for any loss or profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

MODULE 1

Designing web pages HTML:

INTRODUCTION

Web Page Creation Using HTML: Introduction

1. Getting Ready
   - What Software is Needed
   - FourSteps to Follow
2. What Will Be On a Page
   - Technical, Content, & Visual Issues
3 Creating the HTML Files
   - Structural Tags
   - Header Tags
   - Formatting Tags
   - Separator Tags
   - List Tags
     - Three Types of Lists
   - Link Tags
     - Two Kinds of Links
     - Two Kinds of URLs
4. Saving Your HTML File
5. Testing Your Web File
   - Look at File on the Web
   - Viewing the HTML Source
   - Copying the HTML Source
6 Putting Your File on the Server
   - Creating a Web Directory
   - Uploading Your HTML File

Getting Ready

WWW, World Wide Web, is a system used to find and access different Internet resources. It uses hypertext to cross-reference or link related resources anywhere on the Internet.

HTML(Hypertext Markup Language) is the language used by the Web to define and display its files. These files can contain text, or multimedia.

HTML files are ASCII text files that contain the text to be displayed and the markup tags that tell how to display them.

If you have traveled the Internet and searched the Web, then you may be

interested in creating and authoring your own web page.

What Software is Needed

The Internet software you will need for web authoring includes:

◆ Web browser to view a web page, suchas Netscape, Internet Explorer, Mosaic, or even a text browser like Lynx.

◆ Text editor to create the HTML file; such as Notepad or WordPad, etc.

◆ FTP (File Transfer Protocol) program to upload a page. There are several available for a Mac or a PC.

◆ Graphics editor to create new graphics. This is optional. If you decide you need one there are several available.

What Steps to Follow

Creating a page on the Web can be a simple or complex process. However, the steps are always the same:

◆ Decide what information will be on a page and how that information will be arranged on the page.

◆ Create the HTML file with the text and commands using any editor.

◆ Test the page in various browsers and on various platforms.

◆ Finally, upload the HTML file to the Web server.

What Will Be on a Page

There are three types of standards to keep in mind when composing a page.

They are:

- Technical
- Content
- Visual

Technical Issues

Technical standards define what linksa page should have and what HTML tags every page should have.

Example: Every departmental page should have a link back to the Ohio

University home page.

Content Issues

Content standards describe what items every page should contain.

Example: Every page should contain,the authors name, E-mail address, and the date of creation.

Visual Issues

Visual standards describe what every page should have for appearances.

Example: It describes the graphics, the format, the layout, and suggested colors for the background.

Creating the HTML File

You can use any word processor to compose this file. We will be using Wordpad during the class.

HTML commands or tagsare enclosed in angled brackets: < >.

Some tags stand alone and some come inpairs. In paired tags the ending

tag starts with a slash: /.

The Types of HTML Tags

We will cover six basic types of commands:

- Structural tags (mandatory)
- Formatting tags (optional)
- Separator tags (optional)
- Heading tags (optional)
- List tags (optional)

- Link tags (optional)

6

Structural tags:

These are at the beginning and end of an HTML file.

<HTML> </HTML> For an HTML document

<HEAD> </HEAD> For the head section

<TITLE> </TITLE> For the title of the bookmark <BODY> </BODY> For the body section

Example: <TITLE> My Personal Page </TITLE>

Headings:

There are 6 levels of Headings. Level 1 is the largest font size.

<H1> </H1> Heading level 1

<H2> </H2> Heading level 2

<H3> </H3> Heading level 3

<H4> </H4> Heading level 4

<H5> </H5> Heading level 5

Example: <H2> This is My Personal Page</H2>

Formatting tags:

These tags affect the formatof the word or sentence.

<I> </I> For Italic text

<B> </B> For Bold text

<U> </U> For Underlined text

<STRONG> </STRONG> For Bold text

Example: <I> My Personal Page</I>

7

Separators tags:

These tags separate words,or sentences on a page.

<P> Start printing a new Paragraph

 Break; breaks up text onto two lines <HR> Make a Horizontal Rule (or line)

Example: <P> This is the beginning of the second paragraph on my Personal Web page.

Types of Lists:

There are three main types of lists. An Ordered Listis a list of numbered items. An Unordered Listis a list of unnumbered items. A Definition List is used for definitions of terms, say, in a glossary.

Lists:

<UL> </UL> Make an Unordered List

<OL> </OL> Make an Ordered List

<DL> </DL> Make a Definition List

<LI> Used for each List Item

Example Tags: <OL> Here is a list of my hobbies:

<LI> swimming

<LI> hiking

<LI> fishing

</OL>

Example List: Here is a list of my hobbies:

1. swimming

2. hiking

3. fishing

8

Links:

Hyperlinksare what the Web is all about. Before you create Hyperlinks, you need to understand URLs.

A URL(Uniform Resource Locator) is a Web address. Just as you can have two forms of E-mail address, a long and a short one, you can have two forms of a URL address.

- AbsoluteURL - This is a complete address. Use this if the link refers

to a page or file on another server (computer).

◆ RelativeURL - This is a shortened address, without the server name.

Use this if the link is to a page or file on the same server (computer).

Example Absolute URL:

http://home.netscape.com/training/chapter1.html

Example Relative URL:

chapter1.html

Type of Links:

There are two main types of hyperlinks we will cover in this class:

◆ Link from the current document to beginning of another document.

9

◆ Link from the current document to a specific spot(anchor) in another

document or in the same document.

Link Tags:

<A> </A> Create Link to another document

HREF = URL URL of document to be linked text The text to be clicked, usually in blue

Example Tag with Absolute URL:

<A HREF="http://www.ohiou.edu/mainmenu.html">Menu</A>

Example Tag with Relative URL:

<A HREF="mainmenu.html">Menu</A>

Link Tags to an Anchor Spot

To create a link to a specific spotin a second document,there must be an

anchorname in the second document. Then you create a link in the first

document that points to the anchor in the second document.

<A> </A> Create link to a document

NAME = name Name of the anchor spot

Example Creating an Anchor Name:

<A NAME=“Start_Place”>Table of Contents</A>

Example Referencing an Anchor:

<A HREF="#Starting_Place”>Go To the Contents</A>

10

Saving Your HTML File:

- Go to the Filemenu and choose Save As
- Enter any valid file name, with an extension of .html

The first file in your account should be named:index.html

- Specify your HTML Directory
- In the “Save as Type” specify “All Files”

Testing Your Web Page:

Test the page under various browsers, including Lynx. Test the page under various platforms and with various screen resolutions. Also remember to test all the links on the page. To test our HTML file using Netscape:

- Start Netscape
- Go to Fileand choose Open Page or Open File
- Type in the complete address or click Choose File.
- After selecting the appropriate file, click Open.

11

Viewing the Source on the Web

The best way to learn is through example. If you see a layout or design that

is interesting and want to find out how it was coded in HTML, just follow

the steps below. This will show you the source code for the entire page.

Go to the View menu

Choose Page Source or Document Source

Copying the Source:

Now you have found the source code of your favorite pageand would like

to keep a copy of it on disk. Follow the steps below:

- Go to the Filemenu and choose Save As
- Specify any valid file name and file extension of .html
- Specify any directory
- Specify the Format as HTML Filein “Save as Type”.

But Not Least

Now that you have created and testedyour HTML file, you are ready to upload the file 12

Last to your account on the web server. Put all your HTML files

in the same folder or directory on your PC or your MAC before uploading. Your personal Web pages will be uploaded to your personal OAK account. It is recommended that you nameyour personal home page: index.html.

You must first create a directory in your OAK account for your web files.

Creating a Directory on OAK:

Save your HTML file with an extension of .html. Then log on to OAK. The

following steps create the proper OAK directory for you, calleded:

public_html You still need to upload the files.

- choose - Change your Personal Information
- choose - Manage a Personal WWW directory
- choose - Create a Personal WWW directory
- choose - Enable access to Personal WWW directory
- Quit OAK

Uploading an HTML File:

At present, your HTML file can only beseen from your computer. In order for anyone on the Internet to see your file, you will have to upload it to the server where you have a personal account or departmental account. You can upload a file with the software listed below.

On a PCuse one of the following programs:

MS-KERMIT, CUTCPIP, WS_FTP

On a MACuse one of the following programs:

VersaTerm, MacKermit, FETCH

13

Uploading a Personal Page:

(From your PC to Oak using WS_FTP)

Select and start WS_FTP

for Host name type: oak.cats.ohiou.edu

for Userid type: your Oak userid

for Password type: your Oak password

Click OK

On the left side: double click on the proper directory

Select the proper file

Select: ASCII

On the right side: double click on the public_html directory

Click the Right arrow

Click Close

Uploading a Personal Page:

(From your MAC to Oak using FETCH)

Select and start FETCH

for Host name type: oak.cats.ohiou.edu

for Userid type: your Oak userid

for Password type: your Oak password

the Directory box: should be empty

Click OK

In FETCH: click on the Public_Html directory

Click Put File

In the next box, enter the File Name to be uploaded

In the next box, enter the File Name for Oak

Set "Format" to Text for text files; and BixHex or Raw Data for other files

Click OK The URLof your personal web page:

http://oak.cats.ohiou.edu/~userid/filename

CGI Scripts:

Introduction to CGI

Common Gateway Interface (CGI) is a standard for interfacing external programs with information servers on the Internet. So what does this mean? Basically, CGI is distinguished from a plain HTML document in that the plain HTML document is static, while CGI executes in real-time to output dynamicinformation. A program that implements CGI is executable, while the plain HTML document exists as a constant text file that doesn't change. CGI, then, obtains information from users and tailors pages to their needs. While there are newer ways to perform the same kinds of actions that traditionally have been implemented with CGI, the latter is older and, in many ways, more versatile. It is for this reason that, over time, CGI has become generalized to refer to any program that runs on a Web server and interacts with a browser.

For example, if you wanted to allow people from all over the world to query some database you had developed, you could create an executable CGI script that would transmit information to the database engine and then receive results and display them in the user's Web browser. The user could not directly access the database without some gateway to allow access. This link between the database and the user is the "gateway,"

which is where the CGI standard originated.

A CGI script can be written in any language that allows it to be executed (e.g., C/C++, Fortran, PERL, TCL, Visual Basic, AppleScript, Python, and Unix shells), but by far, the most common language for CGI scripting is PERL, followed by C/C++. A CGI script is easier to debug, modify, and maintain than the typical compiled program, so many people prefer CGI for this reason.

For tutorials on CGI, please see http://cgi.resourceindex.com/Documentation/. Significance The importance of CGI lies in the fact that its flexibility has made it a standard for running executable files from Web servers. This standard allows for true interactivity, in endless ways, on Web sites. For example, CGI scripts can implement the following kinds of features

Access Counters: Display the number of visitors to your site in a text-based or graphical manner

Advertisements: Set up banner rotations on your Web page and track their statistics

Auctions Provide for Web-based auctions

Audio Management:Provide and manage audio files in different formats for users to listen to

Bulletin Board Message Systems:Provide on-line message forums for threaded discussions

Calendars: Schedule events and/or allow users to post dated information

Chat:Provide for real-time chats on the Web

Classified Ads:Allow users to post information on buying, selling, and/or trading possessions

Clocks:Display the current time on your Web page in an image- or text-based format

Commerce and Finance:Allow users to use calculators, credit cards, etc. Content Retrieval Retrieve and integrate content of all kinds into your Web site (e.g., news and headlines, stock quotes, weather)

Cookies:Track visitors and store user information

Countdowns :Display on your Web page the length of time until a specified event **Customer Support**:Maintain knowledge bases; provide for customer support e-mail ticketing, real-time

customer support, and FAQ maintenance

Database Manipulation :Create, edit, and manipulate databases;

allow users to search your databases

Development Environments:Aid in the development of programs and promote collaboration

Editing Web Pages:Allow users or administrators to create and edit Web pages

File Management:Manage your files and directories via the Web (e.g., file downloading/uploading, link protection)

Form Processing: Process basic forms and send results via e- mail

Games:Allow users to play games on your Web site

Guestbooks:Allow visitors to sign in and leave a message on your Web site

HTML Manipulation: Create and insert tables, frames, lists, headers, and footers

Homepage Communities:Allow visitors to create their own customized Web pages on your site

Image Display: Display images in different ways and formats; index images; allow for picture posting; timed rotation of images

Imagemaps:Create clickable images that redirect the user to another page

Instant Messaging :Allow users to participate in instant messaging or to use various features of instant messaging systems

Interactive Stories:Allow visitors to add to an existing story

Internet Utilities:Provide for a wide range of standard Internet programs (e.g., finger, telnet, traceroute, whois)

Link Indexing Scripts:Allow visitors to add links to your Web site

Link Verification:Test the links on your page; evaluate your server and its performance

Logging Accesses and Statistics: Track the number of visitors to your site; log useful statistics regarding your site and visitors

Mailing Lists: Allow for mailing lists and management of existing mailing lists

News Posting: Post and manage news and updates to your site

Password Protection:Password protect your Web site

Postcards :Allow visitors to your Web page to send an

Internet postcard to someone

Random Items: Include random links, text, images, pages, etc. on your Web site

Redirection: Redirect users in various ways (e.g., jump boxes, browser-based, error-based)

Reservations and Scheduling: Make reservations and usage schedules for items, people, etc.

Searching:Allow users to perform searches on your Web site using keywords and phrases

Shopping Carts:Set up an on-line store using shopping cart and catalog features

Spam Prevention:Randomly generate fake (but real-looking) e-mail addresses for spammer robots to pick up, resulting in a bunch of bounced spam for the spammer and less spam for you and your users

Surveys and Voting:Conduct surveys, take votes, allow users to post ratings and reviews

Tests and Quizzes:Create and grade tests and quizzes automatically over the Web

Web Server Maintenance:Maintain and monitor your server

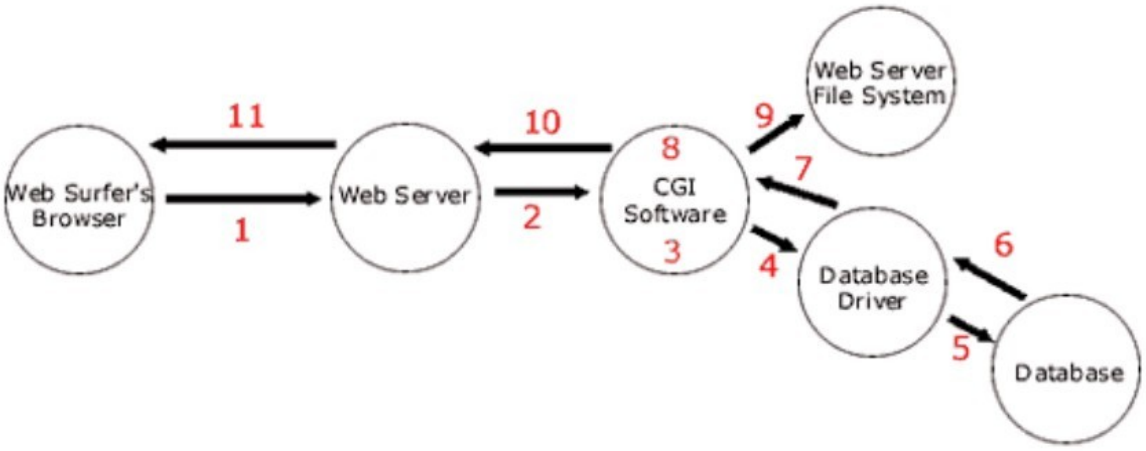
Web-Based E-Mail:Supply e-mail access to users through their Web browser

Website Promotion:Set up affiliate programs for sites referring visitors to you; set up contests/awards to attract visitors to your site; set up a link exchange between sites with banner ads; allow users to recommend your Web site to a friend; submit your URL to search engines; track referrers; create your own ring of Web sites.

How CGI Scripts Work

Http://www.linkyours.com/cgi_overview.htmlprovides an excellent overview

(reproduced here with permission) of the process by which CGI scripts are executed:



1. The Web surfer fills out a form and clicks, “Submit.” The information in the form is sent over the Internet to the Web server.

2. The Web server “grabs” the information from the form and passes it to the CGI software.

3. The CGI software performs whatever validation of this information that is required. For instance, it might check to see if an e-mail address is valid. If this

is a database program, the CGI software prepares a database statement to either add, edit, or delete information from the database.

4. The CGI software then executes the prepared database statement, which is passed to the database driver.

5. The database driver acts as a middleman and performs the requested actionon the database itself.

6. The results of the database action are then passed back to the database driver.

7. The database driver sends the information from the database to the CGI software.

8. The CGI software takes the information from the database and manipulates it into the format that is desired.

9. If any static HTML pages need to be created, the CGI program accesses the Web server computer’s file system and reads, writes, and/or edits files.

10. The CGI software then sends the result it wants the Web surfer’s browser to see back to the Web server.

11. The Web server sends the result it got from the CGI software back to the Web surfer’s browser.

Implementing CGI Scripts

In general, there are several steps necessary for installing CGI scripts that are the same regardless of the kind of script you are trying to install. Since I am assuming that the reader is a novice, the following instructions describe only how to installpre-made CGIs.

You must also know the path to the compiler on your host server. For instance, if you are using Perl, the path to the Perl compiler on your Web host would look something like

“#!/usr/bin/perl” or “#!/usr/local/bin/perl”. This path will always be the first line in your

script. If you do not know the path to the compiler on your server, you can find out by

using telnet to get to your server and then using a “whereis” command (e.g., “whereis

perl), or by simply asking your host provider for the information. Similarly, you must

know the path to your site from your server. It should look something like this:

“/usr/local/etc/httpd/sites/mysite.com”. Depending on the type of CGI script you are

trying to implement, you may also need to know the path to your server’s mail program

or your cgi-bin. You can get this information by checking the CGI help section of your

host provider’s Web site or by asking them directly. Once you have obtained all the

necessary path information, you can simply use a text editor to change the CGI script to

reflect your own path information.

Finally, you will need to upload the script to your server. Conceptually, this means using an FTP program to upload the script to your cgi-bin directory. The default on your FTP program, however, will be to upload files in binarymode. While this is acceptable for most files, CGI files mustbe uploaded in ASCII format. Otherwise, they will not work.

Remember, then, to always switch to ASCII mode before uploading CGI scripts.

After uploading the CGI script, it is important to set permission for it (see the discussion of security issues for CGI below). This tells your server who can read, write (i.e., modify), and execute your script. First check the documentation inside the script to find out whether you should “chmod 755” or “chmod 777” it. “Chmod” refers to the “change mode” command, and the numbers simply designate two different types of permission settings (for a more thorough discussion of these settings, please refer to

http://jgo.local.net/LinuxGuide/linux-chmod.html). Once you have determined the

proper permission setting for your script, permission can be set one of three ways: (1)

using your FTP program, (2) by telnetting to your server and using the “chmod”

command, or (3) by asking the support personnel at your Web host to change permission

settings for you. If you wanted to change permission settings using the FTP program,

WSFTP, for instance, you could do this by right-clicking on the script and selecting

“chmod.” This will bring up a “Remote file permissions” box with three users for whom you must set permission (i.e., owner, group, and other) and three permission levels (i.e.,

read, write, and execute). If your script designates a chmod 755 permission level, you will allow the owner all three permission levels, and allow “group” and “other” read and execute permissions, but not write permission. Basically, this means that you are allowing users a read-only version of your file. They will be allowed to execute the

script, but not to change it. If your script designates a chmod 777 permission level, you allow all users all levels of permission. Http://jgo.local.net/LinuxGuide/linuxchmod.html, however, warns of the dangers with allowing chmod 777 permission levels on a CGI script, as this basically “allows the world to replace the program with whatever

they’d like.”

An Example

The following CGI script was created using Perl. After users vote for their favorite

names (another CGI script), this script tabulates votes to create the table of output

following the script:

```
#!/usr/local/bin/perl

print "Content-type:text/html\n\n";

open(INF,"votes.out");

@NAMES = <INF>;

close(INF);

foreach $line (@NAMES)

{

$linecount++;
```

```
@values = split(/\|/, $line);
foreach $value (@values)
{
$FORM{$value}++;
}
}
print "<html><head><title>Current Results</title></head>\n";
print "<BODY BGCOLOR=\"#AABBBB\" TEXT=BLACK LINK=\"#001170\"
VLINK=\"#001170\" ALINK=\"#001170\">\n";
print "<h2>Current Results</h2><BR><BR>\n";
print "<TABLE WIDTH=400 BORDER=1 CELLSPACING=0 CELLPADDING=0
BGCOLOR=\"#779E9E\">";
print "<TR><TD colspan=1 align=center><h3>Boy Names:</h3></TD>";
print "<TD colspan=1 align=center><h3>Girl Names:</h3></TD>";
print "<TR BGCOLOR=\"#88AFAF\"><TD><TABLE WIDTH=200 BORDER=0
CELLSPACING=0 CELLPADDING=0";
print " BGCOLOR=\"#779E9E\">\n";
@boynames = ("boyname a","boyname b","boyname c");
foreach $x (@boynames) {
{
 print "<TR BGCOLOR=\"#88AFAF\"><TD WIDTH=33%> </TD>";  print "<TD
WIDTH=30%>$x</TD><TD
WIDTH=4%><B>$FORM{$x} </B></TD>";
print "<TD WIDTH=33%> </TD></TR>\n";
}
}
print "</TABLE></TD><TD>";
print "<TABLE WIDTH=200 BORDER=0 CELLSPACING=0 CELLPADDING=0";
```

```
print " BGCOLOR=\"#779E9E\">\n";

@girlnames = ("girlname a", "girlname b", "girlname c");

foreach $x (@girlnames) {

{

 print "<TR BGCOLOR=\"#88AFAF\"><TD WIDTH=33%> </TD>";   print "<TD
WIDTH=30%>$x</TD><TD  WIDTH=4%><B>$FORM{$x} </B></TD>";

 print "<TD WIDTH=33%> </TD></TR>\n";  }

}

print "</TABLE></TD></TR>";

print "<TR><TD COLSPAN=2 BGCOLOR=\"#779E9E\"

ALIGN=RIGHT><B>$linecount</B> people have voted.</TD></TR>";

print "</TABLE>";

print "<BR><BR><BR>\n";

print "<A HREF=\"http://www.beth.cx/baby/\">Go back to Beth v2.0</A>\n";

print "</DIV>\n";

print "</BODY></HTML>\n";
```

Current Results

Boy Names: Girl Names:

boyname a 6

boyname b 11

boyname c 1

girlname a 6

girlname b 7

girlname c 7

20people have voted.

Why CGI?

The above applications can be implemented using other means as well (e.g., server-side JavaScript, PHP, ACGI, VRML, DHTML), but many of these other means developed after CGI. CGI, then, has become a standard, and many programmers

prefer simply to “tweak” their old CGI scripts for new purposes, instead of starting from scratch with the newer languages. Also, CGI is more versatile in many ways. A traditional CGI application using Perl, for instance, can be run on a large number of platforms with a wide variety of Web servers. A programmer using server-side JavaScript, however, would be limited to Netscape Enterprise Server. CGI has its disadvantages though. Many of the newer languages developed in response to CGI being slow, so they are significantly faster. An informal study of the performance of CGI versus server-side JavaScript, for instance, found the following discrepancy in

access times for two Web pages:

Category page Product page

CGI/Perl 219 ms 5990 ms

LiveWire/JavaScript 198 ms 104 ms

Also, there are significant security issues with CGI. Since a file that uses CGI is executable, it is equivalent to letting anyone in the world run a program on your machine. Obviously, this is not the safest thing to do. For this reason, many Web hosts do not allow users to run CGI scripts. In this case, though, you can have your CGI applications hosted for you remotely. Http://www.hypermart.netis an almost-free host that allows

CGI scripting, and http://cgi.resourceindex.com/Remotely_Hosted/lists a number of other hosts that allow CGI. Related is the fact that programs that use CGI scripts need to reside in a special directory,

so that the server knows to execute the program rather than simply display it to the browser. This directory, commonly /cgi-bin, is under the direct control of the

Webmaster. This prohibits the average user from creating and running programs that use

CGI.

Summary

CGI is a standard for interfacing executable files with Web servers. It allows for the interactive, dynamic, flexible features that have become standard on many Web sites, such as guestbooks, counters, bulletin boards, chats, mailing lists, searches, shopping carts, surveys, and quizzes. Several newer, faster means for accomplishing these same

kinds of tasks have been developed, but CGI is more flexible in a number of ways. CGI is commonly used whenever one needs a Web server to run a program in

real-time, take some kind of action, and then send the results back to a user's browser. Scripts can be written in any language that allows a file to be executed, but the most common language for CGI scripts is Perl. One does not have to be a programmer to use CGI scripts (although this helps!), as there are a number of sites that offer free, "canned" scripts that can be modified with the installer's personal server path information.

MODULE II

DESIGNING WEB APPLICATION:

JAVA - APPLET BASICS:

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are some important dif f erences between an applet and a standalone Javaapplication, including the following:

An applet is a Java class that extends the java.applet.Applet class.

A main() method is not invoked on an applet, and an applet class will not def ine

main().

Applets are designed to be embedded within an HTML page.

When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.

The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lif etime.

Applets have strict security rules that are enforced by the Web browser. The security of an applet is of ten ref erred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

Other classes that the applet needs can be downloaded in a single Java Archive

(JAR) f ile.

Life Cycle of an Applet:

Four methods in the Applet class give you the f ramework on which you build any serious applet:

init: This method is intended for whatever initialization is needed for your applet. It is called af ter the param tags inside the applet tag have been processed.

start: This method is automatically called af ter the browser calls the init method. It is also called whenever the user returns to the page containing the applet af ter having gone of f to other pages.

stop: This method is automatically called when the user moves of f the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

destroy: This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind af ter a user leaves the page that contains the applet.

paint: Invoked immediately af ter the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited f rom the java.awt.

A "Hello, World" Applet:

The following is a simple applet named HelloWorldApplet. java:

```
i mport  java.applet.*;

i mport  java.awt.*;

publ i c  class Hel loWorldApplet  extends  Applet

{

publ i c  voi d  pai nt  (Graphi cs  g)

{

g.drawStri ng  ("Hel lo World" ,  25,  50);

}
```

}

These import statements bring the classes into the scope of our applet class:

java.applet.Applet.

java.awt.Graphics.

Without those import statements, the Java compiler would not recognize the classes

Applet and Graphics, which the applet class ref ers to.

The Applet CLASS:

Every applet is an extension of the java.applet.Applet class. The base Applet class provides methods that a derived Applet class may call to obtain information and services f rom the browser context.

These include methods that do the following:

Get applet parameters

Get the network location of the HTML f ile that contains the applet

Get the network location of the applet class directory

Print a status message in the browser

Fetch an image

Fetch an audio clip

Play an audio clip

Resize the applet

Additionally,the Applet class provides an interf ace by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may:

request information about the author, version and copyright of the applet

request a description of the parameters the applet recognizes

initialize the applet

destroy the applet

start the applet's execution

stop the applet's execution

The Applet class provides def ault implementations of each of these methods. Those implementations may be overridden as necessary.

The "Hello, World" applet is complete as it stands. The only method overridden is the paint method.

Invoking an Applet:

An applet may be invoked by embedding directives in an HTML f ile and viewing the f ile through an applet viewer or Java-enabled browser.

The <applet> tag is the basis for embedding an applet in an HTML f ile.

Below is an example that invokes the "Hello, World" applet:

<html>

<ti tle>The Hel lo, World Applet</ ti tle>

<hr>

<applet code="Hel loWorldApplet.class" wi dth=" 320" hei ght=" 120" >

If your browser was Java-enabled, a "Hel lo, World"

message would appear here.

</applet>

<hr>

</html>

Based on the above examples, here is the live applet example: Applet Example.

Note: You can ref er to HTML Applet Tag to understand more about calling applet f rom HTML.

The code attribute of the <applet> tag is required. It specif ies the Applet class to run. Width and height are also required to specif y the initial size of the panel in which an applet runs. The applet directive must be closed with a </applet> tag.

If an applet takes parameters, values may be passed for the parameters by adding

<param> tags between <applet> and </applet>. The browser ignores text and other tags between the applet tags.

Non-Java-enabled browsers do not process <applet> and </applet>. Therefore, anything that appears between the tags, not related to the applet, is visible in nonJava-enabled browsers.

The viewer or browser looks for the compiled Java code at the location of the document. To specif y otherwise, use the codebase attribute of the <applet> tag as shown:

<applet codebase=" http:/ /amrood.com/applets"

code="Hel loWorldApplet.class" wi dth=" 320" hei ght=" 120" >

If an applet resides in a package other than the def ault, the holding package must be

specif ied in the code attribute using the period character (.) to separate package/class

components. For example:

<applet code="mypackage.subpackage.TestApplet.class"

wi dth=" 320" hei ght=" 120" >

Getting Applet Parameters:

The following example demonstrates how to make an applet respond to setup parameters specif ied in the document. This applet displays a checkerboard pattern of black and a second color. The second color and the size of each square may be specif ied as parameters to the applet within the document. CheckerApplet gets its parameters in the init() method. It may also get its parameters in the paint() method. However, getting the values and saving the settings once at the start of the applet, instead of at every ref resh, is convenient and ef f icient. The applet viewer or browser calls the init() method of each applet it runs. The viewer calls init() once, immediately af ter loading the applet. (Applet.init() is implemented to do nothing.) Override the def ault implementation to insert custom initialization code.

The Applet.getParameter() method f etches a parameter given the parameter's name (the value of a parameter is always a string). If the value is numeric or other noncharacter data, the string must be parsed.

The following is a skeleton of CheckerApplet. java:

i mport java.applet.*;

i mport java.awt.*;

publ i c class CheckerApplet extends Applet

```
{
i nt  squareSi ze  =  50;/ /   i ni ti al i zed  to  default  si ze
publ i c  voi d  i ni t  ()  {}
pri vate  voi d  parseSquareSi ze  (Stri ng  param)  {}
pri vate  Color  parseColor  (Stri ng  param)  {}
publ i c  voi d  pai nt  (Graphi cs  g)  {}
}
```

Here are CheckerApplet's init() and private parseSquareSize() methods:

```
publ i c  voi d  i ni t  ()
{
Stri ng  squareSi zeParam  =  getParameter  (" squareSi ze" );
parseSquareSi ze  (squareSi zeParam);
Stri ng  colorParam  =  getParameter  (" color" );
Color  fg  =  parseColor  (colorParam);
setBackground  (Color.black);
setForeground  (fg);
}
pri vate  voi d  parseSquareSi ze  (Stri ng  param)
{
i f  (param  ==  nul l)  return;
try  {
squareSi ze  =  Integer.parseInt  (param);
}
catch  (Excepti on  e)  {
/ /   Let  default  value  remai n
}
}
```

The applet calls parseSquareSize() to parse the squareSize parameter.

parseSquareSize() calls the library method Integer.parseInt(), which parses a string and returns an integer. Integer.parseInt() throws an exception whenever its argument is invalid. Therefore, parseSquareSize() catches exceptions, rather than allowing the applet to f ail on bad input. The applet calls parseColor() to parse the color parameter into a Color value. parseColor() does a series of string comparisons to match the parameter value to the name of a predef ined color. You need to implement these methods to make this applet works.

Specifying Applet Parameters:

The following is an example of an HTML f ile with a CheckerApplet embedded in it. The HTML f ile specif ies both parameters to the applet by means of the <param> tag.

<html>

<ti tle>Checkerboard Applet</ ti tle>

<hr>

<applet code=" CheckerApplet.class" wi dth=" 480" hei ght=" 320" >

<param name=" color" value=" blue" >

<param name=" squaresi ze" value=" 30" >

</applet>

<hr>

</html>

Note: Parameter names are not case sensitive.

Application Conversion to Applets:

It is easy to convert a graphical Java application (that is, an application that uses the AWT and that you can start with the java program launcher) into an applet that you can embed in a web page.

Here are the specif ic steps for converting an application to an applet.

Make an HTML page with the appropriate tag to load the applet code.

Supply a subclass of the JApplet class. Make this class public. Otherwise, the applet cannot be loaded.

Eliminate the main method in the application. Do not construct a f rame window for the application. Your application will be displayed inside the browser.

Move any initialization code f rom the f rame window constructor to the init method of the applet. You don't need to explicitly construct the applet object.the browser instantiates it for you and calls the init method.

Remove the call to setSize; for applets, sizing is done with the width and height parameters in the HTML f ile.

Remove the call to setDef aultCloseOperation. An applet cannot be closed; it terminates when the browser exits.

If the application calls setTitle, eliminate the call to the method. Applets cannot have title bars. (You can, of course, title the web page itself , using the HTML title tag.) Don't call setVisible(true). The applet is displayed automatically.

Event Handling:

Applets inherit a group of event-handling methods f rom the Container class. The Container class def ines several methods, such as processKeyEvent and processMouseEvent, for handling particular types of events, and then one catch-all method called processEvent. In order to react an event, an applet must override the appropriate event-specif ic method.

```
i mport  java.awt.event.MouseLi stener;

i mport  java.awt.event.MouseEvent;

i mport  java.applet.Applet;

i mport  java.awt.Graphi cs;

publ i c  class  ExampleEventHandl i ng  extends  Applet

i mplements MouseLi stener  {

Stri ngBuffer  strBuffer;

publ i c  voi d  i ni t()  {

addMouseLi stener(thi s);

strBuffer  =  new  Stri ngBuffer();

addItem(" i ni ti al i zi ng  the  apple  " );

}

publ i c  voi d  start()  {

addItem(" starti ng  the  applet  " );

}
```

```
publ i c  voi d  stop()  {
addItem(" stoppi ng  the  applet  " );
}
publ i c  voi d  destroy()  {
addItem(" unloadi ng  the  applet" );
}
voi d  addItem(Stri ng word)  {
System.out.pri ntln(word);
strBuffer.append(word);
repai nt();
}
publ i c  voi d  pai nt(Graphi cs  g)  {
/ /Draw  a  Rectangle  around  the  applet's  di splay  area.
g.drawRect(0,  0,
getWi dth()  -  1,
getHei ght()  -  1);
/ /di splay  the  stri ng  i nsi de  the  rectangle.
g.drawStri ng(strBuffer.toStri ng(),  10,  20);
}
publ i c  voi d mouseEntered(MouseEvent  event)  {
}
publ i c  voi d mouseExi ted(MouseEvent  event)  {
}
publ i c  voi d mousePressed(MouseEvent  event)  {
}
publ i c  voi d mouseReleased(MouseEvent  event)  {
}
publ i c  voi d mouseCl i cked(MouseEvent  event)  {
```

```
addItem("mouse  cl i cked!  " );
}
}
```

Now, let us call this applet as follows:

```
<html>
<ti tle>Event Handl i ng</ ti tle>
<hr>
<applet  code=" ExampleEventHandl i ng.class"
wi dth=" 300"   hei ght=" 300" >
</applet>
<hr>
</html>
```

Initially, the applet will display "initializing the applet. Starting the applet." Then

once you click inside the rectangle "mouse clicked" will be displayed as well. Based on the above examples, here is the live applet example: Applet Example.

Displaying Images:

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, you use the drawImage() method found in the java.awt.Graphics class.

Following is the example showing all the steps to show images:

```
i mport  java.applet.*;
i mport  java.awt.*;
i mport  java.net.*;
publ i c  class  ImageDemo  extends  Applet
{
pri vate  Image  i mage;
pri vate  AppletContext  context;
publ i c  voi d  i ni t()
{
```

```
context  =  thi s.getAppletContext();
Stri ng  i mageURL  =  thi s.getParameter(" i mage" );
i f(i mageURL  ==  nul l)
{
i mageURL  =  " java.jpg" ;
}
try
{
URL  url   =  new URL(thi s.getDocumentBase(),  i mageURL);
i mage  =  context.getImage(url);
}catch(Mal formedURLExcepti on  e)
{
e.pri ntStackTrace();
/ /  Di splay  i n  browser  status  bar
context.showStatus(" Could  not  load  i mage!" );
}
}
publ i c  voi d  pai nt(Graphi cs  g)
{
context.showStatus("Di splayi ng  i mage" );
g.drawImage(i mage,  0,  0,  200,  84,  nul l);
g.drawStri ng("www.javal i cense.com" ,  35,  100);
}
}
```

Now, let us call this applet as follows:

```
<html>
<ti tle>The  ImageDemo  applet</ ti tle>
<hr>
```

<applet code=" ImageDemo.class" wi dth=" 300" hei ght=" 200" >

<param name=" i mage" value=" java.jpg" >

</applet>

<hr>

</html>

Based on the above examples, here is the live applet example: Applet Example.

Playing Audio:

An applet can play an audio f ile represented by the AudioClip interf ace in the java.applet package. The AudioClip interf ace has three methods, including:

public void play(): Plays the audio clip one time, f rom the beginning.

public void loop(): Causes the audio clip to replay continually.

public void stop(): Stops playing the audio clip.

To obtain an AudioClip object, you must invoke the getAudioClip() method of the Applet class. The getAudioClip() method returns immediately, whether or not the URL resolves to an actual audio f ile. The audio f ile is not downloaded until an attempt is made to play the audio clip.

Following is the example showing all the steps to play an audio:

```
i mport java.applet.*;
i mport java.awt.*;
i mport java.net.*;
publ i c class Audi oDemo extends Applet
{
pri vate Audi oCl i p cl i p;
pri vate AppletContext context;
publ i c voi d i ni t()
{
context = thi s.getAppletContext();
Stri ng audi oURL = thi s.getParameter(" audi o" );
i f(audi oURL == nul l)
```

```
{
audi oURL  =  " default.au" ;
}
try
{
URL  url   =  new URL(thi s.getDocumentBase(),  audi oURL);
cl i p  =  context.getAudi oCl i p(url);
}catch(Mal formedURLExcepti on  e)
{
e.pri ntStackTrace();
context.showStatus(" Could  not  load  audi o  fi le!" );
}
}
publ i c  voi d  start()
{
i f(cl i p  !=  nul l)
{
cl i p.loop();
}
}
publ i c  voi d  stop()
{
i f(cl i p  !=  nul l)
{
cl i p.stop();
}
}
}
```

Now, let us call this applet as follows:

<html>

<ti tle>The ImageDemo applet</ ti tle>

<hr>

<applet code=" ImageDemo.class" wi dth=" 0" hei ght=" 0" >

<param name=" audi o" value=" test.wav" >

</applet>

<hr>

</html>

You can use your test.wav at your PC to test the above example.

JAVA SCRIPTS:

Overview

JavaScript is a full weight programming language that can be combined with HTML to expand its functionality.

JavaScript is interpreted; it is not compiled.

Client-side code is included in web pages, providing dynamic capabilities.

Server-side JavaScript is embedded in Netscape's web servers. It is untyped, meaning variables do not have to have a data type specified.

W3C document object model (DOM) elements can be accessed and manipulated with JavaScript.

The language also has: variables and literals, conditions, flow control (aka looping),

operators and expressions, functions.

Specification: JavaScript was originally developed by Brendan Eich of Netscape under the name Mocha, later LiveScript, and finally renamed to JavaScript. Netscape delivered

JavaScript to Ecma International

for standardization and the work on the specification,

ECMA-262, began in November 1996. Four versions of ECMA-262 were published, most

recently in December 2009. The version commonly used today is #3 from December 1999, it

is available at

http://www.ecma-international.org/publications/standards/ecma-262.htm . JavaScript is

considered a dialect of ECMAScript. JavaScript includes extensions like the W3C-specified

DOM (document object model), which is an API for HTML and XML documents; the DOM

enables it to access web document elements. Web-browser implementation is not fully compliant with the specifications. No surprise here, as the lack of compliance also holds for HTML and CSS.

What can you do with JavaScript

create content

change content

restyle content

handle events

change properties of the window

interact with HTML forms

General

case sensitive

white space is ignored

continue-break a code line with a backslash:

document.write(“hello \ world”)

NOT document.write \ (“hello”)

start comments on single line with “//”: sum=a + b //explanation

surround multi-line comment:

/* line one

line two

line three */

The name Ecma is, since 1994, no longer considered an acronym and no longer uses full

capitalization. Ecma is an international, private (membership-based) non-profit standards

organization for information and communication systems.

to put 2 or more statements on same line, separate each with semicolon

when nesting quoted strings, alternate double quotes with single quotes; for example, to indicate a quoted string inside a string literal, use single quotation marks:

document.write("<HR ALIGN='left' WIDTH=50>"

<INPUT TYPE="button" VALUE="Here" onclick="myfunc('astring')">

Variables

Names must begin with letter or underscore: hh, _ww

Declare variables: var hh="Susan", _idx=5

Variables declared within a function are local to the function.

Variables declared outside a function are global to the page.

Refer to an array entry: ArrayName[index]

Variables are only accessible in the page in which they are created. A variable created in index.html cannot be accessed by about.html which was opened by index.html in a separate

window.

Executing Javascript

How to run JavaScript

Code can be run as:

an event handler

automatically when the page loads (not the load event), typically used to generate content

a hypertext link when it is activated

Declare JavaScript in META tag

<META http-equiv="Content-Script-Type" content="text/javascript">

JavaScript can be located in external file, access it with link

External JavaScript file has no HTML. The use of the LANGUAGE attribute is necessary in IE6.

<SCRIPT TYPE=”text/javascript” SRC=”x.js”></SCRIPT>

<SCRIPT TYPE=”text/javascript” SRC=”x.js” LANGUAGE=”javascript”></SCRIPT>

Embed JavaScript within HTML

JavaScript code can be embedded within HTML in the HEAD section and/or the BODY section.

Code in the HEAD section must be explicitly run; it is loaded before the BODY section. Code in the BODY section is run as the page is loaded.

Embed JavaScript in BODY if it generates content of page Place JavaScript after the HTML elements that it references. For example, the code to hide a

section of text must follow that text.

<SCRIPT TYPE=”text/javascript”>

<!-- begin to hide script from old browsers

document.open(“log.html”)

function showlog()

{window.open(“log.html”)}

function docstatus()

{window.status=”message”}

// end hide -->

</SCRIPT>

Embed JavaScript statement or function in HTML statement as a Link object

The code is run when the link is activated.

<A HREF=”javascript: statement”> . . . </A>

<A HREF=”javascript: xyz()”>slower</A>

Invoking an event handler

An event handler must be “registered” in order to be run or invoked. There are three registration

methods: by attribute, by property, and advanced. The first two methods are defined in the Level

0 API, the third is defined in the Level 2 DOM event model.

Event registration by HTML attribute. In this method the event handler is specified in HTML as

an attribute, and applies only to the particular instance of the element. Examples:

<BODY onload="setfocus()"

<A HREF="somewhere.html" onClick="alert('I\'ve been clicked!')">

<BODY onload="window.open('log.html')">

<INPUT TYPE="button" VALUE="try" onClick="statement 1;statement 2">

If “x” is a function it must be defined in the HEAD section or in a linked script file. In this method, the event handler is run before the default action (the link). If the event handler returns a false, the default action is not taken. Not all default actions can be prevented, e.g., unload. This method is not recommended for XHTML because it requires you to write JavaScript behavior code in your XHTML structure layer, where it doesn’t belong.

A limitation of this approach is that when the event handler is specified as a function, it can have no arguments.

If the function uses arguments, it must be invoked as a variable:

```
var varxyz = new Function("a", "b", "return a * b")
. . .
<input name="operand1" type="text" value="5"> First number
<input name="operand2" type="text" value="6"> Second number
<input name="button1" type="button" value="Multiply"
onclick="document.form1.result.value=varxyz(document.form1.operand1.value,
document.form1.operand2.value)"> Result here: 
<input name="result" type="text" value="">
```

Event registration by JavaScript property. In this method, the event handler is specified solely in JavaScript, not in HTML, and applies to all instances of the particular HTML element; it is specified as an element property. Examples:

```
<DIV ID="menu1"> . . .
document.getElementById("menu1").onmouseover = x
```

```
element.onclick = doSomething
```

applies to all instances of <element>.

The event handler can be removed:

```
element.onclick = null;
```

```
function doSomething() { this.style.backgroundColor = '#cc0000'; }
```

The element gets a red background whenever the user clicks on it.

In this method you can specify only one function as the event handler. Should you want to run more than one function, specify them as an anonymous function composed of several named

functions:

```
element.onclick = function () {startDragDrop(); spyOnUser()}
```

As with the inline registration method, one limitation of this approach is that when the event handler is specified as a function, it can have no arguments.

Another limitation of this approach is that if the user interacts with a document element before the document is fully loaded (and before all its scripts have executed), the event handlers for the document element may not yet be defined.

Advanced event registration. There are two versions, W3C and Microsoft IE. While the W3C version is considered better, few browsers support it. I avoid it bcause of the cross-browser inconsistencies.

W3C's DOM Level 2 Event specification offers a simple way to register as many event handlers as you like for the same event on one element: with the method addEventListener(). element.addEventListener('click', doSomething, false)

We can add as many event listeners as we want to the element, however the model does not state

which event handler is fired first:

element.addEventListener('click', startDragDrop, false)

element.addEventListener('click', spyOnUser, false)

To remove an event handler, use the removeEventListener() method:

element.removeEventListener('click', spyOnUser, false)

One problem with the current implementation of W3C's event registration model is that you can't find out if any event handlers are already registered to an element. However, because removeEventListener() doesn't give any errors if the event listener you want to remove has not

been added to the element, using it has no penalty.

Microsoft's IE model is similar. Instead of adding and removing event listeners, it attaches and

detaches events:

element.attachEvent('onclick', doSomething)

element.detachEvent('onclick', doSomething)

The major drawback to the Microsoft model is that the this keyword always refers to the window and is completely useless.

Use JavaScript expression as HTML attribute value

* for Netscape 4 only

<SCRIPT>

barwidth=50

</SCRIPT>

<HR WIDTH="&{barwidth};%" ALIGN=”LEFT>

Manipulate elements of a document

You can refer to a document element in different ways:

a) by its ID attribute

EX: <SPAN ID=a>

var xa = document.getElementById("a");

var yb = document.all["b"]; AVOID

b) by its NAME attribute

EX: <IMG NAME=b>

var xb = document.getElementsByName("b"); // returns an array

c) by its HTML tag name

EX: <IMG . . . >

var xc = document.getElementsByTagName("img")[0];

var allnodes = document.getElementsByTagName("*"); // returns an array

In the last example, variable allnodes is a NodeList object which behaves like an array. It contains all the nodes in the document in the order in which they appear in the HTML text (not the rendered page). Use its length property to access the number of nodes in the list.

Once you have a reference to a document element, you can:

ƒ change the text with document.createTextNode() and .replaceChild()

ƒ insert a node with document.createElement() and .appendChild()

ƒ delete a node

ƒ re-parent a node with .replaceCild() and .appendChild()

rearrange nodes

change CSS attributes

Refer to the DOM for the various nodes.

You can read/write HTML attributes:

var eid = document.getElementById(“x”);

eid.setAttribute(“class”, “wow”);

Text in an HTML document is represented in the DOM by a Text node (which is a special case of

CharacterData. This node has several methods which can be employed to change the text with

JavaScript code:

appendData(), deleteData(), InsertData(), replaceData(), and substringData(). The replaceData()

method replaces one string with another.

The node has two properties:

data: the text contained by the node.

length: the number of characters in the text.

However, text in a SPAN tag, in IE6, is an Element node, not a Text node. There is no method to replace the text in an Element node.

The HTMLElement superclass has an innerHTML property that provides read/write access to the

text contained in the element.

You can use JavaScript to create content with the document.write() method.

Apply style rules to elements

 …

. . .

var eid = document.getElementById(“x”);

eid.style.visibility = “hidden”;

eid.style.display = ”none”

Document Object Model (DOM)

“The Document Object Model is a platform- and language-neutral interface that will allow

programs and scripts to dynamically access and update the content, structure and style of documents.” per the W3C. It is an API.

Version history:

Level 1 was published October 1998.

Most of Level 2 was published November 2000.

Core Level 3 was published April 2004.

DOM conformance is incomplete. IE 6 is non-compliant in the use of node-type constants defined for the Node object; it does use the integer literals. At this time you should avoid features that are known or likely to not be implemented by the major browsers: all of Level 3 and much of Level 2.

arrays are zero-based: [0] is the first item.

Nodes

The DOM is a group of Node objects arranged in a tree structure. A pseudo array of all the nodes in a document can be accessed as in the following example:

var allnodes = document.getElementsByTagName(“*”);

Variable allnodes is a NodeList object (technically an interface) which behaves like an array. It contains all the nodes in the document in the order in which they appear. Use its length property to access the number of nodes in the list. The NodeList always and automatically reflects the current nodes; there is no need to refresh it with code.

Objects

array

boolean

function

string

number

error — instances are thrown as exceptions when runtime errors occur

date

window — self, window, parent, top

navigator

frames[ ]

location—properties refer to various portions of current document's URL

history[ ] — You can access the number of entries in the history list via the history.length

property

screen — info about display monitor

document

styleSheets[ ]

forms[ ]

elements[ ] HTMLElement is a superclass

anchors[ ]

links[ ] array of hypertext links

images[ ]

applets[ ]

embeds[ ]

all[ ] an Internet Explorer 4+ array of all HTML elements AVOID

elements[ ]

CSS2Properties

Refer to objects

object.property, also

object[“property”] and

object[varproperty]

the second form lets you access a property using a string

variable

object.array[index], also

object.array[“itemname”] and

object.array[varname]

the second form lets you access an array item using a

string variable

document.forms[0] first form in document

document.f1 named form. NOTE: In IE 6 you can refer to a form with

just the form name, but not in Netscape 7.

document.f1.elements[varname] named form element; uses associative array feature

document.f1.elements[6] seventh form element (array is zero-based)

document.f1.elements[i] variable i + 1 form element

document.soq.zipcode object that is form element named zipcode in form named

soq

document.soq.zipcode.name name of form element

document.soq.zipcode.value value of named element in named form

document.soq.biztype group of same-named elements, e.g., radio buttons

document.soq.biztype[i] individual element in a group of same-named elements

document.getElementById(“TOC”) element with named ID

document.getElementsByTag(“dd”)

[0]

first element with named tag

document.getElementsByTagName(

name)

returns a NodeList of elements with the given tag name in

the order they appear in the tree; “name” is a string

variable containing the tag name, the special string ″*″

represents all elements; CAUTION: this method of

referencing a document node is unreliable, use ID instead

document.getElementsByTagName(″

TD″).item(0)

first TD tag

document.getElementsByName(“spe

cial”)[i]

element with particular name attribute

document.elements[i] i + 1 element in HTML

frames[1].frames[2] third subframe in second main frame

parent.frame[1] second frame in a window referred to by first frame

parent.mfg when <FRAME name=mfg src=”mfg.html”> referred to

by sibling frame

this in a function, refers to the object that invoked the

function: o.m() . . . function m() {this.length . . . } “this”

refers to o.

this.form refers to the form object of a form element.

this.href refers to the document containing the link object

this.id ID attribute of “this” object

element.style.fontFamily = “Georgia” CSS property object

ƒ Reference only works after objects have been loaded. So best to put scripts just before

</BODY> tag.

Date methods

vardate = new Date() creates date object for today's date and time

vardate = new Date("December 25, 2003") creates date object dated 12-25-2003, time is zero

vardate = new Date("2003, 12, 25, 9, 30, 0) creates date object for 12-25-2003 09:30:00

setDay, setMonth, setFullYear, setTime

getDay, getMonth, getFullYear, getTime, getHours, getMinutes, getSeconds

to (returns string values from Date objects)

Document methods

close() getElementById() returns element which can then be manipulated

getElementByTagName() returns a NodeList array of all Element nodes with the specified

tag in the order in which they appear in the source open() write(text) append text

writeln(text) append text followed by newline character

Document properties

cookie domain lastModified location deprecated synonym for URL, but needed sometimes

URL excludes the ? part

referrer names the document containing the link that brought the browser to the

current document, if any

title offsetWidth width of the named element. example: document.body.offsetWidth is the

width of the browser window

Document events

onload

onunload

onbeforeload

Link object: a hypertext link

document.links[ ]

ƒ properties refer to part of the URL. For example, the URL =

“http://www.me.com:1234/here/there/us.html#we?x=y&a=b”.

ƒ properties are read/write.

hash = #we

host = www.me.com:1234

hostname = www.me.com

href = complete URL

pathname = /here/there/us.html

port = 1234

protocol = http:

search = ?x=y&a=b

target = name of a Window object

text = plain text, if any, between <a> and tags

Location properties

ƒ properties refer to part of the URL. For example, the URL =

“http://www.me.com:1234/here/there/us.html#we?x=y&a=b”.

hash = #we

host = www.me.com:1234

hostname = www.me.com

href = complete URL, including the ? part

pathname = /here/there/us.html

port = 1234

protocol = http:

search = ?x=y&a=b

Navigator properties

appCodeName code name of browser, e.g., “Mozilla”

appName name of browser, e.g., “Netscape”, “Microsoft Internet Explorer”

appVersion version and platform of browser, contents vary widely. Must be parsed

parseInt(appVersion) returns major version number

parseFloat(appVersion) returns major and minor version in floating-point

cookieEnabled true/false

platform e.g., “Win32”, “MacPPC”, “Linux i586”

Screen properties

These are from JavaScript 1.2. They provide information about the monitor display.

availHeight available height in pixels exclusive of task bar etc.

availWidth available width in pixels exclusive of task bar etc.

colorDepth base-2 logarithm of number of colors available as bits per pixel for each of

the three colors (red, green, and blue).

for 128 colors value is 7; for 16 colors value is 3;

value 32 yields 16,777,216 colors plus transparency (256 × 256 × 256)

“Truecolor”).

pixelDepth bits per pixel; unique to Netscape 4, undefined in IE 6

height total height of screen in pixels

width total width of screen in pixels

StyleSheet properties

document.styleSheets[0] accesses first stylesheet on the page; stylesheet can be external

defined with <LINK REL-“stylesheet” . . .> or embedded defined

within the paired <STYLE> tag.

href read-only; URL of external stylesheet

title read-only; title attribute if defined

Some people say you can immediately change a stylesheet with code:

document.styleSheets[0].href=”new.css”

I have done this with IE, but it fails under Mozilla

Window properties

frames an array of window objects, each of which is a frame contained within the

window. If window has no frames, the property is empty. Applies to both

<FRAME> and <IFRAME> tags.

name of the window, can be used as target of <A> tag

parent if current window is a frame, this is a reference to the frame of the window that contains it self refers to current window object top if window is in a frame, refers to the top-level window that contains the frame window same as self opener refers to Window object that opened this one, or “null” if opened by user location refers to the Location object Element style declarations

The syntax is like:

element.style.fontFamily = “sans-serif”

Where style is a property that returns a CSS2Properties object. This ability to test and set styles applies only to in-line styles. There is a JavaScript property for each CSS1 and CSS2 style atribute.

The names in JavaScript are the same as in CSS with the exception of CSS names with hyphens. In this case “camel case” is used. For example:

font-family fontFamily

border-top-color borderTopColor

There is no correspondence to CSS class and pseudo-class selectors. It is not possible to access them. It is not possible to refer directly to an element by its class name. If you need to do something like this, use the HTML name attribute with the function getElementsByName().

There is a way to read/write an element's class:

element.className = "special"

if (element.className == "special") element.className = element.className + "

temp" // adds a class to the element in order to invoke different styling

That said, some developers have created custom JavaScript libraries that mimic the things you

cannot do directly in basic JavaScript. See http://ajaxian.com/archives/javascript-css-selector-engine-timeline for a list of possibilities. HTML Document Object Model The HTML DOM API consists of nodes that correspond to the various HTML elements. These nodes exist in a hierarchical tree structure. There are different types of nodes. The Core DOM API includes interfaces that can apply to a HTML document including Node, Element, and

Document. HTMLDocument is an HTML-specific sub-interface of Document and HTMLElement is an HTML-specific sub-interface of Element. In addition the DOM defines tag-specific interfaces

for many HTML elements.

The nodes of an HTML document can be traversed recursively. Individual nodes can be accessed with document methods.

Functions

What a function can do

A function can

(a) perform one or more actions and stop

(b) perform a calculation and return the resulting value and stop

For case (b) the statement return is used to set the value and stop.

Basic syntax

A function has a name, arguments, and code. It is defined by the word “function.”

no arguments function abc() { . . . . }

one argument function abc(a) { . . . . }

four arguments function abc(a,b,c,d) { . . . . }

function variable is an

expression, no argument

var fv function { . . . . }

this defines an unamed function which is invoked:

y = fv

function variable is an

expression, one argument

var fx = function(x) { . . . . }

this defines an unamed function which is invoked:

y = fx(20)

A function stops after the last statement or after a return statement.

Functions can be recursive, i.e., they can call themselves.

Function arguments: While a function is declared with a fixed number of arguments, any number can be passed when the function is invoked. All arguments are accessible with the arguments[ ] pseudo-array, zero-based.

arguments.length provides the number of arguments. When 2 arguments are

provided, the value =2

A function can test its value to ensure the corrrect number of arguments were passed.

Individual arguments can be referred to in two ways. For a function abc(x)

(a) if x = 3

(b) arguments[0] = 5

The Arguments object has a property callee that refers to the function that is currently being

executed. This allows unnamed functions to invoke themselves recursively.

function check(args)

{

var actual = args.length; // the actual number of arguments

var expected = args.callee.length; // the expected number of arguments

if (actual != expected) { throw new Error("error message text"); }

{

Can a function be an argument of a second function? Yes.

Bad function names—don't use "test"

Use as re-usable block of code

Executed when function is called or triggered by an event (i.e., called by event handler)

Example for function named abc:

abc()

abc(x)

xyz(2,8)

<A ONCLICK="abc(x)"> . . . </A>

Assign a function to a variable

functionObjectName = new Function([arg1, arg2, ... argn], functionBody)

where: functionObjectName is a variable or object property or object event handler (e.g., window.onError); argx are arguments to be used by the function;

functionBody is string specifying JavaScript code to be evaluated as the function. example:

var setBGColor = new Function("document.bgColor='antiquewhite'")

The function is called: setBGColor() and by event handler: onClick="setBGColor()" and by event handler as: document.form(0).element(0).onClick=setBGColor

Function objects are evaluated each time they are used. This is less efficient than declaring a function because the latter is compiled. I wonder if the advantage isn't using a variable inside the functionBody, but cannot find an example.

Process Control

Conditional processing

if (expression)

statement;

if (expression) statement;

if (expression)

{ statement1;

statement2; }

if (expression) statement;

else statementb;

if (expression) statement;

else if (expression2)

statementb;

else

statementc;

switch (expression) {

case value1:

statement1; break;

case value2:

statement2; break;

default:

statement3; break; }

switch (x) {

case value1:

return y;

case value2:

return z; }

Looping

while (expression)

statement

while (expression) {

statement1;

statement2; }

do

statement

while (expression)

for (initialize; test; increment)

statement

example: for (var count=0; count < 10; count++)

  document.write(count + “
”);

The for statement(s) are done while the test is true.

for (variablein object)

statement;

Exit loop prematurely

only for: while, do/while, for, for/in.

In the version with labelname, a line break is not allowed before “labelname”.

break;

break labelname;

continue;

continue labelname;

labelname: statement;

Example:

loopa:

for . . .

{ . . .

break loopa;

. . . }

Expressions

Literals

numeric 1-7

string “fun”

boolean true

null null

array [2, 3, 5, 7]

variable I

Operators

array number [ ]

function call ()

increment value ++ (example: a = ++i;)

decrement value - - (two dashes)

logical complement ! (operand is boolean)

multiply *

division /

remainder %

add +

subtract -

equal ==

less than <

less than or equal <=

greater than >

greater than or equal >=

unequal !=

bitwise AND &

bitwise OR |

bitwise logical AND &&

bitwise logical OR ||

assignment =

concatenate +

Encoding special characters in URL strings and HTML text

The W3C specification for a URL includes: “Where the local naming scheme uses ASCII characters which are not allowed in the URI, these may be represented in the URL by a percent sign ‘%’ immediately followed by two hexadecimal digits (0–9, A–F) giving the ISO Latin 1 [ISO 8859-1] code for that character.”

In addition to URL strings, you may use JavaScript code to format text extracted from a database. That text might include characters that are special to HTML like “&”, “<”, “>” and paragraph breaks (possible in memo fields). The HTML BLOCKQUOTE tag can handle some but not all problems. The safest approach is to encode disallowed and HTML-reserved characters.

Common disallowed characters and their encoded values:

%20 is a space

%3B is a semicolon (;)

%21 is an exclamation point (!)

%26 is an ampersand (&)

Two functions address the encoding:

ƒ escape: encodes a string

ƒ unescape: decodes an encoded string unescape(encoded string)

searches for 2- and 4-digit hexidecimal escape sequences and replaces them in the string with their single character ISO Latin 1 equivalent.

Thus

document.write(unescape("Miss%20Piggy%20loves%20Kermit%21") yields

Miss Piggy loves Kermit!

Arrays

An array is akin to a table.

Arrays can be nested.

Array items are referenced with an index (x[i]) or by its itemname (x["itemname"]).

Arrays are zero-based: [0] is the first item.

Create an array: Create an array from list of values:

```
var mailbill = new Array("billstreet", "billcity", "billstate", "billzip");
```

ƒ Create an array and populate it with a function:

```
var geo = new Array(48);
initGeoArray(geo);
. . .
function initGeoArray(a)
{
a[0] = "allca";
a[1] = "alameda";
a[2] = "alpine";
```

. . .

}

ƒ An array item can contain an array.

How many items in an array

vararray.length

If an object is an array, its length property is “number”.

if (typeof(e.length) = “number”)

If an object is not an array, its length property is “undefined”.

if (typeof(e.length) = “undefined”)

Enumerate array items

ƒ Use the for/in loop:

var a = new Array();

. . . (populate array)

var i = 0;

for (i in a) alert(i);

Assign custom property to array

In this example, biztype is a group of same-named radio buttons.

document.soq.biztype.required = true;

then document.soq.biztype[i] is not required.

String Manipulation

Two objects:

String

ƒ RegExp

String concatenation is done with the + operator.

String properties

length returns integer

Example: var s = “abc”;

s.length // returns 3

String methods

charAt(n) extracts the character at a given position (n) from a string

indexOf(substring)

indexOf(substring, start)

searches the string for a character or substring; optionally, begin at a

numbered position; if none found, returns -1

lastIndexOf(substring)

lastIndexOf(substring,

start)

searches the string backwards for a character or substring

match(regexp) pattern matching with a regular expression

replace(regexp,

replacement)

search and replace with a regular expression

search(regexp) search a string for a substring that matches a regular expression

slice(start)

slice(start, end)

returns a substring in terms of its end position; if end not specified,

returns substring through end of string; “end” is position

immediately after the end of the slice

Example: var s = “abcdefg”;

s.slice(0,4) // returns “abcd”

s.slice(2,4) // returns “cd”

s.slice(4) // returns “efg”

toLowerCase() converts all characters to lower case

toUpperCase() converts all characters to upper case

Regular Expressions

Pattern matching involves Regular Expressions. A Regular Expression is an object, RegExp, that describes a pattern of characters. REs in JavaScript are strongly based on Perl regular expressions. REs have a complicated grammar are and discussed separately.

REs are composed of text between a pair of slashes. The second slash in the pair can be followed by one or more letters which serve to modify the meaning of the pattern.

Example of using regular expression:

```
var re = /^\d+$/;

if (!re.test(e.value))   // then there is an error

…

text.replace(/javascript/gi, “JavaScript”);
```

Special Techniques: Existence Determination

Determine if things exist

Determine if object exists: if (object_name == null) returns true when object does not exist.

Determine if method or property exists: if (!mywin.opener) returns true when the Window .

object mywin was opened by the user (and not another file/window). You can use this technique to test if the browser supports a JavaScript method or property. Example sets the value of an unsupported property:

```
if (!mywin.opener) mywin.opener = self //where mywin = window.open(file,

winname)
```

Determine if variable exists: don’t know.

Special Techniques: Windows

Coping with pop-up blockers

Windows XP SP2 includes the Information Bar which automatically blocks “pop-up” windows.

Microsoft defines these as any window opened automatically from script, with the exception of

createPopup(). Common functions that are affected are:

window.open()

showModelessDialog()

showModalDialog()

showHelp()

A window opened as a direct result of user action (e.g., clicking a page element) is not blocked.

You can tell if IE blocks a window: functions that return a window object will return null if the window is blocked. Always check the function's return value. Open different document in the current window

location.href="url"

One page can open a second page in another window

Use the "onload" attribute in the BODY tag to open a second window at the same time the first page is opened.

```
<HEAD>
<SCRIPT>
function showlog()
// "new" is the name of the window in which the second page is opened
{ window.open("log.html","new","toolbar=no,location=no,directories=no,\
status=no,menubar=yes,scrollbars=yes,resizable=no,copyhistory=no,width=600,\
height=400") }
</SCRIPT>
…
<BODY onload ="showlog()">
* * * * *
function smallwin(u)
{
//opens passed url in new small window
```

```
var winsize = "width=" + (.8 * screen.width) + ", height=" + (.6 *
screen.height) + ", left=80, top=80";
var controls = ", toolbar=yes, location=yes, menubar=yes, status=yes,
scrollbars=yes, resizable=yes";
window.open(u, "new", winsize + controls);
}
```


Open second page in new window. If window is already open give it the focus.

This code sometimes results in an error in IE 5.1, although still works; it seems that focus()

doesn’t work consistently.

```
function openwin(file, winname)
{
mywin = window.open(file, winname)
mywin.focus()
}}
```

Display a popup window with content and close it remotely

```
<SCRIPT LANGUAGE="JavaScript">
<!--
function launch() {
self.name = "index"; // name of current window
remote = open("path/filename.html", // name of file to be opened
"popup", // name of new window
"width=256,height=155,left=50,top=50") // size and location of new window
// the popup window can be closed with remote.close("") - I think
}
```

// -->
</SCRIPT>

** Close window that opened the current window - JavaScript
window.opener.close()
Open new window when current page is closed (by Back or new URL)
<body onUnload="open_windowx();">
Close current window
window.close()
Go to top
Go to top
A page can change its window size and position when it opens
window.resize(300,200)
window.moveTo(180,50)
self.moveBy (x,y) // x is the number of pixels on x axis, y on the y axis; can be negative
self.moveTo (xx,yy) // xx is number of pixels to right of left side, yy down from top
self.resizeBy(x,y) // x is number of pixels of width, y is height; can be negative
self.resizeTo(xx,yy) // xx is number of pixels of width, yy is height
windows can be no smaller than (100,100)
self.scrollBy(x,y)
self.scrollTo(xx,yy)
Open page in full size window
<script type="text/javascript">
<!--
function maxwin() {

```
if (window.screen) {
var aw = screen.availwidth;
var ah = screen.availheight;
window.moveTo(0,0);
window.resizeTo(aw, ah) }
}
//-->
</script>
</head>
```



```
<body onload="maxwin()">
```

OR

Note: setting window height to screen height will hide the status bar at the bottom.

```
function maxwin() {
window.moveTo(0,0)
window.resizeTo(screen.width, screen.height)
}
```

OR

```
window.open(filename, “”, “fullscreen, …”)
```

Open window, then write to it

Could be reused, with no separate HTML file

```
function openSmallWind(pic)
{
```

```
smwin = window.open(“”,”smallWin”, “dependent, resizable, …”)
var td = smwin.document
td.open()
td.write(‘<html><head><title>Small Window for Temporary
Display</title>,</head>’)
td.write(‘<body onLoad=”…”>’)
td.write(‘<img src=”pic.jpg”>’) // or td.write(‘<img src="' + 'Images/' + pic
+ '.jpg">’) where pic is a variable
td.write(‘</body></html>’)
td.close()
}
function closeIt() {
if (td != null && td.open) td.close()
}
window.onfocus=closeIt() //close small window when main window gets the
focus
// -->
</script>
```

Open popup window only once

Put this code in the HEAD section:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- This script and many more are available free online at -->
<!-- The JavaScript Source!! http://javascript.Internet.com -->
<!-- Begin
var expDays = 1; // number of days the cookie should last
var page = "http://www.slackerhtml.com/javascript/windows/only-
popuponce.html";
var windowprops =
"width=300,height=200,location=no,toolbar=no,menubar=no,scrollbars=no,resiz
```

```
able=yes";
function GetCookie (name) { var arg = name + "="; var alen = arg.length; var
clen = document.cookie.length; var i = 0; while (i < clen) { var j = i
+ alen;
if (document.cookie.substring(i, j) == arg) return getCookieVal (j); i
= document.cookie.indexOf(" ", i) + 1; if (i == 0) break; } return
null;}
function SetCookie (name, value) { var argv = SetCookie.arguments; var argc =
SetCookie.arguments.length; var expires = (argc > 2) ? argv[2] : null;
var path = (argc > 3) ? argv[3] : null; var domain = (argc > 4) ? argv[4]
: null; var secure = (argc > 5) ? argv[5] : false; document.cookie = name
+ "=" + escape (value) + ((expires == null) ? "" : ("; expires=" +
expires.toGMTString())) + ((path == null) ? "" : ("; path=" + path)) +
((domain == null) ? "" : ("; domain=" + domain)) + ((secure == true) ?
"; secure" : "");}
function DeleteCookie (name) { var exp = new Date(); exp.setTime
(exp.getTime() - 1); var cval = GetCookie (name); document.cookie = name
+ "=" + cval + "; expires=" + exp.toGMTString();}var exp = new Date();
exp.setTime(exp.getTime() + (expDays*24*60*60*1000));
function amt(){var count = GetCookie('count')if(count == null)
{SetCookie('count','1')return 1}
else {var newcount = parseInt(count) +
1;DeleteCookie('count')SetCookie('count',newcount,exp)return count }}
function getCookieVal(offset) {var endstr = document.cookie.indexOf (";",
offset);if (endstr == -1)endstr = document.cookie.length;return
unescape(document.cookie.substring(offset, endstr));}
function checkCount(){var count = GetCookie('count');if (count == null)
{count=1;SetCookie('count', count, exp);window.open(page, "",
```

windowprops);}

else {count++;SetCookie('count', count, exp); }}

// End

-->

</script>

Add the OnLoad event handler to the BODY tag: OnLoad=″checkCount()″>

Special Techniques: Frames

Break out of someone's frames

if (self.parent.frames.length != 0)

self.parent.location="http://www.mydomain.com"

also seen as (but do not understand differences yet):

if (self != top) top.location.replace(self.location)

Force document to open within a frame

if (top == window) top.location.href= “frameset.html”

where frameset.html is a fixed file

to use as a generic technique applicable to a set of documents, frameset.html needs to be a dynamic document where the contents of one frame is controlled by a query string and some of the actual html is created dynamically by document.write:

if (top == window) top.location.href= “frameset.html?” +

escape(window.location.href);

frameset.html includes:

<html>

<head>

var url = location.search ?

unescape(location.search.substring(1)) : “default.html”;

var html = “”

html += “<frameset rows=’150, *’>”

html += “<frame name=’desktopframe’ src=’desktoplinks.htm’>”

```
html += “<frame name=’deskcontent’ scr=’” + url + “’>”
html += “<\/frameset>”
</script>
</head>
<script>
document.write(html)
</script>
</html>
```

Special Techniques: Dialog Boxes (modal windows)

There are three kinds of dialog boxes:

a. alert box: has message, OK button, and no title.

b. confirmation box: has message, OK and Cancel buttons, and no title.

c. prompt box: has message, OK and Cancel buttons, title, and field to capture entered text.

Open an “alert” box

```
<script type="text/javascript">
alert("Hello World!")
</script>
```

Open a “confirm” box

```
<script type="text/javascript">
var name = confirm("Press a button")
if (name == true)
{ document.write("You pressed OK") }
else
{ document.write("You pressed Cancel") }
```

</script>

Prompt box

Displays window with title “Explorer User Prompt”, label “JavaScript Prompt”, buttons OK and Cancel. Entered text is captured as variable.

<body>

<script type="text/javascript">

var name = prompt("Please enter your name","")

if (name != null && name != "")

document.write("Hello " + name)

</script>

Force a line break with “\n”

alert("Hello you\nFrom me")

presents a message box like:

Hello you

From me

Special Techniques: Cross Window Scripting

Browser instance (?) knows about all its open windows

When about.html does window.open(“index.html”,”home”) the file is loaded into the window named “home” if it is already open (regardless of its contents). If the window is not open, it is opened, the file is loaded, and the window is given the focus. The browser will retain knowledge of a named window after it is closed as long as references to it exist. It is possible to control window B from window A but only in certain limited ways

Helpful code:

winb.focus() – gives focus to the Window object named winb. NOTE: this does not work right

in IE5.

self.blur() – takes focus away from the current window. This has the effect of the focus()

method above ONLY when there are only two browser windows open.

window.opener – refers to the Window object that opened the current window. This is the only

way to know anything about that window. Can be coupled with the ability to create custom

properties. Does not work in IE5!

Example: if (window.opener.name = “you”) …, if (window.opener.custom = “yes”)

window.location.href – setting this property causes the browser to load a new file.

Example: window.location.href = myurl

window.open (file, windowname) – opens file in named window; if window already exists, it does not automatically get the focus.

Theoretically, you should be able to create a custom property of the global object (Window object) which can be accessed in a second window: window.creator = “index”; the access in the second window is like: if (window.opener.creator = “index”) . At least in IE5 a spawned window cannot access it: it is not defined.

The following samples are used to control navigation between two windows—a home page

window and a subject window used for all subjects listed on the home page. The code handles

the situation where a subject file is opened by the user or some page not the home page. (This

code works in IE 5.0, but in IE 5.5 it spawns additional windows.)

window.name – “stmain” // done in home page

/* function openwin opens the named file in the named window. If the window is

already open, it makes it active. Used on the home page; winname is

“subject” in most cases */

var w = '' // variable for window object created by page

linked to this file

```
function openwin(file, winname)
{
if (w.location && !w.closed) // if window object w exists and is not closed
{ w.location.href = file } // load page into its location.href
else
{ w=window.open(file,winname); // open the new window
// if (!w.opener) {w.opener = self} ;
}
self.blur()
}

// next function used to link (back) to Home page from a subject page
function openhome(file)
{
var home = ''
if (window.name == "subject") // then I was opened by home page
 { home = window.open(file, "stmain")
self.blur() }
else
{
 window.name = "subject"
 home = window.open(file, "stmain")
}
}
```

Special Techniques: Messages

Set text in status bar

```
window.status = "put your message here"
```

Status bar message

```
<BODY OnLoad="window.defaultStatus='Hello!';">
```


Scroll text in the browser's status bar to the left

```
<HTML>
<HEAD>
<TITLE>The Herballadies : MicroWater</TITLE>
<script language="JavaScript">
<!-- Begin code
function scroll(seed)
{
var m1 = "Welcome to The Herballadies call 1-888-258-5949";
var msg=m1;
var out = " ";
var c = 1;
if (seed > 100) {
seed--;
var cmd="scroll(" + seed + ")";
timerTwo=window.setTimeout(cmd,100);
}
else if (seed <= 100 && seed > 0) {
for (c=0 ; c < seed ; c++) {
out+=" ";
}
out+=msg;
seed--;
var cmd="scroll(" + seed + ")";
```

```
window.status=out;
timerTwo=window.setTimeout(cmd,100);
}
else if (seed <= 0) {
if (-seed < msg.length) {
out+=msg.substring(-seed,msg.length);
seed--;
var cmd="scroll(" + seed + ")";
window.status=out;
timerTwo=window.setTimeout(cmd,100);
}
else {
window.status=" ";
timerTwo=window.setTimeout("scroll(100)",75);
}
}
}
// -- End code -->
</script>
</HEAD>
<BODY BACKGROUND="sky.jpg" BGCOLOR=#CCCCFF TEXT=#000000 LINK=#0000FF
VLINK=#0000AA ALINK=#FFFF00
onLoad="timerONE=window.setTimeout('scroll(100)',500);">
<CENTER> The Herballadies</CENTER>
```

Special Techniques: Element Dimensions

Determine width of the browser window

You may want to know how wide a window is. This can be helpful in designing a page. It may

also be helpful in other contexts.

You can write a JavaScript program to display the width of the browser window. Display the width in an alert box when the page loads. Locate the program in the BODY element. When the page loads an alert box presents the width. Then adjust the width of the page to suit yourself, then refresh the contents of the page. You'll see a message box with the current window width.

```
<script type="text/javascript">

var w = document.body.offsetWidth

var m = "The width of the browser window: " + w
```



```
alert(m);

</script>
```

Change height of element

You can change the height of an element after it has been rendered, meaning the code must be located below the HTML of the element or executed as a load event and resize event. In this

example, a DIV (ID=xx) is contained within a TD table cell (ID=cell1), the table has 2 rows and 2 columns, but only 3 cells as the first cell spans both rows. I want to force the inner DIV, which has a colored background, to fill the first cell. The final height is reduced by 20 pixels to accommodate a padding set in the stylesheet.

In this example offsetHeight is a number with an intrinsic UOM of pixels, it is a element property. To the contrary, paddingBottom is an attribute of style and has a value that is a string that must end in a UOM. You must be careful to add the UOM in code.

```
<script type="text/javascript">

var cellheight= document.getElementById("cell1").offsetHeight;

var textheight = document.getElementById("xx").offsetHeight;

var diff = cellheight - textheight;

var diff = cellheight - textheight;
```

var adj = diff - 20

document.getElementById("xx").style.paddingBottom = adj + "px"

</script>

Special Techniques: Browser Tools

Refresh page

location.reload()

Print the page with JavaScript

window.print()

<A HREF=”javascript:window.print()”>Click to print this page</A>

<FORM><INPUT TYPE=”button” onClick=”window.print()”></FORM>

Print this page.

Go Back with Link in Text

Back

Print and close a different document with Link in Text

<script type=”text/javascript”>

function printclose()

{ var x = window.open(“different.html”, “whatever”)

x.print()

x.close

}

</script>

. . .

. . .

prompts for print options, then closes the window

Determine browser

ƒ Call this function in the <BODY onload=”getBrowser()”> tag.

```
function getBrowser()
{
var b = new Object();
b.name = navigator.appName;
b.version = parseInt(navigator.appVersion);
b.agent = navigator.userAgent;
document.soq.browser.value = b.name + ", " + b.version + ", " + b.agent;
}
```

ƒ if user agent string contains “SV1” then the browser is IE with SP2

```
If (b.agent.indexOf("SV1")!= -1 // browser is IE with SP2
Browser info
<body>
<script type="text/javascript">
document.write("BROWSER: ")
document.write(navigator.appName + "<br>")
document.write("BROWSERVERSION: ")
document.write(navigator.appVersion + "<br>")
document.write("CODE: ")
document.write(navigator.appCodeName + "<br>")
document.write("PLATFORM: ")
document.write(navigator.platform + "<br>")
document.write("REFERRER: ")
document.write(document.referrer + "<br>")
</script>
</body>
```

yields:

BROWSER: Microsoft Internet Explorer

BROWSERVERSION: 4.0 (compatible; MSIE 5.01; Windows NT 5.0; PGE)

CODE: Mozilla

PLATFORM: Win32

REFERRER: http://www.w3schools.com/js/tryit.asp?filename=tryjs_browserdetails

Special Techniques: Forms

Set focus on a particular form element

It is useful to set the focus on the first element when a form opens.

```
<head>
<script type="text/javascript">
// puts focus on first element in first form
function setfocus(a,b)
{ document.forms[a].elements[b].focus() }
</script>
</head>
<body onLoad=”setfocus(0,0)”>
<form>
<input type="text" name="field" size="30">
<input type="text" name="userid" size="10">
<input type="button" value="Get Focus">
</form>
</body>
```

Show/hide text

1. Group text to be shown/hidden by division and assign each an id.

```
<DIV id=all>
 . . .
```

</DIV>

2. In script just before </BODY> tag set style property:

var e = document.getElementById(“all”);

e.style.display = “none”; // hide division

3. When appropriate button is clicked,

var e = document.getElementById(“all”);

e.style.display = “block”; // show division

Special Techniques: Editing Form

Convert to upper case

field.value = field.value.toUpperCase();

Convert to lower case

field.value = field.value.toLowerCase();

Check for numeric values

form element includes event procedure;

onChange=”checkNumeric(this, 0)”

JavaScript function:

```
function checkNumeric(which, x)
{
// the optional second parameter allows you to edit for presence of dash
var digits;
if (x == 1)
digits="0123456789-;.";
else
digits="0123456789";
var temp;
for (var i=0;i<which.value.length;i++)
{
```

```
temp=which.value.substring(i,i+1);
if (digits.indexOf(temp)==-1)
{
 alert("Enter only digits!");
which.focus();
return false;
}
}
return true;
}
```

Another way to edit for numbers—integers uses regular expression:

```
var re = /^\d+$/;
if (!re.test(e.value))   // then there is an error
```

Edit dates

Uses regular expression to check for the following valid date formats:

MM/DD/YY MM/DD/YYYY MM-DD-YY MM-DD-YYYY

```
function isDate(field) {
 var dateStr = field.value;
if (dateStr != "") {
//  var datePat = /^(\\d{1,2})(\\/|-)(\\d{1,2})\\2(\\d{4})\$/; //
requires 4 digit year
 var datePat = /^(\d{1,2})(\/|-)(\d{1,2})\2(\d{4})$/; // requires 4  digit year
var matchArray = dateStr.match(datePat); // is the format ok?
if (matchArray == null) {
```

```
 alert(dateStr + " Date is not in a valid format(MM-DD- YYYY).");
field.focus();
return false;
}
month = matchArray[1]; // parse date into variables
day = matchArray[3];
year = matchArray[4];
if (month < 1 || month > 12) { // check month range
 alert("Month must be between 1 and 12.");  field.focus();
return false;
}
if (day < 1 || day > 31) {
 alert("Day must be between 1 and 31.");  field.focus();
return false;
}
if ((month==4 || month==6 || month==9 || month==11) && day==31)
{
 alert("Month "+month+" doesn't have 31 days!")  field.focus();
return false;
}
  if (month == 2) { // check for february 29th
 var isleap = (year % 4 == 0 && (year % 100 != 0 || year %  400 == 0));
if (day>29 || (day==29 && !isleap)) {
 alert("February " + year + " doesn't have " + day + "  days!");
field.focus();
return false;
}
}
```

```
}
return true;
}
```

Handle radio buttons

ƒ When accessing elements using the document.soq.elements[i] array, you will access

individual radio buttons. In order to refer to the group of radio buttons of the same name,

you must use the element name:

```
var e = document.forms[0].elements[i];
var radiogroup = document.forms[0].elements[e.name];
```

ƒ Check to see if one radio button in its group is checked:

```
if (e.type == "radio")
{
if (!isVisible(e))
return "N";
var radiogroup = document.soq.elements[e.name]; // get the whole group
var itemchecked = false;
for(var j = 0 ; j < radiogroup.length ; j++)
{
if (radiogroup[j].checked)
itemchecked = true;
}
if ((!itemchecked) && (e == radiogroup[0])) // do only once for group
 // handle the error – but only once for the group
  . . .
  }
```

ƒ Test an index:

```
if (document.soq.biztype[0])
```

ƒ How to tell how many elements are in a group?

document.soq.biztype.length

ƒ Only way to tell if a form element document.soq.elements[varname] is a radio button:

if (typeof(e.length) = “number”)

e = document.soq.elements[i];

if (e.type == “radio”)

Edit radio button or check box

Many times you will want to be able to get the selected radio button and/or check box in a browser environment. For the radio button, you will want to check the value and perform some task. Or, you will want to verify that one radio button has been selected. For check boxes, you may want to see which ones are selected or you could want to check that at least one is selected. Here’s a couple of functions that work with radio buttons and check boxes. These are set up to be generic, so they can be reusable.

```
function getSelectedRadio(buttonGroup) {
// returns the array number of the selected radio button or -1 if no button
is selected
if (buttonGroup[0]) { // if the button group is an array (one button is not
an array)
for (var i=0; i<buttonGroup.length; i++) {
if (buttonGroup[i].checked) {
return i
}
}
```

```
} else {
if (buttonGroup.checked) { return 0; } // if the one button is checked,
return zero
}
// if we get to this point, no radio button is selected
return -1;
} // Ends the "getSelectedRadio" function
function getSelectedRadioValue(buttonGroup) {
// returns the value of the selected radio button or "" if no button is
selected
var i = getSelectedRadio(buttonGroup);
if (i == -1) {
return "";
} else {
if (buttonGroup[i]) { // Make sure the button group is an array (not just
one button)
return buttonGroup[i].value;
} else { // The button group is just the one button, and it is checked
return buttonGroup.value;
}
}
} // Ends the "getSelectedRadioValue" function
function getSelectedCheckbox(buttonGroup) {
// Go through all the check boxes. return an array of all the ones
// that are selected (their position numbers). if no boxes were checked,
// returned array will be empty (length will be zero)
var retArr = new Array();
var lastElement = 0;
```

```
if (buttonGroup[0]) { // if the button group is an array (one check box is
not an array)
for (var i=0; i<buttonGroup.length; i++) {
if (buttonGroup[i].checked) {
retArr.length = lastElement;
retArr[lastElement] = i;
lastElement++;
}
```


```
}
} else { // There is only one check box (it's not an array)
if (buttonGroup.checked) { // if the one check box is checked
retArr.length = lastElement;
retArr[lastElement] = 0; // return zero as the only array value
}
}
return retArr;
} // Ends the "getSelectedCheckbox" function
function getSelectedCheckboxValue(buttonGroup) {
// return an array of values selected in the check box group. if no boxes
// were checked, returned array will be empty (length will be zero)
var retArr = new Array(); // set up empty array for the return values
var selectedItems = getSelectedCheckbox(buttonGroup);
if (selectedItems.length != 0) { // if there was something selected
retArr.length = selectedItems.length;
for (var i=0; i<selectedItems.length; i++) {
```

```
if (buttonGroup[selectedItems[i]]) { // Make sure it's an array
retArr[i] = buttonGroup[selectedItems[i]].value;
} else { // It's not an array (there's just one check box and it's
selected)
retArr[i] = buttonGroup.value;// return that value
}
}
}
return retArr;
} // Ends the "getSelectedCheckBoxValue" function
```

To use one of these functions, just make a call and pass the radio button or check box object. For example, if you want to find out if at least one check box is selected and the check box field name is MyCheckBox, then write the following statements:

```
var checkBoxArr = getSelectedCheckbox(document.forms[0].MyCheckBox);
if (checkBoxArr.length == 0) { alert("No check boxes selected"); }
Check for whitespace characters
function isBlank(e)
{
// e is a form element
var v = e.value;
if ((v == null ) || (v == "") || (v.length == 0))
return true;
// check for whitespace characters in form element
for (var i = 0; i < v.length; i++)
{
var c = v.charAt(i);
```

```
if ((c != ' ') && (c != "\n") && (c != ""))
return false;
}
return true;
}
```

Perform all edits

```
<FORM onsubmit=”return EditForm(this)” . . .>
. . .
</FORM>
. . .
function setEditProperties(f)
{
// f is form name; properties are used by function EditForm
// assume all radio buttons are required; they have no initial value
document.soq.product.required = true;
document.soq.contact.required = true;
```



```
document.soq.name.required = true;
document.soq.zip.zip = true;
setBillAddress();
var mailbill = new Array("billstreet", "billcity", "billstate", "billzip");
// all or none
document.soq.billzip.allornone = mailbill;
document.soq.billzip.groupname = "Complete Mailing/Bill Address";
document.soq.taxid.taxid = true;
document.soq.naics.integer = true;
```

```
document.soq.email.email = true;
document.soq.url.url = true;
var geo = new Array(48);
initGeoArray(geo);
document.soq.allca.atleast = geo; // at least one of a group of
checkboxes is required
document.soq.allca.groupname = "Geographical Service Area";
document.soq.gross.amount = true;
document.soq.start1.monthyear = true;
document.soq.cphone1.telephone = true;
document.soq.expdate.date = true;
. . .
}
function EditForm(f)
{
// this catches all the errors and reports them back at once
setEditProperties(f);
var ErrorCount = 0;
var msg;
var empty_fields = "";
var errors = "";
var n = 0;
var em = "";
var req = "";
// check for required fields
// isEdit() returns an error message or "N" if no error
for (var i = 0; i < document.soq.length; i++)   // loop through elements
in the form
```

```
{
var e = document.soq.elements[i];
em = isEdit(e);
if ((em != "undefined") && (em != "N")) // error
{
if ((e.required) || (e.type == "radio") || (e.atleast))
empty_fields += em;
else
errors += em;
}
}
if (!empty_fields && !errors)
return true;
msg = "_____\n\n";
msg += "The form was not submitted because of the following error(s).\n";
msg += "Please correct these error(s) and resubmit. \n";
msg += "_____\n\n";
if (empty_fields)
msg += "- The following required field(s) are empty:" + empty_fields +
"\n";
if (errors)
msg += "\n" + errors;
alert(msg);
return false;
}
```

function isEdit(e)


```
{
// find the first error for a field
// return error message or "N" (if no error)
var msg = "";
if (e.required)
{
if (!isVisible(e))
return "N";
switch (e.type)
{
case 'text':
{
if (isBlank(e))
{
msg = "\n " + expandName(e.name);
return msg;
}
break;
}
case 'textarea':
{
if (isBlank(e))
{
msg = "\n " + expandName(e.name);
```

```
return msg;
}
break;
}
case 'select-one':
{
if (e.selectedIndex < 1)
{
msg = "\n " + expandName(e.name);
return msg;
}
break;
}
}
}
if (e.integer)
{
msg = isInteger(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.zip)
{
msg = isZip(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.type == "radio")
```

```
{
if (!isVisible(e))
return "N";
var radiogroup = document.soq.elements[e.name]; // get the whole set
of radio buttons
var itemchecked = false;
for(var j = 0 ; j < radiogroup.length ; j++)
{
if (radiogroup[j].checked)
itemchecked = true;
}
```


```
if ((!itemchecked) && (e == radiogroup[0])) // do only once for
group
{
// hope to replace with generic code - to identify division containing form
element
var x = document.getElementById("dod");
if (e.name != "cpuc")
{
msg = "\n " + expandName(e.name);
return msg;
}
else
if (x.style.display == "block")
{
```

```
msg = "\n " + Name(e.name);
return msg;
}
}
}
if ((e.atleast) && (e.type == "checkbox"))
{
if (!isVisible(e))
return "N";
var groupchecked = false;
for (var k = 0; k < e.atleast.length; k++)
{
var en = e.atleast[k]; // element name
var c = document.soq.elements[en].checked;
if (c)
{
groupchecked = true;
break;
}
}
if (groupchecked == false)
{
msg = "\n " + e.groupname;
return msg;
}
}
if ((e.atleast) && (e.type == "text"))
{
```

```
if (!isVisible(e))
return "N";
var groupchecked = false;
for (var k = 0; k < e.atleast.length; k++)
{
if (!isBlank(e))
{
groupchecked = true;
break;
}
}
if (groupchecked == false)
{
msg = "\n " + e.groupname;
return msg;
}
}
if (e.allornone)
{
if (!isVisible(e))
return "N";
var cnterror = 0; // count non-bland fields
for (var k = 0; k < e.allornone.length; k++)
```


```
{
var en = e.allornone[k]; // element name
```

```
var ele = document.soq.elements[en];
if ((ele.type == "text") || (ele.type == "textarea"))
{
if (!isBlank(ele))
{
cnterror++;
}
}
else
if (ele.type == "select-one")
{
if (ele.selectedIndex > 0)
{
cnterror++;
}
}
}
if ((cnterror > 0) && (cnterror < e.allornone.length))
return "\n - " + e.groupname + " must be present or absent";
}
if (e.telephone)
{
msg = isTelephone(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.amount)
{
```

```
if (!isVisible(e))
return "N";
msg = isAmount(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.date)
{
if (!isVisible(e))
return "N";
msg = isSOQDate(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.taxid)
{
msg = isTaxId(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.email)
{
msg = isEmail(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
if (e.url)
{
```

```
msg = isURL(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
```


```
}
if (e.monthyear)
{
msg = isMonthYear(e);
if ((msg != "undefined") && (msg != "N"))
return msg;
}
return "N";
}
function isBlank(e)
{
var v = e.value;
if ((v == null ) || (v == "") || (v.length == 0))
return true;
// check for whitespace characters in form element
for (var i = 0; i < v.length; i++)
{
var c = v.charAt(i);
if ((c != ' ') && (c != "\n") && (c != ""))
return false;
}
return true;
```

```
}
function isInteger(e)
{
// all non-blank characters must be digits; may have trailing blanks; may be
all blank
if (e.type != "text")
{
alert("error! " + expandName(e.name) + " cannot be integer because type = "
+ e.type);
return "N";
}
if (isBlank(e))
return "N";
/*
var digits = "0123456789";
var temp;
for (var i = 0; i < e.value.length; i++)
{
temp = e.value.substring(i, i+1);
alert(e.name + ", " + e.value + ", " + temp + " position in digits = " +
digits.indexOf(temp));
if (digits.indexOf(temp) == -1)
return "\n " + expandName(e.name) + " should be integer and nonzero";
}
if (e.value > 0)
return "N";
else
return "\n " + expandName(e.name) + " should be integer and nonzero";
```

```
-- */
var re = /^\d+$/;
if ((!re.test(e.value)) || (e.value == 0) || (e.value == "0"))
{
if (e.required)
return "\n " + expandName(e.name) + " should be integer and nonzero";
else
return "\n - " + expandName(e.name) + " should be integer and nonzero";
}
return "N";
}
function isZip(e)
{
// all 5 characters must be digits and non-zero
if (e.type != "text")
{
alert("error! " + expandName(e.name) + " cannot be zip because type = " +
e.type);
return "N";
}
var re = /^\d+$/; // match beginning and end
if ((e.value.length != 5) || (!re.test(e.value)) || (e.value == 0)) // if
not integer, or length not = 5, or zero
{
if (e.required)
return "\n " + expandName(e.name) + " should be 5 digits and nonzero";
else
return "\n - " + expandName(e.name) + " should be 5 digits and nonzero";
```

```
}
return "N";
}
function isTelephone(e)
{
if (e.type != "text")
{
alert("error! " + expandName(e.name) + " cannot be telephone because type =
" + e.type);
return "N";
}
if (isBlank(e))
return "N";
var savePhone = e.value;
var phoneDelimiters = "[()- ";
var normalizedPhone = stripCharsInBag(e.value, phoneDelimiters); //
remove blanks, dashes, and parentheses
e.value = normalizedPhone;
if ((e.value.length != 10) || (isInteger(e) != "N"))
{
e.value = savePhone;
if (e.required)
return "\n " + expandName(e.name) + " should be 10-digit telephone
number";
else
return "\n - " + expandName(e.name) + " should be 10-digit telephone
number";
}
```

```
e.value = reformat(normalizedPhone, "", 3, "-", 3, "-", 4); //123-456-7890
return "N";
}
function isTaxId(e)
{
if (isBlank(e))
return "N";
var saveid = e.value;
var bag = "-";
e.value = stripCharsInBag(e.value, bag);
if ((!isInteger(e)) || (e.value.length != 9))
{
if (e.required)
return "\n " + expandName(e.name) + " must be valid tax id";
else
return "\n - " + expandName(e.name) + " must be valid tax id";
}
e.value = reformat(e.value, "", 2, "-", 7);
return "N";
```


```
}
function isEmail(e)
{
if (isBlank(e))
return "N";
var saveid = e.value;
```

```
var reEmail = /^.+\@.+\..+$/
if (!reEmail.test(e.value))
{
if (e.required)
return "\n " + expandName(e.name) + " must be valid email
address";
else
return "\n - " + expandName(e.name) + " must be valid email address";
}
return "N";
}
function isURL(e)
{
if (isBlank(e))
return "N";
var saveid = e.value;
var reURL = /\.+/
if (!reURL.test(e.value))
{
if (e.required)
return "\n " + expandName(e.name) + " must be valid URL";
else
return "\n - " + expandName(e.name) + " must be valid URL";
}
return "N";
}
function stripCharsInBag(s, bag)
{
```

```
//builds return string from characters in s that are not in bag
var returnString = "";
for (var i = 0; i < s.length; i++)
{
var c = s.charAt(i);
if (bag.indexOf(c) == -1)
returnString += c;
}
return returnString;
}
function reformat(s)
{
/* ========
Takes one string argument and any number of other arguments (integers
and/or strings).
The odd-numbered (e.g., 1, 3, 5) other argument must be a string. It can
be "".
reformat processes the other arguments in order one by one.
If the other argument is an integer, reformat appends that number of
sequential characters from s to the
returnString.
If the other argument is a string, reformat appends that string to the
returnString.
======= */
var returnString = "";
var arg;
var sPos = 0;
for (var i = 1; i < reformat.arguments.length; i++) // start with second
```

```
argument
{
arg = reformat.arguments[i];
if (i % 2 == 1) // odd-numbered argument
returnString += arg;
else
{
returnString += s.substring(sPos, sPos + arg);
sPos += arg;
}
}
return returnString;
}
function isAmount(e)
{
if (isBlank(e))
return "N";
var saveamt = e.value;
// strip off $,+
var bag = "$,+";
var result1 = stripCharsInBag(e.value, bag);
// find .
var x = result1.indexOf("."); // position of . in string
// if . present, strip it and following characters
if (x != -1)
{
e.value = result1.substring(0, x);
if ((isInteger(e) != "N") && (isInteger(e) != "undefined"))
```

```
{
e.value = saveamt;
if (e.required)
return "\n " + expandName(e.name) + " should be numeric";
else
return "\n - " + expandName(e.name) + " should be numeric";
}
}
else
{
e.value = result1;
if ((isInteger(e) != "N") && (isInteger(e) != "undefined"))
{
e.value = saveamt;
if (e.required)
return "\n " + expandName(e.name) + " should be numeric and
non-zero";
else
return "\n - " + expandName(e.name) + " should be numeric and nonzero";
}
}
return "N";
}
function isSOQDate(e)
{
if (isBlank(e))
return "N";
// parse value into MM, DD, YY or MM, DD, YYYY; delimiter required
```

```
var dmonth;
var dday;
var dyear;
var re = new RegExp ('/', 'gi');
e.value = e.value.replace(re, "-"); // replace / by -
var delim1 = e.value.indexOf("-");
var delim2;
if (delim1 == -1) // no dash in value
{
if (e.required)
return "\n " + expandName(e.name) + " should be valid date MMDD-YYYY";
else
return "\n - " + expandName(e.name) + " should be valid date MMDD-YYYY";
}
else // dash in value
{
dmonth = e.value.substring(0, delim1);
delim2 = e.value.indexOf("-", delim1 + 1);
if (delim2 == -1) // no / in value
{
if (e.required)
return "\n " + expandName(e.name) + " should be valid date MMDD-YYYY";
else
return "\n - " + expandName(e.name) + " should be valid date MMDD-YYYY";
}
dday = e.value.substring(delim1 + 1, delim2);
dyear = e.value.substring(delim2 + 1, e.value.length);
}
```

```
if (!isDate(dyear, dmonth, dday))
{
if (e.required)
return "\n " + expandName(e.name) + " should be valid date MM-DDYYYY";
else
return "\n - " + expandName(e.name) + " should be valid date MM-DDYYYY";
}
return "N";
}
function isMonthYear(e)
{
if (isBlank(e))
return "N";
// parse value into MM, YYYY; delimiter required
var dmonth;
var dyear;
var delim1 = e.value.indexOf("-");
if (delim1 == -1) // no dash in value
{
delim1 = e.value.indexOf("/");
if (delim1 == -1) // no / in value
{
if (e.required)
return "\n " + expandName(e.name) + " should be valid monthyear MM-YYYY";
else
return "\n - " + expandName(e.name) + " should be valid month-year
MM-YYYY";
}
```

```
dmonth = e.value.substring(0, delim1);
dyear = e.value.substring(delim1 + 1, e.value.length);
}
else // dash in value
{
dmonth = e.value.substring(0, delim1);
dyear = e.value.substring(delim1 + 1, e.value.length);
}
if ((!isMonth(dmonth)) || (!isYear(dyear)))
{
if (e.required)
return "\n " + expandName(e.name) + " should be valid month-year
MM-YYYY";
else
return "\n - " + expandName(e.name) + " should be valid month-year MMYYYY";
}
return "N";
}
```

Special Techniques: Change Images

Images can be changed once displayed.

Special Techniques: Floating Menu

This was copied from www.asianinc.org. This code appears just before the </BODY> tag. It creates a vertical menu that floats (so it appears on the screen in the same place) and that can be dragged to a different location.

```
<!-- vmenu -->
<img name='awmMenuPathImg-vMenu' id='awmMenuPathImg-vMenu'
```

```
src='scripts/awmmenupath.gif' alt=''>
<script type='text/javascript'>var MenuLinkedBy='AllWebMenus [2]', awmBN='494';
awmAltUrl='';</script>
<script src='scripts/vMenu.js' language='JavaScript1.2'
type='text/javascript'></script>
<script type='text/javascript'>awmBuildMenu();</script>
<!-- vMenu ends -->
```

The following code comprises vMenu.js:

```
//----------vMenu----------
var awmMenuName='vMenu';
var awmLibraryPath='/vMenu';
var awmImagesPath='/vMenu';
var awmSupported=(navigator.appName +
navigator.appVersion.substring(0,1)=="Netscape5" || document.all ||
document.layers || navigator.userAgent.indexOf('Opera')>-1)?1:0;
if (awmAltUrl!='' && !awmSupported) window.location.replace(awmAltUrl);
if (awmSupported){
var awmMenuPath;
if (document.all) mpi=document.all['awmMenuPathImg-vMenu'].src;
if (document.layers) mpi=document.images['awmMenuPathImg-vMenu'].src;
if (navigator.appName + navigator.appVersion.substring(0,1)=="Netscape5" ||
navigator.userAgent.indexOf('Opera')>-1)
mpi=document.getElementById('awmMenuPathImg-vMenu').src;
awmMenuPath=mpi.substring(0,mpi.length-16);
var nua=navigator.userAgent,scriptNo=(nua.indexOf('Gecko')>-1)?2:
((document.layers)?3:((nua.indexOf('Opera')>-1)?4:((nua.indexOf('Mac')>-1)?
5:1)));
```

```
document.write("<SCRIPT
SRC='"+awmMenuPath+awmLibraryPath+"/awmlib"+scriptNo+".js'><\/SCRIPT>");
var n=null;
awmzindex=1000;
}
var awmSubmenusFrame='';
var awmSubmenusFrameOffset;
var awmOptimize=1;
function awmBuildMenu(){
if (awmSupported){
awmCreateCSS(1,2,1,'#FFFFFF','#000000',n,'bold 12pt
Verdana',n,'outset',2,'#FF0000',0,4)
awmCreateCSS(0,1,0,n,'#BFBFBF',n,n,n,'outset',0,n,0,0);
awmCreateCSS(1,2,1,'#0000FF','#EFEFEF',n,'bold 10pt
Verdana',n,'none',0,'#0000FF',3,1)
awmCreateCSS(0,2,1,'#FF0000','#EFEFEF',n,'bold 10pt
Verdana',n,'solid',1,'#FF0000',3,1)
awmCreateCSS(1,2,1,'#0000FF','#DFDFDF',n,'bold 10pt
Verdana',n,'none',0,'#0000FF',3,1)
awmCreateCSS(0,2,1,'#FF0000','#DFDFDF',n,'bold 10pt
Verdana',n,'solid',1,'#FF0000',3,1)
awmCreateCSS(1,2,1,'#0000FF','#DFDFDF',n,'bold 9pt
Verdana',n,'none',0,'#0000FF',3,1)
awmCreateCSS(0,2,1,'#FF0000','#DFDFDF',n,'bold 9pt
Verdana',n,'solid',1,'#FF0000',3,1)
awmCreateCSS(1,2,1,n,n,n,'9pt Verdana',n,'none',0,n,0,1)
awmCreateCSS(0,1,0,n,n,n,n,n,'outset',0,n,0,0);
awmCreateCSS(0,1,0,n,n,n,n,n,'none',0,n,0,0);
```

```
awmCreateCSS(1,2,1,'#0000FF','#EFEFEF',n,'bold 10pt
Verdana',n,'none',0,'#FF0000',3,1)
awmCreateCSS(0,2,1,'#FF0000','#EFEFEF',n,'bold 10pt Verdana',n,'solid',1,n,3,1)
awmCreateCSS(1,2,1,n,n,n,'9pt Verdana',n,'outset',0,n,0,1)
var s0=awmCreateMenu(0,0,0,0,1,0,1,2,0,2,147,0,0,1,0,"Main Menu","You can drag
this menu to a new location",n,1,0,1,0,n,n);
it=s0.addItem(2,3,3,"Home",n,n,"","",n,n,n,"../index.html",n);
it=s0.addItem(4,5,5,"News/Events",n,n,"","",n,n,n,"../newsEvents.html",n);
it=s0.addItem(2,3,3,"Census Info Center",n,n,"","",n,n,n,"../census.html",n);
var s1=it.addSubmenu(0,0,1,0,0,0,0,9,8,n,"",n,1,0,1,1,n,n);
it=s1.addItem(2,3,3,"CIC Home",n,n,"","",n,n,n,"../census.html",n);
it=s1.addItem(4,5,5,"Publications",n,n,"","",n,n,n,"../census_pub.html",n);
it=s1.addItem(2,3,3,"Census Links",n,n,"","",n,n,n,"../census_links.html",n);
it=s1.addItem(4,5,5,"Contact",n,n,"","",n,n,n,"../census_contact.html",n);
it=s0.addItem(4,5,5,"Housing",n,n,"","",n,n,n,"../housing.html",n);
var s1=it.addSubmenu(0,0,1,0,0,0,0,10,8,n,"",n,1,0,1,1,n,n);
it=s1.addItem(4,5,5,"About Housing",n,n,"","",n,n,n,"../housing.html",n);
it=s1.addItem(2,3,3,"Affordable
Housing",n,n,"","",n,n,n,"../aHousing.html","new");
it=s1.addItem(4,5,5,"Single-Family
Housing",n,n,"","",n,n,n,"../singleFamilyHousing.html","new");
it=s1.addItem(11,3,12,"First Time
Homebuyer",n,n,"","",n,n,n,"../homebuyers.html","new");
it=s0.addItem(2,3,3,"Small Business",n,n,"","",n,n,n,"../business.html",n);
it=s0.addItem(6,7,7,"WMBE<br>Clearinghouse",n,n,"","",n,n,n,"../clearinghouse.h
tml",n);
var s1=it.addSubmenu(0,0,1,0,0,0,0,10,8,n,"",n,1,0,1,1,n,n);
```

```
it=s1.addItem(4,5,5,"Background
Info",n,n,"","",n,n,n,"../clearinghouse.html",n);
it=s1.addItem(2,3,3,"Application
Form",n,n,"","",n,n,n,"../wmbe_application_form.html",n);
it=s1.addItem(4,5,5,"Printable
Brochure",n,n,"","",n,n,n,"../wmbe_brochure.html",n);
it=s1.addItem(2,3,3,"FAQs",n,n,"","",n,n,n,"../wmbe_faq.html",n);
it=s1.addItem(4,5,5,"Contact Us",n,n,"","",n,n,n,"../wmbe_contact.html",n);
it=s0.addItem(2,3,3,"Social Programs",n,n,"","",n,n,n,"../social.html",n);
it=s0.addItem(4,5,5,"Company Info",n,n,"","",n,n,n,"../contact.html",n);
var s1=it.addSubmenu(0,0,1,0,0,0,0,10,13,n,"",n,1,0,1,1,n,n);
it=s1.addItem(4,5,5,"Contact Info",n,n,"","",n,n,n,"../contact.html",n);
it=s1.addItem(2,3,3,"Board",n,n,"","",n,n,n,"../board.html",n);
it=s1.addItem(4,5,5,"Jobs",n,n,"","",n,n,n,"../jobs.html",n);
s0.pm.buildMenu();
}}
```

Special Techniques: Changing Content Dynamically

Display a series of images one at a time with “x of y”

The design here is to present one image with [Previous] and [Next] buttons and text “x of y”

where x is the relative position of the image in the series and y is the total number of images. To

do this the images are placed in a particular directory, the filenames are placed in an array, and

JavaScript handles the Next and Previous events. CSS centers the “x of y” text between the two

buttons. In this example the image is changed and the text that presents its position in the series

is changed.

```
<head>
```

```
. . .
<style>
.photo { border: 2px solid red; }
#of { margin-left: 100px; margin-right: 100px; }
button { width: 90px; }
</style>
<script type="text/javascript">
var picDirectory = "Photos/";
var cntPics = 5;
var a = new Array(cntPics);
a[0] = "DSC00783.JPG";
a[1] = "DSC00784.JPG";
a[2] = "DSC00785.JPG";
a[3] = "DSC00786.JPG";
a[4] = "DSC00787.JPG";
var i = 0;
var from = 0;
function fullpicURI(j)
{
var URI = picDirectory + a[j];
return URI;
}
function prev()
{
if (i == 0) return;
i--;
var fn = fullpicURI(i);
document.images["pic"].src = fn;
```

```
replaceFrom();
}
function next()
{
if (i == (cntPics - 1)) return;
i++;
var fn = fullpicURI(i);
document.images["pic"].src = fn;
replaceFrom();
}
function writemofn()
{
document.write("1 of " + cntPics);
}
function replaceFrom()
{
document.getElementById("of").innerHTML = (i + 1) + " of " + cntPics;
}
</script>
</head>
<body>
<h1>Test of images</h1>
<div class="photo">
<button onclick="prev()">Previous</button>
<span id="of">
<script type="text/javascript">
writemofn();
</script>
```

<button onclick="next()">Next</button>

<img name=pic src="Photos/DSC00783.JPG">

Special Techniques: Miscellaneous

Using mailto as a URL

mailto can have one or more parts:

ƒ “mailto:” which is like a protocol

ƒ the recipient(s) email address

ƒ one or more “headers,” one for each part of the message (e.g., cc, bcc, subject, and body)

Multiple addresses are separated by a comma. The first header is separated from the recipient’s email address by “?”. Subsequent headers are separated from each other by a “&”. Link to send email with mailto

Email Us

Email Us

Email Us

Email Us

Email Us

This approach causes the email message to be presented to the user for completion and sending, it is not sent invisibly. The user can change anything on the email before sending it manually.

It is possible to initiate the body of the email, but this is trickier. You must encode special characters like interword blanks (%20) and new lines (%0D). You can use

the JavaScript function encodeURIComponent() to make this easier, but it must be used with the document.write method.

For example:

<script language="JavaScript"><document.write("<a

href=\"mailto:you@here.com?subject=" + encodeURIComponent("When, when is

now? (if \"now\" is here)") + "\">mail me!")</script>

Link to email while keeping email address obscure in HTML (to avoid spam)

<script type="text/javascript">

var td = "mailto:sjda@pge.com"

var tdx = "mai" + "lto:" + "sjda@p" + "ge.com"

</script>

click to initiate email with

JavaScript. This works!

click to initiate email with

JavaScript. This works!</p

Using location.href to send email

<script type="text/javascript">

var td = "mailto:sjda@pge.com;mmo7@pge.com?subject=try this?body=blah blah

blah";

parent.location.href=td;

Link to other page

<script type="text/javascript">

function jump()

{ location="http://www.xyz.com/" }

</script>

</head>

<body>

<form><input type="button" onclick="jump()" value="New location"></form>

Tie a behavior to an event

```
window.onblur = function() { window.close() } // close window when loses focus
```

Display date like Tuesday, August 07, 2001

```
<BODY>
<SCRIPT LANGUAGE="Javascript"><!--
// ***********************************************
// AUTHOR: WWW.CGISCRIPT.NET, LLC
// URL: http://www.cgiscript.net
// Use the script, just leave this message intact.
// Download your FREE CGI/Perl Scripts today!
// ( http://www.cgiscript.net/scripts.htm )
// ***********************************************
// Get today's current date.
var now = new Date();
// Array list of days.
var days = new
Array('Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday
');
// Array list of months.
var months = new
Array('January','February','March','April','May','June','July','August','Se
ptember','October','November','December');
// Calculate the number of the current day in the week.
var date = ((now.getDate()<10) ? "0" : "")+ now.getDate();
// Calculate four digit year.
function fourdigits(number)  {
 return (number < 1000) ? number + 1900 : number;
}
```

```
// Join it all together
today = days[now.getDay()] + ", " +
months[now.getMonth()] + " " +
date + ", " +
(fourdigits(now.getYear())) ;
// Print out the data.
document.write("Today\'s date is " +today+ ".");
//--></SCRIPT>
Mouseover changes image and displays text
from www.dwellmag.com
function changeImages() {
if (document.images) {
for (var i=0; i<changeImages.arguments.length; i+=2) {
 document[changeImages.arguments[i]].src =
eval(changeImages.arguments[i+1] + ".src");
} } }
…
 document.write('<td height=23 width=144 bgcolor=#ffffff>
<a href="curr_01.html?noflash" onmouseover="changeImages(' + "'image1',
'image1on', 'infoimage', 'infoimage1'" + ')" onmouseout="changeImages(' +
"'image1', 'image1off', 'infoimage', 'infoimagedefault'" + ')"><img
src=imgs/button_mainnav.gif width=24 height=23 border=0 name=image1></a>
<a href="event_01.html?noflash" onmouseover="changeImages(' + "'image2',
'image2on', 'infoimage', 'infoimage2'" + ')" onmouseout="changeImages(' +
"'image2', 'image2off', 'infoimage', 'infoimagedefault'" + ')"><img
src=imgs/button_mainnav.gif width=24 height=23 border=0 name=image2></a>
<a href="diary1_01.html?noflash" onmouseover="changeImages(' + "'image3',
'image3on', 'infoimage', 'infoimage3'" + ')" onmouseout="changeImages(' +
```

"'image3', 'image3off', 'infoimage', 'infoimagedefault'" + ')"><img

src=imgs/button_mainnav.gif width=24 height=23 border=0 name=image3>

Function indent is used for index entries where each lower level is indented a little more.

It builds a line of HTML that starts with the indentation (in the form of non-breaking space)

continues with the text of the entry with a link to the document and ends with a line break.

<head>

<script type="text/javascript">

<!--

/*

If there is no linked document, the a tag is ommitted. The html that precedes the calls to this function must include the paragraph tag. The html that follows the calls to this function must include the end paragraph tag

(so the style will work).

Three parameters: 1) number of spaces to indent; 2) text for entry; 3) document to link to

(optional).

*/

function indent(w, t, l) {

var c

for (c=0; c<=w; c=c+1) { document.write(" ") }

if (l == null) {document.write(t, "
")}

else {document.write("", t, "
")}

}

//-->

</script>

code that calls the above function:

<p class=e>

<script type="text/javascript">

indent(0, "Adjustments", "adjustments.pdf")

indent(5, "Automatically created adjustments")

Get IP address of client

<HEAD>

<SCRIPT>

var ip = '<!--#echo var="REMOTE_ADDR"-->'

function ipval() {

document.myform.ipaddr.value=ip;

}

</SCRIPT>

</HEAD>

<BODY onLoad="ipval()">

Block copy of files from client

Dan, This web site somehow has protected its content from being copied. After you look at it, try right-clicking on an image - a ″no you don't″ message box appears. How do they do that?

http://members.teleweb.at/tuscaloosa-maine-coons/default.html

From: Niranjan Upadhayay <NiranjanUpadhayay@Mortgage.com>

This is what they have done!

<script language="javascript">

function prevent(e) {

if (document.all) {

if (event.button == 2) {

alert(message);

return false;

}

}

}

document.onmousedown=prevent()

</script>

This guys were not smart enough to prevent it from the menu

View-Source

Go ahead and use it from menu!!

Niranjan

Evaluate an expression or call a function after a period of time

setInterval(expression, milliseconds) // evaluates expression every interval

until cancelled by clearInterval

setInterval(function, milliseconds) // executes function every interval

until cancelled by clearInterval

setTimeout(expression, milliseconds) // evaluates expression once after time

period

setTimeout(function, milliseconds) // executes function once after time

period

Updating copyright notice

This copyright notice places the current year on the page so that copyright notices are always

current. Once it’s on your page ther is no need to update the script.

<script language = 'JavaScript'>

```
<!—
var today = new Date()
var year = today.getFullYear());
document.write('© '+year+' all rights reserved');
//-->
</script>
```

Date last modified

Use this script to show automatically when your page was last modified. This only works if the

web server provides the date-time, otherwise the value is zero. The lastModified property is a date

and time string like M/D/Y h:m:s.

```
<script language="javascript">
<!—
document.write('Last modified '+document.lastModified);
//--
</script>
```

Error handling

NOTE: untested

At beginning of first embedded script:

```
self.error = reportError;
. . .
function reportError(msg, line, url)
{
var w = window.open();
var d = w.document;
d.write(‘<HTML><HEAD><TITLE>Error Handler</TITLE></HEAD><BODY>’);
d.write(‘<P>A JavaScript error has occurred.<>BR>Message: ‘ + msg + ‘<BR>Line
number: ‘ + line’);
```

```
d.write('</BODY</HTML>');

d.close();

}
```

Add this page to favorites

```
<span
style='cursor:hand;textdecoration:underline;'onclick='window.external.AddFavori
te(location.href,do

cument.title);'>

Click here to add this page to your favorites</span>
```

Dynamically set link href to same-named file on different website

```
<script type="text\javascript" langualge'javascript">

function newURL()

{

//old URL is in format "http://pages.sbcglobal.net/sjdorey/"

var tp = location.pathname; // format /sjdorey/...

var tl = tp.length;

var np = tp.slice(8);   // strip off /sjdorey

var nu = "http://www.susandoreydesigns.com" + np;

location.href = nu;

}

</script>

</head>

<body>

<p>ATTENTION.<br>

Effective immediately, this website has moved to <br>

<a href="javascript: newURL()">www.susandoreydesigns.com</a>.<br>

Click on the link to go there.</p>

</body>
```

Special Techniques: Change Text Size

Include on the page text and/or image to increase text size and to decrease text size. Each of these

has a hyperlink with an onclick event that changes the text size. The following code uses a

function resident in dw_sizerdx.js which I purloined from

http://www.newenglandancestors.org/common/js/dw_sizerdx.js

http://www.newenglandancestors.org/common/js/dw_cookies.js

<a href="" onclick="dw_fontSizerDX.adjust(0.1); return false" title="Incease

Text Size">

<a href="" onclick="dw_fontSizerDX.adjust(-0.1); return false" title="Decrease

Text Size">

. . .

<script type="text/javascript">

dw_fontSizerDX.setDefaults('em', 1, 0.7, 2.2, ['#secondary-content','#homepagecontent']);

dw_fontSizerDX.init();

</script>

Special Techniques: Viewing Series of Images on a Web Page

This was developed with file HTML/Test/test-images.html

Relevant design elements:

- page elements:

controls: [Previous] x of y [Next]

one image IMG tag

ƒ JavaScript changes content of IMG tag when user clicks [Previous] or [Next]

ƒ image file names are stored in an array, done with JavaScript

ƒ the number of images in the array, the “y” in “x of y”, is determined by JavaScript as the

number of entries in the array

ƒ first image in the array is shown when page opens

ƒ CSS defines styling

```
.photo { border: 2px solid red; }
#of { margin-left: 100px; margin-right: 100px; }
button { width: 90px; }
<script type="text/javascript"> --- in HEAD section
var picDirectory = "Photos/";
var cntPics = 5;
var a = new Array(cntPics);
a[0] = "DSC00783.JPG";
a[1] = "DSC00784.JPG";
a[2] = "DSC00785.JPG";
a[3] = "DSC00786.JPG";
a[4] = "DSC00787.JPG";
var i = 0;
var from = 0;
function fullpicURI(j)
{
var URI = picDirectory + a[j];
return URI;
}
function prev()
{
if (i == 0) return;
```

```
i--;
var fn = fullpicURI(i);
document.images["pic"].src = fn;
replaceFrom();
}
function next()
{
if (i == (cntPics - 1)) return;
i++;
var fn = fullpicURI(i);
document.images["pic"].src = fn;
replaceFrom();
}
function writexofy()
{
document.write("1 of " + cntPics);
}
function replaceFrom()
{
document.getElementById("of").innerHTML = (i + 1) + " of " + cntPics;
}
</script>
</head>
<body>
<h1>Test of images</h1>
<div class="photo">
<button onclick="prev()">Previous</button>
<span id="of">
```

```
<script type="text/javascript">
writexofy();
</script>
</span>
```



```
<button onclick="next()">Next</button>
<p>
<img name=pic src="Photos/DSC00783.JPG">
</p>
</div>
</body>
</html>
```

Special Techiniques: Asynchronicity

AJAX is the label coined to refer to for Asynchronous JavaScript and XML. Asynchronous loading of content first became practical when Java applets were introduced in the first version of the Java language in 1995. These allow compiled client-side code to load data asynchronously from the web server after a web page is loaded. In 1996, Internet Explorer introduced the IFrame element to HTML, which also enabled asynchronous loading. In 1999, Microsoft created the XMLHTTP ActiveX control in Internet Explorer 5, which was later adopted by Mozilla, Safari, Opera and other browsers as the native XMLHttpRequest object. Microsoft has adopted the native XMLHttpRequest model as of Internet Explorer 7, though the ActiveX version is still supported.

The term ″Ajax″ was coined in 2005 by Jesse James Garrett. On April 5, 2006 the World Wide Web Consortium (W3C) released the first draft specification for the object in an attempt to create an official web standard http://www.w3.org/TR/XMLHttpRequest/

Special Techniques: CopyText to Clipboard

It can be helpful to provide a tool to the user that makes it easy to copy text from a web page to

the user's Clipboard after which they can easily paste it somewhere else. Most HTML elements

have onClick and onDbleClick events. While I've seen this done with a button, like [Copy to Clipboard] it could also be done with an INPUT element (text box).

Note: There are many scripts for copying text from a Web page that work fine in Internet Explorer but not in Firefox, Netscape, or Opera. An IE-only script is very short and simple as it uses the proprietary clipboardData variable. For other browsers there is still no common solution with pure Javascript but there is one successful approach that uses a small Flash file embeded in inviibly current page.

I got the following script of the web, have not yet tried it. Mozilla-based browsers will ask the user for permission before allowing this script to access the clipboard.

Here's the HTML:

```
<textarea id='testText'>#COPYTOCLIPBOARD CODE#</textarea><br>

<button onclick='copyToClipboard(document.getElementById('testText').value);'>
```

Here's the JavaScript to make the copy:

```
function copyToClipboard(s)

{

 if( window.clipboardData && clipboardData.setData )

{

clipboardData.setData("Text", s);

}

else

{


  // You have to sign the code to enable this or allow the action in

about:config by changing

user_pref("signed.applets.codebase_principal_support", true);

netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect'
```

```
);
var clip
Components.classes['@mozilla.org/widget/clipboard;[[[[1]]]]'].createInstanc
e(Components.interfaces.nsIClipboard);
if (!clip) return;
  // create a transferable
var trans =
Components.classes['@mozilla.org/widget/transferable;[[[[1]]]]'].createInst
ance(Components.interfaces.nsITransferable);
if (!trans) return;
  // specify the data we wish to handle. Plaintext in this case.
trans.addDataFlavor('text/unicode');
  // To get the data from the transferable we need two new objects
  var str = new Object();
  var len = new Object();
            var str = Components.classes["@mozilla.org/supportsstring;
[[[[1]]]]"].createInstance(Components.interfaces.nsISupportsString);
var copytext=meintext;
str.data=copytext;
trans.setTransferData("text/unicode",str,copytext.length*[[[[2]]]]);
var clipid=Components.interfaces.nsIClipboard;
  if (!clip) return false;
  clip.setData(trans,null,clipid.kGlobalClipboard);
}
}
```

JAVA SERVLETS:

The JavaSMDeveloper ConnectionSM(JDC) presents a Short Courseon the Fundamentals of Java Servlets written by Java Softwarelicensee, the MageLang Institute. A leading provider of JavaTMtechnology training, MageLang has contributed regularly to the JDCsince 1996.The MageLang Institute, since its founding in 1995, has beendedicated to promoting the growth of the Java technologycommunity by providing excellent education and acting as anindependent resource. To find out more about MageLang's Javatechnology training, visit the MageLang web site.

Servlets are pieces of JavaTMsource code that add functionality to aweb server in a manner similar to the way applets add functionality to a browser. Servlets are designed to support a request/response computing model that is commonly used in web servers. In a request/response model, a client sends a request message to a server and the server responds by sending back a reply message. From the Java Servlet Development Kit (JSDK), you use the Java

Servlet API to create servlets for responding to requests from clients. These servlets can do many tasks, like process HTML forms with a custom servlet or manage middle-tier processing to connect to existing data sources behind a corporate firewall. In addition, servlets can maintain services, like database sessions, between requests to manage resources better than Common Gateway Interface (CGI) technologies. The Java Servlet API is based on several Java interfaces that are provided in standard Java extension (javax) packages.

In this course you will:

Learn how to use the Java Servlet API

Create and run a servlet with the JSDK

Install a servlet in Sun's Java Web ServerTM

Process parameters from HTML forms

Manage middle-tier processing

Fundamentals of JavaTMServlets:

The Java Servlet API

MageLang Institute

[Course Notes| Magercises| Module Intro]

A servlet is a JavaTMcomponent that can be plugged into a

Java-enabled web server to provide custom services. These services

can include:

New features ●

Runtime changes to content ●

Runtime changes to presentation ●

New standard protocols (such as FTP) ●

New custom protocols ●

Servlets are designed to work within a request/response processing

model. In a request/response model, a client sends a request

message to a server and the server responds by sending back a

reply message. Requests can come in the form of an

HTTP

URL,

FTP,

URL,

or a custom protocol.

The request and the corresponding response reflect the state of theclient and the server at the time of the request. Normally, the state of the client/server connection cannot be maintained across different request/response pairs. However, session information is maintainable with servlets through means to be described later.

The Java Servlet API includes several Java interfaces and fully defines the link between a hosting server and servlets. The Servlet API is defined as an extension to the standard JDK. JDK extensions are packaged under javax--the root of the Java extension library tree. The Java Servlet API contains the following packages: Package javax.servlet

Package javax.servlet.http

Servlets are a powerful addition to the Java environment. They are fast, safe, reliable, and 100% pure Java. Because servlets plug into an existing server, they leverage a lot of existing code and

The Java Servlet API technology. The server handles the network connections, protocol negotiation, class loading, and more; all of this work does not need to be replicated! And, because servlets are located at the middle tier, they are positioned to add a lot of value and flexibility to a system.

In this course you will learn about the Servlet API and you will get a

brief tour of the types of features servlets can implement.

Architectural Roles for Servlets

Because of their power and flexibility, servlets can play a significantrole in a system architecture. They can perform the application processing assigned to the middle tier, act as a proxy for a client,and even augment the features of the middle tier by adding support for new protocols or other features. A middle tier acts as the application server in so called three-tier client/server systems, positioning itself between a lightweight client like a web browser and a data source.

Middle-Tier Process

In many systems a middle tier serves as a link between clients and back-end services. By using a middle tier a lot of processing can be off-loaded from both clients (making them lighter and faster) and servers (allowing them to focus on their mission). One advantage of middle tier processing is simply connection management. A set of servlets could handle connections with hundreds of clients, if not thousands, while recycling a pool of expensive connections to database servers.

Other middle tier roles include:

Business rule enforcement

Transaction management

Mapping clients to a redundant set of servers

Supporting different types of clients such as pure HTML and

Java capable clients

Proxy Servers

When used to support applets, servlets can act as their proxies.This can be important because Java security allows applets only tomake connections back to

the server from which they were loaded.If an applet needs to connect to a database server located on adifferent machine, a servlet can make this connection on behalf of the applet.

Protocol Support

The Servlet API provides a tight link between a server and servlets.This allows servlets to add new protocol support to a server. (Youwill see how HTTP support is provided for you in the API packages.)Essentially, any protocol that follows a request/response computing

The Java Servlet API

model can be implemented by a servlet. This could include:

SMTP ●

POP ●

FTP ●

Servlet support is currently available in several web servers, andwill probably start appearing in other types of application servers inthe near future. You will use a web server to host the servlets inthis class and only deal with the HTTP protocol.Because HTTP is one of the most common protocols, and becauseHTML can provide such a rich presentation of information, servletsprobably contribute the most to building HTTP based systems.

HTML Support

HTML can provide a rich presentation of information because of itsflexibility and the range of content that it can support. Servlets canplay a role in creating HTML content. In fact, servlet support forHTML is so common, the javax.servlet.httppackage is dedicated tosupporting HTTP protocol and HTML generation.Complex web sites often need to provide HTML pages that aretailored for each visitor, or even for each hit. Servlets can bewritten to process HTML pages and customize them as they are sentto a client. This can be as simple as on the fly substitutions or it canbe as complex as compiling a grammar-based description of a pageand generating custom HTML.

Inline HTML Generation

Some web servers, such as the Java Web ServerTM(JWS), allowservlet tags to be embedded directly into HTML files. When theserver encounters such a tag, it calls the servlet while it is sendingthe HTML file to the client. This allows a servlet to insert itscontribution directly into the outgoing HTML stream.

Server-Side Includes

Another example is on the fly tag processing known as server-sideincludes(SSI). With SSI, an HTML page can contain specialcommands that are processed each time a page is requested.Usually a web server requires HTML files that incorporate SSI to usea unique extension, such as .shtml. As an example, if an HTML page(with an .shtmlextension) includes the following: <!--#include

virtual="/includes/page.html"-->

it would be detected by the web server as a request to perform aninline file include. While server side includes are supported by mostweb servers, the SSI tags are not standardized.Servlets are a great way to add server side include processing to aweb server. With more and more web servers supporting servlets, it would be possible to write a standard SSI processing servlet anduse it on different web servers.The Java Servlet API

Replacing CGI Scripts

An HTTP servlet is a direct replacement for Common GatewayInterface (CGI) scripts. HTTP servlets are accessed by the userentering a URL in a browser or as the target of an HTML formaction. For example, if a user enters the following URL into a browser address field, the browser requests a servlet to send an

HTML page with the current time: http://localhost/servlet/DateTimeServlet

The DateTimeServletresponds to this request by sending an HTML page to the browser. Note that these servlets are not restricted to generating web pages; they can perform any other function, such as storing and fetching database information, or opening a socket to another machine.

Installing Servlets

Servlets are not runin the same sense as applets and applications.Servlets provide functionality that extends a server. In order to test a servlet, two steps are required:Install the servlet in a hosting server1.

Request a servlet's service via a client request2.

There are many web servers that support servlets. It is beyond thescope of this course to cover the different ways to install servlets ineach server. This course examines the JSDK's servletrunnerutility andthe JWS.

Temporary versus Permanent Servlets

Servlets can be started and stopped for each client request, or they can be started as the web server is started and kept alive until theserver is shut down. Temporary servlets are loaded on demand andoffer a good way to conserve resources in the server for less-usedfunctions.

Permanent servlets are loaded when the server is started, and liveuntil the server is shutdown. Servlets are installed as permanentextensions to a server when their start-up costs are very high (suchas establishing a connection with a DBMS), when they offer permanent server-side functionality (such as an RMI service), or

when they must respond as fast as possible to client requests.There is no special code necessary to make a servlet temporary or permanent; this is a function of the server configuration. Because servlets can be loaded when a web server starts, they can use this auto-loading mechanism to provide easier loading of server-side Java programs. These programs can then provide functionality that is totally unique and independent of the web

server. For example, a servlet could provide R-based services

(rlogin, rsh, ...) through TCP/IP ports while using the servlet

request/response protocol to present and process HTML pages used

to manage the servlet.

Using servletrunner

The Java Servlet API For both JDK 1.1 and the Java 2 platform, you need to install the

Java Servlet Development Kit (JSDK). To use servletrunner, make

sure your PATHenvironment variable points to its directory. For the

JSDK 2.0 installed with all default options, that location is:

c:\jsdk2.0\binon a Windows platform.

To make sure that servletrunnerhas access to the Java servlet

packages, check that your CLASSPATHenvironment variable is

pointing to the correct JAR file, c:\jsdk2.0\lib\jsdk.jaron a Windows

platform. With the Java 2 platform, instead of modifying the

CLASSPATH, it is easier to just copy the JAR file to the extdirectory

under the Java runtine environment. This treats the servlet

packages as standard extensions.

With the environment set up, run the servletrunnerprogram from the

command line. The parameters are:

Usage: servletrunner [options]

Options:

-p port the port number to listen on

-b backlog the listen backlog

-m max maximum number of connection handlers

-t timeout connection timeout in milliseconds

-d dir servlet directory

-s filename servlet property file name

The most common way to run this utility is to move to the directory

that contains your servlets and run servletrunnerfrom that location.

However, that doesn't automatically configure the tool to load the

servlets from the current directory.

Magercise

Hosting with servletrunner 1.

Using Java Web Server

Sun's Java Web Server (JWS) is a full featured product. For servletdevelopers, a nice feature is its ability to detect when a servlet hasbeen updated. It detects when new class files have been copied tothe appropriate servlet directory and, if necessary, automaticallyreloads any running servlets.The JWS can be installed as a service under Windows NT. While thismakes it convenient for running a production server, it is notrecommended for servlet development work. Under Windows 95,there are no OS services, so the command line start-up is your onlyoption.To run JWS from the c:\JavaWebServer1.1\bindirectory, type in thehttpdcommand. This starts the server in a console window. Nofurther display is shown in the console unless a servlet executes a

System.out.println()statement.

Servlets are installed by moving them to the

c:\JavaWebServer1.1\servletsdirectory. As mentioned, JWS detects

when servlets have been added to this directory. Although you can use the JWS management applet to tailor the servlet installation, this is generally not advised except for production server installations.To shut down the JWS, press <Control>+C in the command window. The server prints a message to the console when it has finished shutting down.

Magercise

Hosting with the Java Web Server 2.

Servlet API

The Java Servlet API defines the interface between servlets and servers. This API is packaged as a standard extension to the JDK under javax:

Package javax.servlet

Package javax.servlet.http

The API provides support in four categories:

- Servlet life cyclemanagement
- Access to servlet context
- Utility classes
- HTTP-specific support classes
- The Servlet Life Cycle

Servlets run on the web server platform as part of the same processas the web server itself. The web server is responsible for initializing, invoking, and destroying each servlet instance.A web server communicates with a servlet through a simpleinterface, javax.servlet.Servlet. This interface consists of three main methods:

- init()
- service()
- destroy()
- and two ancillary methods:

getServletConfig()

getServletInfo()

You may notice a similarity between this interface and that of Javaapplets. This is by design! Servlets are to web servers what appletsare to web browsers.An applet runs in a web browser, performingactions it requests through a specific interface. A servlet does thesame, running in the web server.

The init()Method

When a servlet is first loaded, its init()method is invoked. Thisallows the servlet to per form any setup processing such as opening

The Java Servlet API

files or establishing connections to their servers. If a servlet hasbeen permanently installed in a server, it loads when the serverstarts to run. Otherwise, the server activates a servlet when itreceives the first client request for the services provided by theservlet.The init()method is guaranteed to finish before any other calls aremade to the servlet--such as a call to the service()method. Note thatinit()will only be called once; it will not be called again unless theservlet has been unloaded and then reloaded by the server.The init()method takes one argument, a reference to a ServletConfigobject which provides initialization arguments for the servlet. Thisobject has a method getServletContext()that returns a ServletContextobject containing information about the servlet's environment (seethe discussion on Servlet Initialization Contextbelow).

The service()Method

The service()method is the heart of the servlet. Each requestmessage from a client results in a single call to the servlet's service()method. The service()method reads the request and produces theresponse message from its two parameters:A ServletRequestobject with data from the client. The dataconsists of name/value pairs of parameters and an InputStream.Several methods are provided that return the client's parameter information. The InputStreamfrom the client can beobtained via the getInputStream()method. This method returns aServletInputStream, which can be used to get additional datafrom the client. If you are interested in processingcharacter-level data instead of byte-level data, you can get a

BufferedReaderinstead with getReader().

●

A ServletResponserepresents the servlet's reply back to theclient. When preparing a response, the method setContentType()is called first to set the MIME

type of the reply. Next, the method getOutputStream()or getWriter()can be used to obtain a ServletOutputStreamor PrintWriter, respectively, to send data back

to the client.

As you can see, there are two ways for a client to send information to a servlet. The first is to send parameter values and the second is to send information via the InputStream(or Reader). Parameter values can be embedded into a URL. How this is done is discussed below. How the parameter values are read by the servlet is discussed later. The service()method's job is conceptually simple--it creates aresponse for each client request sent to it from the host server.However, it is important to realize that there can be multiple service requests being processed at once. If your service method requires any outside resources, such as files, databases, or some external data, you must ensure that resource access is thread-safe. Making your servlets thread-safe is discussed in a later sectionof this

The destroy()Method

The destroy()method is called to allow your servlet to clean up any resources (such as open files or database connections) before the servlet is unloaded. If you do not require any clean-up operations, this can be an empty method. The server waits to call the destroy()method until either all service calls are complete, or a certain amount of time has passed. This means that the destroy()method canbe called while some longer-running service()methods are still running. It is important that you write your destroy()method to avoid closing any necessary resources until all service()calls have completed.

Sample Servlet

The code below implements a simple servlet that returns a static HTML page to a browser. This example fully implements the Servlet

interface.

```
import java.io.*;

import javax.servlet.*;

public SampleServlet implements Servlet {

private ServletConfig config;

public void init (ServletConfig config)

throws ServletException {

this.config = config;

}
```

```
public void destroy() {} // do nothing
public ServletConfig getServletConfig() {
return config;
}
public String getServletInfo() {
return "A Simple Servlet";
}
public void service (ServletRequest req,
ServletResponse res
) throws ServletException, IOException {
res.setContentType( "text/html" );
PrintWriter out = res.getWriter();
out.println( "<html>" );
out.println( "<head> );
out.println( "<title>A Sample Servlet</title>" );
out.println( "</head>" );
out.println( "<body>" );
out.println( "<h1>A Sample Servlet</h1>" );
out.println( "</body>" );
out.println( "</html>" );
out.close();
}
}
```

Servlet Context

A servlet lives and dies within the bounds of the server process. To understand its operating environment, a servlet can get information about its environment at different times. Servlet initialization information is available during servlet start-up; information about the hosting server is available at any time; and each service request can contain specific contextual information.

Servlet Initialization Information

Initialization information is passed to the servlet via the ServletConfig parameter of the init()method. Each web server provides its ownway to pass initialization information to a servlet. With the JWS, if a servlet class DatePrintServlettakes an initialization argument timezone, you would define the following properties in a servlets.propertiesfile:

servlet.dateprinter.code=DatePrinterServlet

servlet.dateprinter.timezone=PST

or this information could be supplied through a GUI administration

tool.

The timezone information would be accessed by the servlet with the

following code:

String timezone;

public void init(ServletConfig config) {

timeZone = config.getInitParameter("timezone");

}

An Enumerationof all initialization parameters is available to the

servlet via the getInitParameterNames()method.

Server Context Information

Server context information is available at any time through the ServletContextobject. A servlet can obtain this object by calling the getServletContext()method on the ServletConfigobject. Remember that

this was passed to the servlet during the initialization phase. A well

written init()method saves the reference in a private variable.

The ServletContextinterface defines several methods. These are

outlined below.

getAttribute()

An extensible way to get information about a

server via attribute name/value pairs. This is

server specific.

getMimeType() Returns the MIME type of a given file.

getRealPath()

This method translates a relative or virtual

path to a new path relative to the server's

HTML documentation root location.

getServerInfo()

Returns the name and version of the network

service under which the servlet is running.

getServlet()

Returns a Servletobject of a given name.

Useful when you want to access the services

of other servlets.

getServletNames()

Returns an enumeration of servlet names

available in the current namespace.

log()

Writes information to a servlet log file. The

log file name and format are server specific.

The following example code shows how a servlet uses the host

server to write a message to a servlet log when it initializes:

```
private ServletConfig config;
public void init(ServletConfig config) {
// Store config in an instance variable
this.config = config;
ServletContext sc = config.getServletContext();
sc.log( "Started OK!" );
}
```

Servlet Context During a Service Request

Each service request can contain information in the form of name/value parameter pairs, as a ServletInputStream, or a BufferedReader. This information is available from the ServletRequest object that is passed to the service()method.

The following code shows how to get service-time information:

```
BufferedReader reader;
String param1;
String param2;
public void service (
ServletRequest req,
ServletResponse res) {
reader = req.getReader();
param1 = req.getParameter("First");
param2 = req.getParameter("Second");
}
```

There are additional pieces of information available to the servlet through ServletRequest. These are shown in the following table.

getAttribute()	Returns value of a named attribute for this request.
getContentLength()	Size of request, if known.
getContentType()	Returns MIME type of the request message body.
getInputStream()	Returns an InputStreamfor reading binary data from the body of the request message.

getParameterNames()

Returns an array of strings with the names of all parameters.

getParameterValues()

Returns an array of strings for a specific parameter name.

getProtocol()

Returns the protocol and version for the request as a string of the form <protocol>/<major version>.<minor version>.

getReader()

Returns a BufferedReaderto get the text from the body of the request message.

getRealPath()

Returns actual path for a specified virtual path.

getRemoteAddr()

IP address of the client machine sending this request.

getRemoteHost()

Host name of the client machine that sent this request.

getScheme()

Returns the scheme used in the URL for this request (for example, https, http, ftp, etc.).

getServerName()

Name of the host server that received this

request.

getServerPort()

Returns the port number used to receive

this request.

The following Magercise shows you how to extract parameters from

a service request.

Magercise

Accessing Servlet Service-Time Parameters 3.

Utility Classes

There are several utilities provided in the Servlet API. The first is the interface javax.servlet.SingleThreadModelthat can make it easier to write simple servlets. If a servlet implements this marker interface, the hosting server knows that it should never call the servlet's

service()method while it is processing a request. That is, the server processes all service requests within a single thread. While this makes it easier to write a servlet, this can impede performance. A full discussion of this issue is located laterin this course.

Two exception classes are included in the Servlet API. The exception javax.servlet.ServletExceptioncan be used when there is a general failure in the servlet. This notifies the hosting server that there is a problem.

The exception javax.servlet.UnavailableExceptionindicates that a servlet

is unavailable. Servlets can report this exception at any time. There

are two types of unavailability:

Permanent. The servlet is unable to function until an

administrator takes some action. In this state, a servlet should

write a log entry with a problem report and possible

resolutions.

Temporary. The servlet encountered a (potentially) temporary problem, such as a full disk, failed server, etc. The problem can correct itself with time or may require operator intervention.

HTTP Support

Servlets that use the HTTP protocol are very common. It should not be a surprise that there is specific help for servlet developers who write them. Support for handling the HTTP protocol is provided in the package javax.servlet.http. Before looking at this package, take a look at the HTTP protocol itself. HTTP stands for the HyperText Transfer Protocol. It defines a protocol used by web browsers and servers to communicate with each other. The protocol defines a set of text-based request messages called HTTP methods. (Note: The HTTP specification calls these HTTP methods; do not confuse this term with Java methods.

Think of HTTP methodsas messages requesting a certain type of

respoNse). The HTTP methods include:

GET

HEAD

POST

PUT

DELETE

TRACE

CONNECT

OPTIONS

For this course, you will only need to look at only three of these

methods: GET, HEAD, and POST.

The HTTP GETMethod

The HTTP GETmethod requests information from a web server. Thisinformation could be a file, output from a device on the server, oroutput from a program (such as a servlet or CGI script).

An HTTP GETrequest takes the form:

GET URL <http version>

Host: <target host>

in addition to several other lines of information.

For example, the following HTTP GETmessage is requesting the

home page from the MageLang web site:

```
GET / HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (
compatible;
MSIE 4.01;
Windows NT)
Host: www.magelang.com
Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg
```

On most web servers, servlets are accessed via URLs that start with

/servlet/. The following HTTP GETmethod is requesting the servlet

MyServleton the host www.magelang.com:

```
GET /servlet/MyServlet?name=Scott&
company=MageLang%20Institute HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (
compatible;
MSIE 4.01;
Windows NT)
Host: www.magelang.com
Accept: image/gif, image/x-xbitmap,
image/jpeg, image/pjpeg
```

The URL in this GETrequest invokes the servlet called MyServletand

contains two parameters, nameand company. Each parameter is a

name/value pair following the format name=value. The parametersare specified by following the servlet name with a question mark('?'), with each parameter separated by an ampersand ('&').Note the use of %20in the company's value. A space would signalthe end of the URL in the GETrequest line, so it must be "URLencoded", or replaced with %20instead. As you will see later,servlet developers do not need to worry about this encoding as itwill be automatically decoded by the HttpServletRequestclass.HTTP GETrequests have an important limitation. Most web serverslimit how much data can be passed as part of the URL name(usually a few hundred bytes.) If more data must be passedbetween the client and the server, the HTTP POSTmethod should beused instead.It is important to note that the server's handling of a GETmethod isexpected to be safeand idempotent. This means that a GETmethodwill not cause any side effects and that it can be executedrepeatedly.When a server replies to an HTTP GETrequest, it sends an HTTPresponse message back. The header of an HTTP response looks like

the following:

HTTP/1.1 200 Document follows

Date: Tue, 14 Apr 1997 09:25:19 PST

Server: JWS/1.1

Last-modified: Mon, 17 Jun 1996 21:53:08 GMT

Content-type: text/html

Content-length: 4435

<4435 bytes worth of data -- the document body>

The HEADMethod

The HTTP HEADmethod is very similar to the HTTP GETmethod. The

request looks exactly the same as the GETrequest (except the word

HEADis used instead of GET), but the server only returns the header

information.

HEADis often used to check the following:

The last-modified date of a document on the server for caching

purposes

The size of a document before downloading (so the browser

can present progress information)

The server type, allowing the client to customize requests for

that server

The type of the requested document, so the client can be sure

it supports it

Note that HEAD, like GET, is expected to be safeand idempotent.

The POSTMethod

An HTTP POSTrequest allows a client to send data to the server.

This can be used for several purposes, such as

Posting information to a newsgroup

Adding entries to a web site's guest book

Passing more information than a GETrequest allows

Pay special attention to the third bullet above. The HTTP GETrequest passes all its arguments as part of the URL. Many webservers have a limit to how much data they can accept as part ofthe URL. The POSTmethod passes all of its parameter data in aninput stream, removing this limit.

A typical POSTrequest might be as follows:

POST /servlet/MyServlet HTTP/1.1

User-Agent: Mozilla/4.0 (

compatible;

MSIE 4.01;

Windows NT)

Host: www.magelang.com

Accept: image/gif, image/x-xbitmap,

image/jpeg, image/pjpeg, */

Content-type: application/x-www-form-urlencoded

Content-length: 39

name=Scott&company=MageLang%20Institute

Note the blank line--this signals the end of the POSTrequest header

and the beginning of the extended information.

Unlike the GETmethod, POSTis notexpected to be safenor

idempotent; it can perform modifications to data, and it is not

required to be repeatable.

HTTP Support Classes

Now that you have been introduced to the HTTP protocol, considerhow the javax.servlet.httppackage helps you write HTTP servlets. Theabstract class javax.servlet.http.HttpServletprovides an implementationof the javax.servlet.Servletinterface and includes a lot of helpfuldefault functionality.The easiest way to write an HTTP servlet is toextend HttpServletand add your own custom processing.The class HttpServletprovides an implementation of the service()method that dispatches the HTTP messages to one of severalspecial methods. These methods are:

doGet()

doHead()

doDelete()

doOptions()

doPost()

doTrace()

and correspond directly with the HTTP protocol methods.As shown in the following diagram, the service()method interpretseach HTTP method and determines if it is an HTTP GET, HTTP POST,

HTTP HEAD, or other HTTP protocol method:

The class HttpServletis actually rather intelligent. Not only does itdispatch HTTP requests, it detects which methods are overridden in a subclass and can report back to a client on the capabilities of the server. (Simply by overriding the doGet()method causes the class to respond to an HTTP OPTIONSmethod with information that GET, HEAD, TRACE, and OPTIONSare all supported. These capabilities are in fact all supported by the class's code).

In another example of the support HttpServletprovides, if the doGet() method is overridden, there is an automatic response generated for the HTTP HEADmethod. (Since the response to an HTTP HEAD method is identical to an HTTP

GETmethod--minus the body of the message--the HttpServletclass can generate an appropriate response to an HTTP HEADrequest from the reply sent back from the doGet() method). As you might expect, if you need more precise control,

you can always override the doHead()method and provide a custom response.

Using the HTTP Support Classes

When using the HTTP support classes, you generally create a new servlet that extends HttpServletand overrides either doGet()or doPost(), or possibly both. Other methods can be overridden to get more fine-grained control. The HTTP processing methods are passed two parameters, an HttpServletRequestobject and an HttpServletResponseobject. The HttpServletRequestclass has several convenience methods to help parse the request, or you can parse it yourself by simply reading the text of the request.

A servlet's doGet()method should Read request data, such as input parameters

Set response headers (length, type, and encoding)

Write the response data

It is important to note that the handling of a GETmethod is

expected to be safeand idempotent.

Handing is considered safeif it does not have anyside effects for which users are held responsible, such as charging them for the access or storing data.

Handling is considered idempotentif it can safely be repeated.

This allows a client to repeat a GETrequest without penalty.

Think of it this way: GETshould be "looking without touching." If

you require processing that has side effects, you should use another

HTTP method, such as POST.

A servlet's doPost()method should be overridden when you need to process an HTML form posting or to handle a large amount of data being sent by a client. HTTP POSTmethod handling is discussed in detail later. HEADrequests are processed by using the doGet()method of an HttpServlet. You could simply implement doGet()and be done with it; any document data that you write to the response output stream will notbe returned to the client. A more efficient implementation, however, would check to see if the request was a GETor HEAD request, and if a HEADrequest, not write the data to the response output stream.

Summary

The Java Servlet API is a standard extension. This means that there is an explicit definition of servlet interfaces, but it is not part of the Java Development Kit (JDK) 1.1 or the Java 2 platform. Instead, the servlet classes are delivered with the Java Servlet Development Kit

(JSDK) version 2.0 from Sun

(http://java.sun.com/products/servlet/). This JSDK version is

intended for use with both JDK 1.1 and the Java 2 platform. There

are a few significant differences between JSDK 2.0 and JSDK 1.0.

See belowfor details. If you are using a version of JSDK earlier

than 2.0, it is recommended that you upgrade to JSDK 2.0.

Servlet support currently spans two packages:

javax.servlet: General Servlet Support

Servlet

An interface that defines communication between a web server and a servlet. This interface defines the init(), service(), and destroy()methods (and a few others).

ServletConfig

An interface that describes the configuration parameters for a servlet. This is passed to the servlet when the web server calls its init()method. Note that the

servlet should save the reference to the ServletConfigobject, and define a getServletConfig()method to return it when asked. This interface defines how to get the initialization parameters for the servlet and the context under which the servlet is running.

ServletContext

An interface that describes how a servlet can get information about the server in which it is running. It can be retrieved via the getServletContext()method of the

ServletConfigobject.

ServletRequest

An interface that describes how to get information about a client request.

ServletResponse An interface that describes how to pass

information back to the client.

GenericServlet

A base servlet implementation. It takes care of saving the ServletConfigobject reference, and provides several methods that delegate their functionality to the ServletConfigobject. It also provides a dummy implementation for init()and destroy().

ServletInputStream

A subclass of InputStreamused for reading the data part of a client's request. It adds a readLine()method for convenience.

ServletOutputStream

An OutputStreamto which responses for the client are written.

ServletException Should be thrown when a servlet problem is encountered.

UnavailableException Should be thrown when the servlet is unavailable for some reason.

javax.servlet.http: Support for HTTP Servlets

HttpServletRequest

A subclass of ServletRequestthat defines several methods that parse HTTP request headers.

HttpServletResponse

A subclass of ServletResponsethat provides access and interpretation of HTTP status codes and header information.

HttpServlet

A subclass of GenericServletthat provides automatic separation of HTTP request by method type. For example, an HTTP GET request will be processed by the service() method and passed to a doGet()method.

HttpUtils

A class that provides assistance for parsing HTTP GETand POSTrequests. Servlet Examples

Now for an in-depth look at several servlets. These examples include:

Generating Inline Content

Processing HTTP Post Requests

Using Cookies

Maintaining Session Information

Connecting to Databases

Generating Inline Content

Sometimes a web page needs only a small piece of information that is customized at runtime. The remainder of a page can be static information. To substitute only small amounts of information, some web servers support a concept known as server-side includes, or SSI. If it supports SSI, the web server designates a special file extension (usually .shtml) which tells the server that it should look for SSI tags in the requested file. The JWS defines a special SSI tag called the

<servlet>tag, for example:

<servlet code="DatePrintServlet">

<param name=timezone value=pst>

</servlet>

This tag causes the invoking of a servlet named DatePrintServletto generate some in-line content.

SSI and servlets allow an HTML page designer to write a skeleton for a page, using servlets to fill in sections of it, rather than require the servlet to generate the entire page. This is very useful for features like page-hit counters and other small pieces of functionality. The DatePrintServletservlet works just like a regular servlet except that it is designed to provide a very small response and not a complete HTML page. The output MIME type gets set to "text/plain" and not "text/html". Keep in mind that the syntax of server-side includes, if they are even

supported, may vary greatly from one web server to another. In the following Magercise you create the DatePrintServletand see how to use it in an HTML page.

Processing HTTP Post Requests

HTTP POSTmethod processing differs from HTTP GETmethod processing in several ways. First, because POSTis expected to modify data on the server, there can be a need to safely handle updates coming from multiple clients at the same time. Second, because the size of the information stream sent by the client can be very large, the doPost()method must open an InputStream(or Reader) from the client to get any of the information. HTTP POSTdoes not support sending parameters encoded inside of the URL as does the

HTTP GETmethod.

The problem of supporting simultaneous updates from multiple clients has been solved by database systems (DBMSs); unfortunately the HTTP protocol does not work well with database systems. This is because DBMSs need to maintain a persistent connection between a client and the DBMS to determine which client is trying to update the data. The HTTP protocol does not support this type of a connection as it is a message based, stateless protocol. Solving this problem is not easy and never elegant! Fortunately, the Servlet API defines a means to track client/server sessions. This is covered in theMaintaining Session Informationsection later in this course. Without session management tracking, one can resort to several different strategies. They all involve writing data to the client in hidden fieldswhich is then sent back to the server. The simplest way to handle updates is to use an optimistic locking scheme based on date/time stamps. One can use a single date/time stamp for a whole form of data, or one could use separate date/time stamps for each "row" of information. Once the update strategy has been selected, capturing the data sent to the server via an HTTP POSTmethod is straightforward. Information from the HTML form is sent as a series of parameters

(name/value pairs) in the InputStreamobject. The HttpUtilsclass contains a method parsePostData()that accepts the raw InputStreamfrom the client and return a Hashtablewith the parameter information already processed. A really nice feature is that if a parameter of a given name has multiple values (such is the case for a column name with multiple rows), then this information can be retrieved from the Hashtableas an array of type String.In the following Magercise, you will be given skeleton code that implements a pair of servlets that display data in a browser as an editable HTML form. The structure of the data is kept separate from the actual data. This makes it easy to modify this code to run against arbitrary tables from a JDBC-connected database.

Using Cookies

For those unfamiliar with cookies, a cookie is a named piece of datamaintained by a browser, normally for session management. SinceHTTP connections are stateless, you can use a cookie to store persistent information accross multiple HTTP connections. The Cookie class is where all the "magic" is done. The HttpSessionclass, described next, is actually easier to use. However, it doesn't support retaining the information across multiple browser sessions. To save cookie information you need to create a Cookie, set the content type of the HttpServletResponseresponse, add the cookie to the response, and then send the output. You must add the cookie after setting the content type, but before sending the output, as the

cookie is sent back as part of the HTTP response header.

```
private static final String SUM_KEY = "sum";

...

int sum = ...; // get old value and add to it

Cookie theCookie =

new Cookie (SUM_KEY, Integer.toString(sum));

response.setContentType("text/html");

response.addCookie(theCookie);
```

It is necessary to remember that all cookie data are strings. You

must convert information like intdata to a String object. By default,

the cookie lives for the life of the browser session. To enable a cookie to live longer, you must call the setMaxAge(interval)method. When positive, this allows you to set the number of seconds a cookie exists. A negative setting is the default and destroys the cookie when the browser exits. A zero setting immediately deletes the cookie. Retrieving cookie data is a little awkward. You cannot ask for the cookie with a specific key. You must ask for all cookies and find the specific one you are interested in. And, it is possible that multiple cookies could have the same name, so just finding the first setting is not always sufficient. The following code finds the setting of a

single-valued cookie:

```
int sum = 0;

Cookie theCookie = null;
```

```
Cookie cookies[] = request.getCookies();
if (cookies != null) {
for(int i=0, n=cookies.length; i < n; i++) {
theCookie = cookies[i];
if (theCookie.getName().equals(SUM_KEY)) {
try {
sum = Integer.parseInt(theCookie.getValue());
} catch (NumberFormatException ignored) {
sum = 0;
}
break;
}
}
}
```

The complete code example shown above is availablefor testing.

Maintaining Session Information

A session is a continuous connection from the same browser over a fixed period of time. (This time is usually configurable from the web server. For the JWS, the default is 30 minutes.) Through the implicit use of browser cookies, HTTP servlets allow you to maintain session information with the HttpSessionclass. The HttpServletRequestprovides the current session with the getSession(boolean)method. If the boolean parameter is true, a new session will be created when a new session is detected. This is, normally, the desired behavior. In the event the parameter is false, then the method returns nullif a new session is detected.

```
public void doGet (HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
HttpSession session = request.getSession(true);
// ...
```

Once you have access to an HttpSession, you can maintain a

collection of key-value-paired information, for storing any sort of session-specific data. You automatically have access to the creation time of the session with getCreationTime()and the last accessed time with getLastAccessedTime(), which describes the time the last servlet request was sent for this session. To store session-specific information, you use the putValue(key, value) method. To retrieve the information, you ask the session with getValue(key). The following example demonstrates this, by continually summing up the integer value of the Addendparameter. In the event the value is not an integer, the number of errors are also counted.

```
private static final String SUM_KEY =

"session.sum";

private static final String ERROR_KEY =

"session.errors";

Integer sum = (Integer) session.getValue(SUM_KEY);

int ival = 0;

if (sum != null) {

ival = sum.intValue();

}

try {

String addendString =

request.getParameter("Addend");

int addend = Integer.parseInt (addendString);

sum = new Integer(ival + addend);

session.putValue (SUM_KEY, sum);

} catch (NumberFormatException e) {

Integer errorCount =

(Integer)session.getValue(ERROR_KEY);

if (errorCount == null) {

errorCount = new Integer(1);

} else {

errorCount = new Integer(errorCount.intValue()+1);
```

```
}
session.putValue (ERROR_KEY, errorCount);
}
```

As with all servlets, once you've performed the necessary operations, you need to generate some output. If you are using sessions, it is necessary to request the session with HttpServletRequest.getSession()before generating any output. response.setContentType("text/html");

```
PrintWriter out = response.getWriter();
out.println("<html>" +
"<head><title>Session Information</title></head>" +
"<body bgcolor=\"#FFFFFF\">" +
"<h1>Session Information</h1><table>");
out.println ("<tr><td>Identifier</td>");
out.println ("<td>" + session.getId() + "</td></tr>");
out.println ("<tr><td>Created</td>");
out.println ("<td>" + new Date(
session.getCreationTime()) + "</td></tr>");
out.println ("<tr><td>Last Accessed</td>");
out.println ("<td>" + new Date(
session.getLastAccessedTime()) + "</td></tr>");
out.println ("<tr><td>New Session?</td>");
out.println ("<td>" + session.isNew() + "</td></tr>");
String names[] = session.getValueNames();
for (int i=0, n=names.length; i<n; i++) {
out.println ("<tr><td>" + names[i] + "</td>");
out.println ("<td>" + session.getValue (names[i])
+ "</td></tr>");
out.println("</table></center></body></html>");
out.close();
```

The complete code example shown above is availablefor testing. One thing not demonstrated in the example is the ability to enda session, where the next call to request.getSession(true)returns a different session. This is done with a call to invalidate(). In the event a user has browser cookies disabled, you can encode the session ID within the HttpServletResponseby calling its encodeUrl() method.

Connecting to Databases

It is very common to have servlets connect to databases through JDBC. This allows you to better control access to the database by only permitting the middle-tier to communicate with the database.If your database server includes sufficient simultanious connection licenses, you can even setup database connections once, when the servlet is initialized, and pool the connections between all the different service requests. The following demonstrates sharing a single Connectionbetween all service requests. To find out how many simultaneous connections the driver supports, you can ask its DatabaseMetaDataand then create a pool of Connectionobjects to share between service requests. In the init()method connect to the database.

```
Connection con = null;

public void init (ServletConfig cfg)

throws ServletException {

super.init (cfg);

// Load driver

String name = cfg.getInitParameter("driver");

Class.forName(name);

// Get Connection

con = DriverManager.getConnection (urlString);

}
```

In the doGet()method retrieve database information.

```
public void doGet (HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

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

```
// Have browser ignore cache - force reload
response.setHeader ("Expires",
"Mon, 01 Jan 1990 00:00:00 GMT");
Statement stmt = null;
ResultSet result = null;
try {
// Submit query
```

●

The Java Servlet API


```
stmt = con.createStatement();
result = stmt.executeQuery (
"SELECT programmer, cups " +
"FROM JoltData ORDER BY cups DESC;");
// Create output
PrintWriter out = response.getWriter();
while(result.next()) {
// Generate output from ResultSet
}
} finally {
if (result != null) {
result.close();
}
if (stmt != null) {
stmt.close();
}
}
```

```
out.flush();

out.close();

}
```

In the destroy()method disconnect from the database.

```
public void destroy() {

super.destroy();

con.close();

}
```

It is not good practice to leave a database connection permanently open, so this servlet should not be installed as a permanent servlet. Having it as a temporary servlet that closes itself down after a predefined period of inactivity allows the sharing of the database connection with requests that coincide, reducing the cost of each request. You can also save some information in the HttpSessionto possible page through the result set.

Security Issues

As with Java applets, Java servlets have security issues to worry

about, too.

The Servlet Sandbox

A servlet can originate from several sources. A webmaster may have written it; a user may have written it; it may have been bought as part of a third-party package or downloaded from another web site. Based on the source of the servlet, a certain level of trust should be associated with that servlet. Some web servers provide a means to associate different levels of trust with different servlets. This concept is similar to how web browsers control applets, and is known as "sandboxing".

A servlet sandbox is an area where servlets are given restricted authority on the server. They may not have access to the file system or network, or they may have been granteda more trusted status. It is up to the web server administrator to decide which servlets are granted this status. Note that a fully trusted servlet has full access to the server's file system and networking capabilities. It could even perform a System.exit(), stopping the web server...

Access Control Lists (ACLs)

Many web servers allow you to restrict access to certain web pages and servlets via access control lists (ACLs). An ACL is a list of users who are allowed to perform a specific function in the server. The list specifies:

What kind of access is allowed

What object the access applies to

Which users are granted access

Each web server has its own means of specifying an ACL, but in general, a list of users is registered on the server, and those user names are used in an ACL. Some servers also allow you to add users to logical groups, so you can grant access to a group of users without specifying all of them explicitly in the ACL. ACLs are extremely important, as some servlets can present or modify sensitive data and should be tightly controlled, while others only present public knowledge and do not need to be controlled.

Threading Issues

A web server can call a servlet's service()method for several requests at once. This brings up the issue of thread safety in servlets.

But first consider what you do notneed to worry about: a servlet's

init()method. The init()method will onlybe called oncefor the duration of the time that a servlet is loaded. The web server calls init()when loading, and will not call it again unless the servlet has been unloaded and reloaded. In addition, the service()method or

destroy()method will notbe called until the init()method has

completed its processing. Things get more interesting when you consider the service()method. The service()method can be called by the web server for multiple clients at the same time. (With the JSDK 2.0, you can tag a servlet with the SingleThreadModelinterface. This results in each call to service()being handled serially. Shared resources, such as files and databases, can still have concurrency issues to handle.) If your service()method uses outside resources, such as instance data from the servlet object, files, or databases, you need to carefully examine what might happen if multiple calls are made to service()at the same time. For example, suppose you had defined a counter in your servlet class that keeps track of how many service()

method invocations are currently running:

private int counter = 0;

Next, suppose that your service()method contained the following

code:

```
int myNumber = counter + 1; // line 1

counter = myNumber; // line 2

// rest of the code in the service() method

counter = counter - 1;
```

What would happen if two service()methods were running at the same time, and both executed line 1 before either executed line 2? Both would have the same value for myNumber, and the counter would not be properly updated. For this situation, the answer might be to synchronize the access to

the counter variable:

```
synchronized(this) {

myNumber = counter + 1;

counter = myNumber;

}

// rest of code in the service() method

synchronized(this) {

counter = counter - 1 ;

}
```

This ensures that the counter access code is executed only one thread at a time.

There are several issues that can arise with multi-threaded execution, such as deadlocks and coordinated interactions. There are several good sources of information on threads, including Doug Lea's book Concurrent Programming in Java.

JSDK 1.0 and JSDK 2.0

The Java Servlet Development Kit (JSDK) provides servlet support for JDK 1.1 and Java 2 platform developers. JSDK 1.0 was the initial release of the development kit. Everything worked fine, but there were some minor areas that needed

improvement. The JSDK 2.0 release incorporates these mprovements. The changes between JSDK 1.0 and JSDK 2.0 are primarily the addition of new classes. In addition, there is also one deprecated methods. Because some web servers still provide servlet support that complies with the JSDK 1.0 API definitions, you need to be careful about upgrading to the new JSDK.

New Servlet Features in JSDK 2.0

JSDK 2.0 adds the following servlet support:

The interface SingleThreadModelindicates to the server that only one thread can call the service()method at a time. -Readerand Writeraccess from ServletRequestand ServletResponse Several HTTP session classes that can be used to provide state information that persists over multiple connections and requests between an HTTP client and an HTTP server. :Cookie support is now part of the standard servlet extension. Several new HTTP response constants have been added to

HttpServletResponse

Delegation of DELETE, OPTIONS, PUT, and TRACEto appropriate methods in HttpServlet

JSDK 2.0 deprecated one method:

getServlets()--you should use getServletNames()instead