HTML, XML and JavaScript

by
Willi-Hans Steeb
International School for Scientific Computing

email addresses of the author:

steeb_wh@yahoo.com

Willi-Hans.Steeb@fhso.ch

whs@na.rau.ac.za

Contents

Preface

Chapter 1 covers the basics of HyperText Markup Language (HTML). HTML tells a Web browser how to lay out a Web page. JavaScript is an object-oriented scripting language created by Netscape Communications Corporation for developing Internet applications. XML stands for Extensible Markup Language. XML allows the author to define his own tags and his own document structure. An introduction to XML is given in chapter 2. Chapter 3 gives an introduction to JavaScript. Finally chapter 4 lists important Web sites for HTML, XML and JavaScript.

The level of presentation is such that one can study the subject early on in ones education in programming. There is a balance between practical programming and the underlying language. The book is ideally suited for use in lectures on Java and object-oriented programming. The beginner will also benefit from the book. The reference list gives a collection of textbooks useful in the study of the computer language Java. There are a number of good textbooks for Java available [1], [2]. For applications of Java in science we refer to Tan Kiat Shi, W.-H. Steeb and Yorick Hardy [6] and Steeb [5]. Comprehensive introductions into JavaScript are given by [3] and [4].

Without doubt, this book can be extended. If you have comments or suggestions, we would be pleased to have them. The email addresses of the author are:

whs@na.rau.ac.za
steeb_wh@yahoo.com

The web sites of the author are:

http://zeus.rau.ac.za
http://issc.rau.ac.za

Chapter 1

HTML

1.1 Introduction

Hypertext Markup Language (HTML) is a system for marking up documents with tags that indicate how text in the documents should be presented and how the documents are linked together. Hypertext links are quite powerful. Within the HTML markup scheme lies the power to create interactive, cross-platform, multimedia, client-server applications. Such a system is the World Wide Web (also known as WWW or just simply, the Web).

The Web is an interlinked collection of living documents containing formatted text, images, and sound. These documents are organized into webspaces. A webspace is typically structured around a home page with links to other pages or documents both in and outside of the webspace. A home page functions as a virtual meeting place in cyberspace for the exchange of information. We write a home page in HTML.

There are many home pages with information about HTML and the World Wide Web, HTML is a language under construction. The continuing development of HTML is conducted on the Web in an open process. New tools and techniques appear frequently and are quickly spread throughout the community of Web authors.

HTML is not a programming language and an HTML document is not a computer program. A computer program is a series of procedures and instructions applied, typically, to external data. An HTML document, however, is the data. The definition of HTML specifies the grammar and syntax of markup tags that, when inserted into the data, instruct browsers - computer programs that read HTML documents - how to present the document.

Technically, HTML is defined as a Standard Generalized Markup Language (SGML) Document Type Definition (DTS). An HTML document is said to be an *instance* of a SGML document. SGML orginated as GML (General Markup Language) at IBM in the late 1960s as an attempt to solve some of the problems of transporting documents across different computer systems. The term markup comes from the publishing industry. SGML is generalized, meaning that instead of specifying exactly how to present a document, it describes document types, along with markup languages to format and present instances of each type. GML become SGML when it was accepted as a standard by the International Standards Organization (ISO) in Geneva, Switzerland (reference number ISO 8879:1986).

An SGML document has three parts. The first describes the character set and, most importantly, which characters are used to differentiate the text from the markup tags. The second part declares the document type and which markup tags are accepted as legal. The third part is called the document instance and contains the actual text and the markup tags. The three parts need not be in the same physical file. All HTML browsers assume the same information for the SGML character-set and document-type declarations, so we only have to work with HTML document instances - simple text files.

The base character set of an HTML document is Latin-1 (ISO 8859/1). It is an 8-bit alphabet with characters for most American and European languages. Plain old ASCII (ISO 646) is a 7-bit subset of Latin-1. There is no obligation to use anything but the 128 standard ASCII characters in an HTML document. In fact, sticking to straight ASCII is encouraged as it allows an HTML document to be edited by any text editor on any computer system and be transported over any network by even the most rudimentary of e-mal and data transport systems. To make this possible, HTML includes character entities for most of the commonly used non-ASCII Latin-1 characters. These character entities begin with the ampersand character (`&`), followed by the name or number of the character, followed by a semicolon. In the next section we describe the HTML language. Almost all of the development work on HTML is done on the Internet in form of discussion groups, which post proposed changes and issue requests for comments. The complete specifications of HTML (the SGML DTS) can always be found on the Web. The Web is also the place to look for the most up-to-date HTML and SGML documentation, most of it in hypertext.

Extensible Markup Language, or XML for short, is a new technolgy for web applications. XML is a World Wide Web Consortium standard that let us create our own tags. HTML tags decribe how something should render. They do not contain any information about what the data is, they only describe how it should look. An XML element is made up of a start tag, an end tag and data between. The start and end tags describe the data within tags, which is considered the value of the element. An element can optionally contain one or more attributes. An attribute is a name-value pair separated by an equal sign (`=`). XML is case-sensitive.

1.2 HTML Tags

HTML works as a system of *tags*, one word or coded commands surronded by right-angle parenthese (<>). Most tags has a front and a back form which encases the text and instructs the browser software or computer server on how that text should appear and what functions it might activate. All web pages begin and end with the <HTML> tags. After the initial <HTML> tag, the <HEAD> </HEAD> tags reside and contain the <TITLE> </TITLE> tags between which resides the title of the page that is displayed on the top bar of the browser screen. Next the <BODY> </BODY> tags are placed before and after all the text included in the page, and within body tag are specifications for text and background colors. The ordinary text is diplayed as text. The tags (also called elements) are special instructions. Tags are identified by their names enclosed in angle brackets. The special symbols (also called entities) are code punctuation marks such as the ampersand & and the quotation marks ".

Thus HTML markup tags are delimited by the angle brackets,

```
< ... >
```

They appear either singularly, like the tag <P> to indicate a paragraph break in the text, or as a pair of starting and ending tags that modify the content contained. For example

```
<B> Attention! </B>
```

is an instruction to present the text string Attention! in a bold typeface. Other examples are

```
<COMMENT> This is a comment </COMMENT>
```

which is a comment on an HTML file and

```
<CENTER> ... </CENTER>
```

will centre the text or image. There are

tags for formatting text,

tags for specifying hypertext links,

tags for including sound and picture elements,

tags for defining input fields for interactive pages.

That's all there is to Hypertext Markup Language – character entities and markup tags. However, this system of entities and tags is evolving. HTML is not case-sensitive.

1.3 Main Commands

As described above an HTML program (also called an HTML script) is a sequence of three kinds of tokens

ordinary text characters, tags, and special symbols.

The main commands in HTML are

```
Structure Tags
<HTML> ... </HTML> Encloses the entire HTML document
<HEAD> ... </HEAD> Encloses the head of the HTML document
<BODY> ... </BODY> Encloses the body (text and tags)
                   of the HTML document

An attribute is
BGCOLOR="..."

Headings and Title
<H1> ... </H1> A first-level heading
<H2> ... </H2> A second-level heading
<H3> ... </H3> A third-level heading
<H4> ... </H4> A fourth-level heading
<H5> ... </H5> A fifth-level heading
<H6> ... </H6> A sixth-level heading
<TITLE> ... </TITLE> Indicates the title of the document.
                     Used with <HEAD>
All heading tags accept the attribute
ALIGN="..."  Possible values are CENTER, LEFT, RIGHT

Comments
<COMMENT>  ... </COMMENT>   Comments
<!- ... ->                  Comments

Paragraphs and Regions
<P> ... </P>  A plain paragraph.
              The closing tag (</P>) is optional
Attribute
ALIGN="..."  Align text to CENTER, LEFT, RIGHT

<DIV> ... </DIV>  A region of text to be formatted.

Attribute
ALIGN="..."  Align text to CENTER, LEFT, RIGHT
```

```
Lists
<OL> ... </OL> An ordered (numbered) list (items marked with <LI>)
<UL> ... </UL> An unordered (bulleted) list (items marked with <LI>)
<MENU> ... </MENU>  A menu list of items
<DIR> ... </DIR>  A directory listing
<LI> A list item of use with <OL>, <UL>, <MENU>, <DIR>
<DL> ... </DL>  A definition or glossary list

Blocks of Text
<BLOCKQUOTE> ... </BLOCKQUOTE> Extended quotation
<ADDRESS> ... </ADDRESS> Address often used for document
                         author identification

Character Formatting
<CODE> ... </CODE> Code sample (usually Courier)
<B> ... </B> Boldface text
<I> ... </I> Italic text
<TT> ... </TT> Typewriter text
<EM> ... </EM> Emphasis (usually italic)
<CITE> ... </CITE> A citation
<PRE> ... </PRE> Preformatted text (preserves linebreak)
<BIG> ... </BIG> Test is slighly smaller than normal
<SMALL> ... </SMALL> Test is slighly smaller than normal
<SUB> ... </SUB> Subscript
<SUP> ... </SUP> Superscript
<KBD> ... </KBD> Keyboard entry

Other Elements
<HR>  A horizontal rule line

Attributes
SIZE="..."  The thickness of the rule, in pixels
WIDTH="..." The width of the rule, in pixels
ALIGN="..." How the rule line will be aligned on the page
NOSHADE="..."  Causes the rule line to be drawn as a solid line

<BR> A line break
<CENTER> ... </CENTER>  Centers text or images
<BLINK> ... </BLINK>  Causes the enclosed text to blink irritatingly
<FONT> ... </FONT>  Changes the size of the font for the enclosed text

Attributes
SIZE="..."  The size of the font
COLOR="..." Changes the color of the text
FACE="..."  Name of font to use
```

```
Tables
<TABLE> ... </TABLE> Creates a table that can contain a caption
                     (<CAPTION>) and any number of rows (<TR>)
<CAPTION> ... </CAPTION> The caption of the table
<TR> ... </TR> Defines a table row, containing headings and data
<TH> ... </TH> Defines a table heading cell
<TD> ... </TD> Defines a table data cell

Images, Sounds, and Embedded Media
<IMG>  Inserts an inline image into the document

Attributes
SRC="..." The URL of the image
ALIGN="..."  Determines the alignment of the given image
VSPACE="..." The space between the image and the text above or
             below it
HSPACE="..." The space between the image and the text to its
             left or right
WIDTH="..."  The width in pixels of the image
HEIGHT="..." The height in pixels of the image

Links
<A> ... </A>  With the HREF attribute, creates a link to another
              document or anchor; with the NAME attribute, creates
              an anchor that can be linked to.

Attributes
HREF="..."  The URL of the document to be called when the link is
            activated
NAME="..."  The name of the anchor

Forms
<FORM> ... </FORM>  Indicates an input form

Attributes
ACTION="..."  The URL of the script to process this form input
METHOD="..."  How the form input will be sent to the gateway
              on the server side. Possible values are GET and POST

<INPUT>     Input element with a specific type and symbolic name

Attributes
TYPE="..."  The type for this input widget. Possible values are
            CHECKBOX, HIDDEN, RADIO, RESET, SUMBIT, TEXT, SEND,
```

```
            FILE, IMAGE
NAME="..."  The name of this item, as passed to the gateway script
            as part of a name/value pair
VALUE="..." For a text or hidden widget, the default value;
            for a check box or radio button,
            the value to be submitted with the form;
            for Reset or Submit buttons,
            the label for the button itself
SRC="..."   The source file for an image
SIZE="..."  The size, in characters, of a text widget

<TEXTAREA>  Text area (lines of editable text)
<SELECT>    Select element with symbolic name
<OPTION>    Option element used with SELECT

Applets and Scripting
<APPLET> ... </APPLET>

Attributes
CLASS="..."  The name of the applet
SRC="..."    The URL of the directory where the compiled applet
             can be found
ALIGN="..."  Indicates how the applet should be aligned with
             any text that follows
WIDTH="..."  The width of the applet output area in pixels
HEIGHT="..." The height of the applet output area in pixels

<SCRIPT> ... </SCRIPT>  An interpreted script program

Attributes
LANGUAGE="..." For example JavaScript
SRC="..."      Specifies the URL of a file that includes the script
               program
```

The escape sequences

```
&LT   &GT   &AMP   &QUOT
```

are used to enter characters such as <, >, and & and " into HTML documents. The mnemonics of the original four sequences from HTML 2.0 stand for Less Than, Greater Than, AMPersand, and double QUOTe. HTML 3.2 adds NonBreaking SPace, REGistered trademark and COPYright

```
    &REG   &COPY
```

For example, `&LT;BR>` will display as <BR>.

1.4 Linking to other Documents

HTML defines hypertext links which enables us to link regions of text, files or graphics objects to our document. We do this via the HTML anchor tab. For example, the following snippet from an HTML document

```
<A HREF="ScSchool.html">
The International School for Scientific Computing</A>
at the Rand Afrikaans University offers a
<A HREF="..\diploma\diploma.html"> Diploma in    </A>
```

has two links to other HTML documents. The first is to `SciSchool.html` which must reside in the same directory as the current HTML document. The first anchor tag contains the information for the document and the text between this tag and the `</A>` is displayed by the Web browser in a different colour (normally blue). If the user clicks with the mouse onto this highlighted text, the browser will fetch the corresponding document. The second anchor tag is a file `diploma.html` which resides in a directory `diploma` situated at one branch lower in the directory hierarchy. Thus if the current document is in the directory `c:\webdocs\scschool` then the Web browser will search for `c:\webdocs\diploma\diploma.html`. It is a good idea to make all links relative, so that if we move the web site to a different location, all links will remain intact. We can also establish links to documents on other web sites. For example

```
<A HREF="http://issc.rau.ac.za"> ISSC </A>
```

Of course we can also establish a link to a specific page other than the home page of another web site. For huge HTML documents it can be very useful to be able to jump to a specific section within that document. This is achieved by supplying a named anchor within that document and using the hash character `#` to specify a jump to a named anchor.

```
...
<LI> <A HREF="#Introduction"> Introduction </A>
<LI> <A HREF="#FirstSection"> First Section </A>
...
```

If the user clicks on the link the web browser shows the corresponding section of the document.

```
<H3> <A NAME = "INTRODUCTION"> Introduction </A> </H3>
This is the body of the introduction ...

<P>

<H3> <A NAME = "FirstSection"> First Section </A> </H3>
This is the body of the first section ...
```

1.5 Forms and CGI Programming

HTTP is the protocol of the Web, by which Servers and Clients (typically browsers) communicate. An HTTP transaction comprises a Request sent by the Client to the Server, and a Response returned from the Server to the Client.

The HTML `FORM` is the most-frequently used format in a Web page for collecting data and interacting with the user. The `<FORM>` tag is used to start an HTML form and to specify the `URL` location and the CGI (Common Gateway Interface) script that we want to run. For the task, we assign the `URL` to the `ACTION` attribute within the `<FORM>` tag. To end a form, we put the ending tag `</FORM>` in the HTML code. Here, the `ACTION` attribute is given the `URL` of the CGI script called for example `chtml.exe`. CGI itself is not a language. It is just an interface between a client and a server across the Internet. The CGI script refers to a program that supports CGI and is written in a computing language such as C, C++ or Perl.

`POST` and `GET` are two methods used to send data collected from an HTML form to the CGI scripts running on the Web server.

`GET` method: a form method that sends the user's request into the program as command line arguments (equivalent to `argc` and `argv[]` in C++ and C).

`POST` method: a form method that sends the user's request to a cgi-bin program as `stdin` (standard input). The user's request can be retrieved by the cgi-bin program using any `stdin` routine, for example in C the function `gets()` and in C++ the function `cin.getline()`. The function `char* gets(char* s)` gets a string from `stdin`. The function `istream& getline(char*,int,char)` extracts characters into the given streambuffer until the delimiter is encountered. The delimiter is not copied to buffer.

For example the HTML file on the client site could be

```
<HTML>
<COMMENT> File name: c3test.html </COMMENT>
<HEAD><TITLE> Password for access </TITLE></HEAD>
<BODY BGCOLOR='pink'>
Enter password for access
<FORM METHOD="POST"
ACTION="http://zeus.rau.ac.za/scripts/chtml.exe">
<P> <INPUT TYPE = "text" NAME = "pass" > </P>
<P> <INPUT TYPE = "submit" NAME = "submit" > </P>
</FORM>
</BODY>
</HTML>
```

The exe-file `chtml.exe` is on the server site with the `URL` given by

`http://zeus.rau.ac.za`

`scripts` is a pointer to the sub-directory where `chtml.exe` is located.

Non-whitespace characters or non-alphanumeric characters that need to be sent over HTTP to and from our script need to be encoded to allow their transmission The *alphanumeric characters* are given by

`AB...YZab...yz01...9`

Anything except alphanumeric characters gets encoded. Spaces are encoded by a `+`. The others are coded as hex values for their ASCII value, led by `%`.

The HTML server will send the data in the following format:

`mailtoname=MAIL_STRING&emailad=EMAIL_STRING&message=MESSAGE_STRING`

The parts in capitals are the variable information coming from the user (client). The lowercase are the names of the variables followed by the equal sign `=` and separated by the ampersand `&`. Each variable except the last one will end with an ampersand `&` sign to identify it. The final variable ends with a carriage return `\n`. There will be an equal sign before the actual user value. All variables are send as a single string. Spaces are replaced by `+` signs. For example,

`Willi & Hans Steeb`

is encoded as

`Willi+%26+Hans+Steeb`

where `26` (hex) is the ASCII value for the ampersand `&` and `%26` is the HTML form of `&`.

The following Perl program does the URL encoding and decoding.

```
# url.pl

sub url_encode
{
   my $text = shift;
   $text =~ s/([^a-z0-9_.!~*'() -])/sprintf "%%%02X",ord($1)/egi;
   $text =~ tr/ /+/;
   return $text;
}

sub url_decode
```

```
{
   my $text = shift;
   $text =~ tr/\+/ /;
   $text =~ s/%([a-f0-9][a-f0-9])/chr(hex($1))/egi;
   return $text;
}

my $url = "willi-hans & steeb @ ISSC";
my $enc = url_encode($url);
my $dec = url_decode($enc);

print "original = $url\n";
print "encoded = $enc\n";
print "decoded = $dec\n";
```

The underlying C++ program for the exe-file `chtml.exe` on the server site could be as follows

```
// chtml.cpp

#include <iostream.h>
#include <string.h>

const int numpass = 3;
const char* passlist[numpass] = { "123", "456" , "789" };

void main(void)
{
 int accepted = 0;
 int i;

 // assume password no more than 1023 characters
 char password[1024];

 // expect "pass=_password_" from stdin
 cin.getline(password,1024,'=');

 // get entered password
 cin.getline(password,1024,'&');

 for(i=0;i<numpass;i++)
  accepted = (strcmp(password,passlist[i])==0)?1:accepted;

 // generate HTML output
 cout << "Contents-type: text/html\n\n";
```

```
 cout << "<HTML>\n";
 cout << "<HEAD><TITLE>\n";

 if(accepted) cout << "Password accepted\n";
 else cout << "Password incorrect\n";

 cout << "</TITLE></HEAD>\n";
 cout << "<BODY>\n";

 // if accepted add line to automatically send to next webpage
 if(accepted)
 {
 cout << "Password accepted\n";
 cout << "<A HREF=" << "\"http://zeus.rau.ac.za/openme.html\">"
         << "openme" << "</A>\n";
 }
 else cout << "Password incorrect.\n";

 cout << "</BODY>\n";
 cout << "</HTML>\n";
}
```

The HTML commands are embedded as a string in `cout`. Obviously the HTML file `openme.html` is on the server site.

There are various advantages of CGI:

1) Simplicity. CGI provides a simple way of running programs on the server when a request is received and it is conceptually easy to understand the underlying process.

2) Process Isolation. Since CGI applications run in separate processes, buggy applications will not crash the Web server or access the server's private internal state.

3) Portability. CGI is an open standard. CGI is not tied to any particular (such as single- or multi-threaded) server architecture. CGIs are far more portable over other alternatives such as server extension APIs.

4) Language independency. CGI applications can be written in any language.

5) Support. CGI is a proven technology, and some form of CGI has been implemented on almost every Web server on a variety of platforms. There are many CGI scripts available for free for a variety of applications: as user-friendly front-ends to databases, search engines, scientific analysis tools, traditional inventory systems, gateways to network services such as `gopher` and `whois`.

The INPUT tag has the form

```
<INPUT NAME="name" TYPE="checkbox |hidden |image |password |radio
                                  reset |submit |text">
```

where "text" is the default. The pushbutton "submit" causes the current form to be packaged up into a query URL and sent to a remote server.

Inside <FORM> ... </FORM> any number of SELECT tags are allowed, freely intermixed with other HTML elements (including INPUT and TEXTAREA elements) and text but no additional forms. Unlike INPUT, SELECT has both opening and closing tags. Inside SELECT, only a sequence of OPTION tags each followed by an arbitrary amount of plain text. For example

```
<SELECT NAME = "books">
<OPTION> First option.
<OPTION> Second option.
</SELECT>
```

The attribute NAME is the symbolic name for the SELECT element. This must be present, as it is used when putting together the query string for the submitted form.

The TEXTAREA tag can be used to place a multiline text entry field with optional default contents in a fill-out form. The attribute NAME is the symbolic name of the text entry field. The attribute ROWS is the number of rows (vertical height in characters) of the text entry field. COLS is the number of columns (horizontal width of characters) of the text entry field. The TEXTAREA element requires both an opening and a closing tag. TEXTAREA fields automatically have scrollbars; any amount of text can be entered in them. A TEXTAREA with no default contents looks like this

```
<TEXTAREA NAME = "foo" ROWS=4 COLS=40></TEXTAREA>
```

A TEXTAREA with default contents looks like this

```
<TEXTAREA NAME = "foo" ROWS = 4 COLS = 40>
The default contents go here
```

The default contents must be straight ASCII text. Newlines are respected.

1.6 Images

Images can be created in two ways - with the <IMG> tag and with the Image() constructor. <IMG> are placed on the form and have a whole set of properties that can be used to format their position, size and appearance. The images are automatically formed into the array images[], and numbered in the order of their position in the document. Thus the first image can be referred to as

```
document.images[0]
```

Images must be in gif or jpg format. The command

```
<IMG> inserts an inline image into the document
attributes
SRC="..." the URL of the image
```

An example is given by

```
<IMG SRC=
"http://www.paris.org/Musees/Louvre/Treasures/gifs/Mona_Lisa.jpg">
```

We can also use the following options (attributes):

```
ALIGN = "left/center/right"
ALT = "description"
WIDTH = "how wide is the picture"
HEIGHT = "how tall is the picture"
BORDER = "border around the picture"
HSPACE = "horizontal distance between picture and text"
VSPACE = "vertical distance between the picture and the text"
```

The option ALIGN sets the position of the image across the page. The option ALT is the text to display if the image is not loaded into a browser.

We can tile a page with the

```
BACKGROUND = "filename"
```

option in the <BODY> tag. The image is repeated to fill the window. For example

```
<BODY BACKGROUND="mypicture.gif" ALIGN=right ALT="My picture to watch">
```

1.7 HTML Sound Tags

Not all browsers support every sound tag and not every computer has the proper install and configuration of a sound card, sound card drivers, and multimedia players such as Quick Time and Windows Media Player. If the volume setting for the tag is set low and the computer's volume setting is low, no sound may be heard. Finally, if our web page visitor does not have a sound capability, they may be get annoying error messages for any automatically started sound. Thus, it is still best to let the visitor to our page control the sound (no hidden controls or auto starts).

The background sound tag

```
<BGSOUND SRC="URL">

<BGSOUND SRC="URL" LOOP=n>
```

identifies a `.wav`, `.au`, `.mid` or `.aif` resource that will be played when the page is opened. The optional `LOOP` attribute will cause the resource to be played `n` times. The command `LOOP="INFINITE"` will cause the resource to be played continuously as long as the page is open.

The embed element

```
<EMBED SRC="URL">
```

is used to embed a plugin into a document. The `OBJECT` tag can also be used to embed objects.

The `META` tag can also be used to play sound.

```
<HTML>
<HEAD>
<TITLE> SOUND1 </TITLE>
</HEAD>

<BODY>
<H2 ALIGN="center">
wav file in HTML document
</H2>

<BGSOUND SRC="hello.wav" AUTOSTART="true" LOOP="true">

</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE> SOUND2 </TITLE>
</HEAD>

<BODY>
<H2 ALIGN="center">
au file in HTML document
</H2>

<META HTTP-EQUIV="Refresh" CONTENT="1; URL=hello.au">

</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE> SOUND3 </TITLE>
</HEAD>

<BODY>
<H2 ALIGN="center">
aiff file in HTML document
</H2>

<EMBED SRC="hello.aiff" AUTOSTART="true" LOOP="false"
HIDDEN="true" HEIGHT="0" WIDTH="0">

</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE> SOUND4 </TITLE>
</HEAD>

<BODY>
<H2 ALIGN="center">
au file in HTML document
</H2>

<OBJECT DATA="hello.au" TYPE="AUDIO/BASIC"></OBJECT>

</BODY>
</HTML>
```

1.8 Tables

Tables are not just for tables of data. They are also a convenient way to present sets of images, or to lay out images and related text.

A *table* begins and ends with the <TABLE> </TABLE> tags. Tables rows are defined by <TR> </TR>; header cells by <TH> </TH>; and data cells by <TD> </TD>. A table must have at least one row and one cell. An optional table sub-element, <CAPTION> </CAPTION> supplies a caption for the table. A simple example is given by

```
<TABLE BORDER>
<TD> Row One - Column One </TD>
</TABLE>
```

Typically, a table with a border is cosmetically more appealing.

A more advanced example is given in the following program

```
<HTML>

<COMMENT> Table.html </COMMENT>

<BODY>

<TABLE>
<CAPTION> Table Items </CAPTION>
<TR><TH></TH>
<TH> Lunch </TH> <TH> Dinner </TH>
<TR> <TH> Kitchen </TH> <TD> 23 <TD> 30 </TD> </TR>
<TR> <TH> Dining Room </TH> <TD> 31 </TD> <TD> 45 </TD> </TR>
</TABLE>

</BODY>

</HTM>
```

1.9 Examples

Let us assume the file name of this HTML program is `mona.html` and the file is on the C drive. To run it at your favorite browser Netscape's Navigator or Microsoft's Internet Explorer type at the address:

```
C:\mona.html
```

Then we press enter.

```
<HTML>
<HEAD>
<TITLE> An Example of an HTML File Structure </TITLE>
</HEAD>

<BODY BGCOLOR='pink'>
Good Morning Egoli !
<BR> <BR>
Let's do mathematics.
<BR> <BR>

<MATH>
&int;_a_^b^{f(x)<over>1+x</over>} dx
</MATH>

<COMMENT> verbatim </COMMENT>
<PRE>
#include < iostream.h >

int main()
{
   int a = 7;
   int b = 2*a;
   cout << "b = " << b;
   return 0;
}
</PRE>

<IMG SRC=
"http://www.paris.org/Musees/Louvre/Treasures/gifs/Mona_Lisa.jpg">

</BODY>
</HTML>
```

Next we give the HTML file of the web site of the International School for Scientific Computing. It shows how <IMG> and <A> ... </A> is applied.

```
<HTML>
<HEAD>
<TITLE> ISSC  Certificate Courses </TITLE>
</HEAD>

<BODY  BACKGROUND="../breadc2.gif"  >

 <CENTER>
  <H1>Certificate Courses in Scientific Computing
      and  Software Engineering</H1>
  <H4>presented by the</H4>
  <H1><EM>International School for Scientific Computing</EM></H1>
</CENTER>

<CENTER>
  <IMG SRC="../ISSCweb.gif" HSPACE=50 ALIGN=CENTER>
</CENTER>

<HR>
Currently the ISSC offers the following certificate courses:

<P>
 <UL>
   <LI> <B>Core Courses</B>
    <OL>
    <LI> <A HREF="oopcpp/c_cpp.html">
         Object-Oriented Programming in C++</A>
    <LI> <A HREF="ooadp/c_ooadp.html">
         Object-Oriented Analysis and Design using UML</A>
    <LI> <A HREF="java/c_java.html">
         Programming in Java, HTML and JavaScript</A>
    <LI> <A HREF="AdvJava/c_AdvJava.html">Advanced Java</A>
    <LI> <A HREF="c_assemb.html">Assembly Language and C++</A>
    </OL>

  <LI> <B>Scientific Courses</B>
    <OL>
    <LI> <A HREF="neural/c_neural.html">
         Neural Networks, Genetic Algorithms and Optimization</A>
    <LI> <A HREF="fuzzy/c_fuzzy.html">
         Fuzzy Logic with C++ and Hardware Implementations</A>
```

```
    <LI> Numerical Methods for Science and Engineering
    <LI> Computational Graph & Transport Theory
         with Applications and C++ Implementations
    <LI> Algebraic Computing using
         SymbolicC++, Mathematica, Maple and Reduce
    <LI> <A HREF="microproc/c_micropr.html">
         Programming Microprocessors & Embedded Systems</A>
    <LI> <A HREF="c_infoth.html">
         Information Theory and Maximum Entropy Inference</A>
    <LI> Fibre Optics with Applications in Communications
    <LI> <A HREF="digital/c_digital.html">
         Digital Electronics, Programmable Logic Devices and VHDL</A>
    <LI> Microprocessor Based Mobile Robots
    <LI> <A HREF="unix/c_unix.html">
         Unix, Shell Programming and Perl</A>
    <LI> <A HREF="c_cqcomp.html">
         Classical and Quantum Computing</A>
    </OL>

  <LI> <B>Specialized Courses</B>
    <OL>
    <LI> <A HREF="CORBA/c_CORBA.html">
         Distributed Objects with CORBA and Java</A>
    <LI> <A HREF="c_lisp.html">
         Artificial Intelligence using Lisp and C++</A>
    <LI> <A HREF="VisualCPP/c_VisCPP.html">
         Programming for Windows using Visual C++.</A>
    <LI> <A HREF="c_abap.html">SQL, Oracle and ABAP/4</A>
    </OL>

  <LI> <B>Commerce, Economics and Administration</B>
    <OL>
    <LI> <A HREF="Finance/c_Finance.html">
         Computational Methods for Finance and Economics</A>
   </OL>
</UL>

<HR>
All lectures are presented in English.
Each certificate course includes about 15 hours of formal lectures
and up to 15 hours of practicals after which you will write an exam.
Should you not pass the exam the first time, you will be offered an
opportunity to rewrite the exam (free of charge). After successful
completion you will receive a certificate from the International
School for Scientific Computing certifying that you completed the
```

```
particular course successfully.
At any stage you may decide to enroll for the <A HREF="diploma.html">
Diploma in Scientific Computing and Software Engineering</A>.
You will be given credits for certificate courses that you have
already completed.

<P> <CENTER>
<A HREF="enrol00.html"><H2>Timetable and Enrollment Form 2000</H2></A>
</CENTER>

<HR>
<P> For further information contact
<OL>
 <LI> <A HREF="../steeb/steeb.html">Prof. Willi-Hans Steeb</A>
      <ADDRESS>
      <UL>
       <LI> Telephone: (+27) (011) 489-2331.
       <LI> Fax: (+27) (011) 489-2616.
       <LI> <A HREF="mailto:whs@na.rau.ac.za"> E-Mail:
             whs@na.rau.ac.za</A>
        </ADDRESS>
      </UL>
 <LI> Prof. Charles Villet
      <ADDRESS>
      <UL>
        <LI> Telephone: (+27) (011) 489-2316.
        <LI> Fax: (+27) (011) 489-2616.
        <LI> <A HREF="mailto:cmv@na.rau.ac.za"> E-Mail:
               cmv@na.rau.ac.za</A>
        </ADDRESS>
      </UL>

<HR>
<MAP NAME="ToHomeMap">
  <AREA SHAPE=RECT COORDS="0,0,600,150"
        HREF="../scschool/scschool.html">
</MAP>

<CENTER>
  <IMG SRC="../ISSChome.gif" BORDER=0 USEMAP="#ToHomeMap">
</CENTER>
<HR>

</BODY>
</HTML>
```

1.10 Applets

The key methods of the `Applet` class are:

```
void init()
void start()
void stop()
void destroy()
```

The method `init()` is called whenever an applet is loaded by a browser. It is called before the `start()` method and is normally used to initialize variables. The method `start()` is called after the initialization of the applet. While the `init()` method is only called when the web browser is loaded, the `start()` method is called every time that the page is loaded and restarted. When we go to another page and then return, the `start()` method is called. The `stop()` method is called whenever the browser leaves a page and goes to another page. The `destroy()` method is called just before an applet finishes, if, for example the browser is shut down.

To run an applet from a web page, we first have to create an HTML program. Consider the applet (filename `Greetings.java`)

```
import java.applet.Applet;
import java.awt.Graphics;

public class Greetings extends Applet
{
   public void paint(Graphics g)
   {
   g.drawString("Good Morning Egoli",50,50);
   g.drawString("Good Night Egoli",100,100);
   }
}
```

Next we give a minimal HTML program from which we can run the applet.

```
<HTML>
<APPLET CODE="Greetings.class" width=300 height=150></APPLET>
</HTML>
```

Enter this code into a separate text file named `Greetings.html`. Then execute the following two JDK commands at the command prompt

```
javac Greetings.java
appletviewer Greetings.html
```

The `javac` command compiles the applet producing the class file `Greetings.class`. The `appletviewer` command launches a special program provided in the JDK that runs HTML programs just like a Web browser. In this case, the `appletviewer` runs the two-line HTML program that we entered with `Greetings.html` The HTML program tells the `appletviewer` to run the Java code contained in the `Greetings.class` file, using a panel of 300 pixels wide and 150 pixels high.

The HTML program consists of a single applet tag. This tag has three attributes: `CODE="Greetings.class"` `width=300` `height=150`. These are like arguments passed to a method of function, providing information about what is to be done. The action to be performed is the execution of a Java applet. The appletviewer needs to know the name of the applet `CODE="Greetings.class"` and the size of the panel (`width=300 height=150`) to use. The code `</APPLET>` signals the end of the tag.

```
<HTML>

<COMMENT> File name: mona1.html </COMMENT>

<HEAD>
<TITLE> An Example of an HTML File Structure </TITLE>
</HEAD>

<BODY BGCOLOR='pink'>
Good Morning Egoli !<br>

<BR>

<APPLET CODE="Greetings.class" width=300 height=150>
</APPLET>

<IMG SRC=
"http://www.paris.org/Musees/Louvre/Treasures/gifs/Mona_Lisa.jpg">

</BODY>
</HTML>
```

We saw in chapter 2 that we can pass arguments to a Java application when it starts running. We can do a similar thing with applets, but it requires co-operation between a Java applet and the HTML file. Within the `<APPLET>` `</APPLET>` pair, the name of the Java class file and the `width` and `height` of the applet drawing area are specified. To pass information to the applet, the names and values of the parameters are passed using `PARAM NAME` and `VALUE`.

For example

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<APPLET CODE="PassArg.class" width = 350 height = 150>
<PARAM NAME = first VALUE = "Willi">
<PARAM NAME = second VALUE = "Hans">
</APPLET>
</BODY>
</HTML>
```

Next we have to write some Java statements in our applet to receive these arguments. We put them inside the `init()` method. For example

```
// PassArg.java

import java.applet.Applet;
import java.awt.Graphics;

public class PassArg extends Applet
{
   public String firstPar, secondPar;

   public void init()
   {
   firstPar = getParameter("first");
   secondPar = getParameter("second");
   }

   public void paint(Graphics g)
   {
   g.drawString(firstPar,50,50);
   g.drawString(secondPar,100,100);
   g.drawString(firstPar.concat(secondPar),150,150);
   }
}
```

Chapter 2

XML

2.1 Introduction

XML stands for Extensible Markup Language. XML is a public standard developed and maintained by the World Wide Web Consortium (W3C). The W3C develops inter operable technologies (specifications, guidelines, software and tools) to increase the potential of the Web as a forum for information, commerce, communication and collective understanding. XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. It was designed to describe data. XML is a markup language much like HTML. As with HTML, we identify data using tags (identifiers enclosed in angle brackets). XML tags are not predefined in XML. We must define our own tags. XML is self describing. XML uses a DTD (Document Type Definition) to formally describe the data. XML was designed to describe data and to focus on what data is. HTML was designed to display data and to focus how data looks. HTML is about displaying information, XML is about describing information.

The tags used to markup HTML documents and the structure of HTML documents are predefined. For example the HTML tag `<B> ... </B>` renders the text in bold font style. The author of HTML documents can only use tags that are defined in the HTML standard. XML allows the author to define his own tags and his own document structure.

XML can keep data separated from our HTML. XML can be used to store data inside HTML documents. XML can be used as a format to exchange information. XML can be used to store date in files or in databases. Thus we can use XML to create an unlimited number of our own custom tags and attributes, assign a data type to each tag and attribute, access values associated with the tags, and accomplish all of this in a custom structured format that we have also created.

As described above, HTML pages are used to display data. Data are often stored inside HTML pages. With XML this data can now be stored in a separate XML file. This way we can concentrate on using HTML for formatting and display, and be sure that changes in the underlying data will not force changes to any of our HTML code.

However, XML can store data inside HTML documents as so-called *data islands*.

Computer systems and databases contain data in incompatible formats. One of the big problems is to exchange data between such systems over the Internet. Converting data to XML can greatly reduce this complexity and create data that can be read by different types of applications.

XML can also be used to store data in files or in databases. Applications can be written to store and retrieve information from the store, and generic applications can be used to display the data.

An XML document is an ordered, labeled tree: character data leaf nodes contain the actual data (text strings) where usually, character data nodes must be non-empty and non-adjacent to other character data nodes

elements nodes, are each labeled with a name (often called the element type), and a set of attributes, each consisting of a name and a value. These nodes can have child nodes.

In addition, XML trees may contain other kinds of leaf nodes: processing instructions - annotations for various processors

comments - as in programming languages

document type declaration.

In general, we should avoid attributes. Attributes cannot contain multiple values (elements can). Attributes are not expandable (for future changes). Attributes cannot describe structures (like child elements can). Attributes are more difficult to manipulate by program code. Attribute values are not easy to test against DTD (Document Type Definition).

XML documents may be encoded in a range of character sets. For multi-language support XML supports 16-bit Unicode (ISO/IEC 10646). Unicode supports all spoken languages including the European, Semitic, Arabian, Indian, Japenese and Chinese language. Of the roughly 40 000 characters defined by Unicode about 20 000 characters are used for Mandarin and other Chinese languages, 11 000 for Hangul (Korean). The remaining 9 000 characters suffice for most characters of the remaining spoken languages of the world.

Any combination or repeated occurances of space characters, horizontal tabs, line feeds and/or carriage returns is treated in XML as a single white space. All white space characters within the content of elements is preserved by parsers while multiple white space characters within element tags and attribute values may be removed and replaced with a single white space character.

There are three commonly used end-of-line delimiters, namely carriage-return (CR), line-feed (LF) or carriage-return + line-feed (CR+LF). To simplify processing, XML parsers were required to replace all end-of-line delimiters with a single line feed.

The following conditions for an XML document must be satisfied.

1) All XML elements must have a closing tag.

2) XML tags are case sensitive. Opening and closing tags must therefore be written with the same case. For example, an XML parser interprets `<BOOK>` and `<book>` as two different tags. Recall that HTML tags are not case sensitive.

3) All XML elements must be properly nested.

4) All XML documents must have a root tag. Recall that an XML document is an ordered, labeled tree.

5) Attributes must always be quoted. XML elements can have attributes in name/value pair just like in HTML. In XML the attribute value must always be quoted. XML attributes are normally used to describe XML elememts, or to provide additional information about elements.

XML names may only start with a letter, an underscore character or a colon. The subsequent characters may include the latter plus numerals, dashes and full-stops. Colons should, however, be used for namespace delimiters. Furthermore, XML names should not start with the three letters `XML` in any case (`xml`, `Xml`, `xmL`, ...). Names starting with these characters are reserved for W3C use only. Note that XML names are case-sensitive.

Since spaces are not allowed in elements names, the underscore (_) is a common symbol used to add space between words. Capitalizing the first letter of each word in the element is another way to delineate separate words. Our tags can be composed of almost any string of characters, with certain limitations: The first character of an XML tag must be an upper or lower case letter, the underscore, or a colon. The remaining characters can be composed of any combination of upper or lower case letters, colons, hyphens, numbers, periods, or underscores.

The very first line of every XML document must be the XML declaration. For example

```
<?xml version="1.0" standalone="yes" encoding="ISO-8859-1"?>
```

where <? begins a processing instruction. xml declares this to be an XML instruction. version identifies the version of XML specification in use. Version 1.0 is the only current version so the value must be 1.0. The markup standalone states if this file is completely by itself. A no value signals the parser that it should import other mark-up files, such as an external DTD (Document Type Definition). Since well-formed XML docs are complete by itself, this attribute value would be yes. The command encoding indicates which international character set is used. ISO-8859-1 is the default and is essentially the English character set. The markup ?> terminates the processing instruction.

A comment is given by

```
<!-- This is a comment -->
```

XML documents require a root element. The root element appears only once and encapsulates the entire XML elements and data in the document, in the same way <HTML> ... </HTML> contains the contents of an HTML document.

Since XML uses certain characters for its own syntax, these characters must be supplied in another way. For this reason XML introduces five standard entity references

```
&lt;   &gt;
```

These entity references must be used for literal left and right angle brackets which are used in XML as element tag delimiters.

```
&apos;   &quot;
```

These entity references must be used for single and double quotes which are used in XML as element tag delimiters.

```
 &amp;
```

& must be used for literal ampersands. Ampersands are used in XML for entity references.

In the following we give an example. The first line is the XML declaration. It defines the XML version of the document. Furthermore `library` is the root element appearing only once and containing all the other elements. We save the XML with any filename and an `.xml` extension. Then we load the file onto an XML-parser such as IE 6, NetScape 6, Opera 4+, Ice Browser, or Mozilla.

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="library.xsl"?>

<!-- File name: library.xml -->
<!-- address: file://c:\books\java\library.xml -->

<library>
  <book>
    <title> SymbolicC++ </title>
    <subtitle> An Introduction to Computer Algebra </subtitle>
    <author> Tan Kiat Shi, Willi Hans Steeb, Yorick Hardy </author>
    <ISBN> 1852332603 </ISBN>
    <publisher> Springer </publisher>
    <city> London </city>
    <year> 2000 </year>
  </book>

  <book>
    <title> Classical and Quantum Computing </title>
    <subtitle> with C++ and Java Simulations </subtitle>
    <author> Yorick Hardy and Willi Hans Steeb </author>
    <ISBN> 3764366109 </ISBN>
    <publisher> Birkhauser Publishing </publisher>
    <city> Basel </city>
    <year> 20001 </year>
  </book>

</library>
```

If we consider the XML file from an SQL table point of view then each

```
<book> ... </book>
```

block would be one row in the table `library`.

The following example shows how to use an attribute. The attribute is `sex`.

```
<?xml version="1.0" standalone="yes"?>

<people>
  <person sex="male">
    <lastname> Steeb </lastname>
    <firstname> Willi Hans </firstname>
    <profession> Professor </profession>
  </person>

  <person sex="female">
    <lastname> de Sousa </lastname>
    <firstname> Nela </firstname>
    <profession> Lecturer </profession>
  </person>

</people>
```

We can rewrite it without using an attribute.

```
<?xml version="1.0" standalone="yes"?>

<people>
  <person>
    <sex> male </sex>
    <lastname> Steeb </lastname>
    <firstname> Willi Hans </firstname>
    <profession> Professor </profession>
  </person>

  <person>
    <sex> female </sex>
    <lastname> de Sousa </lastname>
    <firstname> Nela </firstname>
    <profession> Lecturer </profession>
  </person>

</people>
```

The polynomial

$$p(x, y) = 2x^3y^2 - 4y^5$$

could be encoded in XML as

```
<?xml version="1.0"?>

<polynomial>
     <term>
       <factor> 2 </factor>
       <variable>
       <name> x </name>
       <exponent> 3 </exponent>
       </variable>
       <variable>
       <name> y </name>
       <exponent> 2 </exponent>
       </variable>
     </term>

     <term>
       <factor> -4 </factor>
       <variable>
       <name> x </name>
       <exponent> 0 </exponent>
       </variable>
       <variable>
       <name> y </name>
       <exponent> 5 </exponent>
       </variable>
     </term>

</polynomial>
```

2.2 Schema

A *schema* is a formal way of defining and validating the content of an XML document. A well-formed XML document that conforms to its schema is said to be valid. The schema is how we assign the data types to each tag and any attributes that are contained in the XML document. A schema is a structured document which must obey XML syntax rules. It is composed of a series of predefined tags and attributes that are part of the XML language and are used to set the data types for the values associated with our custom tags. Not only do we get to create custom XML tags, but we can also denote that an XML data value is, for example, an integer data type. Thus we can assign specific data types to specific XML data values. A schema can be part of the XML document or a separate file.

The XML tags and attributes to create a schema are:

The `Schema` tag serves as a container element that delimits the beginning and end of the schema. This tag must be closed.

The `xmlns` attribute is used to declare the data types of the schema XML namespace. The value is a URL or URN address that the browser will access to get information on the data types to allow code validation.

The `xmlns:dt` attribute is used to declare the data types of the schema XML namespace. The value is a URL (Uniform Resource Locator) or URN (Uniform Resource Name) address that the browser will access to get information on the data types to allow code validation. If we are using IE 5 to view our XML document, then we must include the `xmlns` and the `xmlns:dt` attributes exactly as displayed below

```
<Schema xmlns="urn:schema-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
...
</Schema>
```

The `AttributeType` tag is used to declare the data type for an attribute of an XML tag. This tag must be closed.

The `name` attribute provides the name of the attribute.

The `dt:type` attribute dictates the data type of the attribute.

The permitted values are:

```
binary  boolean  byte  date
decimal  double  float  int
integer  long  negativeInteger
nonNegativeInteger  nonPositiveInteger
positiveInteger  recurringInstant
short  string  time  timeDuration
timeInstant
unsignedByte  unsignedShort  unsignedInt  unsignedLong
```

The `attribute` tag is used to associate a previously data typed attribute to a tag. This tag must be closed.

The `type` attribute provides the data type of the custom attribute.

The `ElementType` tag is used to declare the data type for a custom XML tag. The tag must be closed.

The `content` attribute describes the intended content of the XML tag. There are four permitted values:

```
Type       Description
=======    ==============================
eltOnly    contains only elements
empty      contains no content
mixed      contains both elements and text
textOnly   contains only text
=========================================
```

The `name` attribute provides the name of the tag (element).

The `dt:type` attributes dictates the data type of the tag (element). There are 23 permitted values (see above).

The `element` tag is used to associate a previously data typed tag to an element. This tag must be closed.

The `type` attribute provides the type of the custom XML element.

2.3 Document Type Definition

The purpose of Document Type Definition (DTD) is to define the legal building blocks of an XML document. A DTD can be declared inline in our XML document or as an external reference. An example with an external reference is given below. XML provides an application independent way of sharing data. With a DTD, independent groups of developers can agree to use a common DTD for interchanging data. Our application can use a standard DTD to verify that data that we receive from the outside world is valid. We can also use a DTD to verify our own data.

XML documents (and HTML documents) are made up by the following building blocks:

`Elements, Tags, Attributes, Entities, PCDATA, CDATA`

Attributes provide extra information about elements. Attributes are placed inside the start tag of an element. Attributes come in name/value pairs. PCDATA means parsed character data. We think of character data as the text found between the start tag and the end tag of an XML. PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded. CDATA also means character data. CDATA is text that will not be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded. Entities are variables used to define common text. Entities references are references are references to entities. For example, the HTML entity reference ` ` is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser.

```
<?xml version="1.0"?>
<!-- familytree.dtd -->

<!-- PCDATA parseable character data -->
<!-- attributes  keyword ATTLIST -->
<!-- DTD describing an individual person -->

<!ENTITY % reference "person IDREF #REQUIRED">

<!ELEMENT person (name | father)*>
<!ATTLIST person id ID #REQUIRED sex (m | f) #IMPLIED>

<!ELEMENT name (#PCDATA)>

<!ELEMENT father EMPTY>
<!ATTLIST father %reference;>
```

The XML file is given by

```
<?xml version="1.0"?>
<!-- familytree.xml -->
<!-- Each person is identified by a unique ID -->

<!DOCTYPE familytree SYSTEM "familytree.dtd">

<familytree>
     <person id="p1" sex = "m">
       <name> Olaf </name>
     </person>

     <person id="p2" sex = "m">
       <name> Theo </name>
       <father person = "p1" />
     </person>

     <person id="p3" sex = "m">
       <name> Ario </name>
       <father person = "p1" />
     </person>

     <person id="p4" sex = "m">
       <name> Carl </name>
       <father person = "p2" />
     </person>
</familytree>
```

Each person is identified by a unique `id`. This will be invaluable for later processing of individuals. For processing the XML file we have to provide an XSL file.

2.4 Displaying XML using Data Binding

Internet Explorer 5 and 6 contain a DSO (Data Source Object) that permits displaying the data in an XML document on an HTML page by using the technology of client-side data binding. This involves taking the XML data from the server-side, transferring the data to the client side for future access, and binding the data to an HTML tag. The data is treated like a record set in ADO (ActiveX Data Object) and the data can be manipulated by using the various methods and properties of the ADO `Recordset` object. In our example, we use these features to navigate through our data. Once the data is bound to an HTML tag, it can be viewed on IE 5 and 6. There are fifteen HTML tags that can be used for data binding:

```
a  applet  button div  frame  iframe  img  input
label  marquee  object  select  span  table  textarea
```

These fifteen tags can use two special proprietary attributes, `datasrc` and `datafld`, that are only recognized on IE 4, IE 5 and IE 6.

`<span>` or `<div>` elements can be used to display XML data.

The `datasrc` attribute allows the linking of the HTML tag to the XML data document. The `datafld` attribute is used to name the XML data field that we wish to display.

These attributes are very easy to use and only require one line of code to access an XML data value. The sign `#` signifies a link. Note that this is a complete HTML element with both an opening and closing tag.

For our data binding example we use the `MyXML.xml` file with the `library` example. We want to create an HTML page that displays the XML data for the

```
title  subtitle  author  ISBN  publisher  city  year
```

We want to be able to scroll through the list of books using navigation buttons and see the data for any book. To do so, we use the `src` attribute of an inline `xml` element to link our HTML page to the XML document. We use the HTML `span` tag for the data binding. We use the `datasrc` attribute to link each `span` element to the `xml` element. We use the `datafld` attribute to bind the specified XML data value to the `span` element. We use previous and next `input` buttons to call data navigation functions. We use `MoveNext()` and `MovePrevious()` ADO methods to navigate our data. We apply `RecordCount` and `AbsolutePosition` ADO properties to limit our data navigation so that we never go to `BOF` or `EOF`.

```
<HTML>
<TITLE> Data Binding </TITLE>
<BODY>
<xml id="xmlLibrary" src="file://c:\books\java\library.xml">
</xml>

<B> Use the buttons to scroll up and down the library </B>
<BR><BR>

TITLE: <span datasrc="#xmlLibrary" datafld="title"></span>
<BR>
SUBTITLE: <span datasrc="#xmlLibrary" datafld="subtitle"></span>
<BR>
AUTHOR: <span datasrc="#xmlLibrary" datafld="author"></span>
<BR>
ISBN: <span datasrc="#xmlLibrary" datafld="ISBN"></span>
<BR>
PUBLISHER: <span datasrc="#xmlLibrary" datafld="publisher"></span>
<BR>
CITY: <span datasrc="#xmlLibrary" datafld="city"></span>
<BR>
YEAR: <span datasrc="#xmlLibrary" datafld="year"></span>
<BR>

<input type="button" value="Previous" onclick="Previous()">
<input type="button" value="Next" onclick="Next()">

<SCRIPT LANGUAGE="JavaScript">

function Previous()
{
  if(xmlLibrary.recordset.AbsolutePosition > 1)
     xmlLibrary.recordset.movePrevious();
}

function Next()
{
  if(xmlLibrary.recordset.AbsolutePosition <
                xmlLibrary.recordset.RecordCount)
     xmlLibrary.recordset.moveNext();
}
</SCRIPT>
</BODY>
</HTML>
```

2.5 Displaying XML Using XSL

XSL is the acronym for the *Extensible Stylesheet Language* which is an application of XML. XSL is a powerful styling language that uses special stylesheets to create templates to display the data contained in an XML page in a variety of ways. We use XSL to transform an XML document into an HTML page that can be viewed on Internet Explorer. We do this by creating a separate XSL document that is linked to the XML document. XSL is composed of two parts: XSLT which stands for XSL Transformation and is used to transform from an XML doucment to another format which may be HTML, PDF or LaTEX. XSLFO which stands for XSL Formatting which is the native XML equivalent of Cascading Style Sheets (CSS). XSLFO is thus reponsible for rendering information in a visual way.

This technology can be used with IE 5 and IE 6. The Microsoft version and XSL does not fully support all the recommendations set for this language by the World Wide Web Consortium (W3C). However, XSL can be used on the server-side so that the output is browser independent.

XSL is not a large language. It is composed of twenty tag-like elements and associated attribute-like methods. Like XML, all XSL elements must have an opening and closing tag. All XSL tags have the same namespace prefix, `xsl:`, to denote that this is an XSL element. Notice the use of the colon. After the prefix, the suffix designates the tag's purpose.

In order to display our XML document, we only need three of the XSL elements.

`xsl:template`

The `xsl:template` element is used to define a template. It can also be used as a container element to declare the start and stop of an XSL coding sequence. We use it in this manner in our example. In order for this to work in IE 6, we must use the following code exactly

```
<xsl:template xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

The next element we use is

`xsl:for-each`

This element is used to create a `for ... each` loop which allows looping through all the values for an XML data field. We use the `select` attribute to specify the name of the XML data element.

`xsl:value-of`

The `xsl:value-of` element is used to insert the value of the XML data field into the template. We use the `select` attribute to specify the name of the XML data field. The `xsl:value-of` element allows us to display the data value for an XML tag.

An example is given in the following. The `Library.xml` file is given by

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/xsl" href="library.xsl"?>

<!-- File name: library.xml -->
<!-- address: file://c:\books\java\library.xml -->

<library>
  <book>
    <title> SymbolicC++ </title>
    <subtitle> An Introduction to Computer Algebra </subtitle>
    <author> Tan Kiat Shi, Willi Hans Steeb, Yorick Hardy </author>
    <ISBN> 1852332603 </ISBN>
    <publisher> Springer </publisher>
    <city> London </city>
    <year> 2000 </year>
  </book>

  <book>
    <title> Classical and Quantum Computing </title>
    <subtitle> with C++ and Java Simulations </subtitle>
    <author> Yorick Hardy and Willi Hans Steeb </author>
    <ISBN> 3764366109 </ISBN>
    <publisher> Birkhauser Publishing </publisher>
    <city> Basel </city>
    <year> 20001 </year>
  </book>

</library>
```

The file `library.xsl` is given by

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
         xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">

<HTML>
<BODY>
<xsl:for-each select="library/book">
<DIV>
TITLE: <xsl:value-of select="title"/>
<BR/>
AUTHOR: <xsl:value-of select="author"/>
<BR/>
YEAR: <xsl:value-of select="year"/>
<HR/>
</DIV>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

To put an conditional `if` test against the content of the file, we simply add an `xsl:if` element to our document. In the example we consider the table for a sporting competition

```
 name           points
======================
 Snowbird John   6234
 Adler Jack      3234
 Eagle Carl       134
======================
```

Each row of the table is identified as a

```
<result> ... </result>
```

block for the xml-file. We only want the display the table for the participants who have more than 200 points.

The file `if.xml` is given by

```
<?xml version="1.0"?>

<!-- File name: If.xml -->
<!-- address: file://c:\books\java\if.xml -->

<?xml-stylesheet type="text/xsl" href="if.xsl"?>

<ranking>
  <result>
    <name> Snowbird John </name>
    <points> 6234 </points>
  </result>

  <result>
   <name> Adler Jack </name>
   <points> 3234 </points>
  </result>

  <result>
   <name> Eagle Carl </name>
   <points> 134 </points>
  </result>

</ranking>
```

The `if.xsl`-file is given by

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- address: file://c:\books\java\if.xsl -->

<xsl:template match="/">
<HTML>
<BODY>
<TABLE border="2">
  <TR>
  <TH> Name </TH>
  <TH> Points </TH>
  </TR>

  <xsl:for-each select="ranking/result">
    <xsl:if test="points &gt; 200">
    <TR>
    <TD><xsl:value-of select="name"/></TD>
    <TD><xsl:value-of select="points"/></TD>
   </TR>
  </xsl:if>
  </xsl:for-each>
  </TABLE>
  </BODY>
  </HTML>
 </xsl:template>

</xsl:stylesheet>
```

The following two programs show an application of `xsl:sort`. The attributes we can use are

```
order = "ascending/descending"
lang = "..."
data-type = "text/number"
case-order = "upper-first/lower-first"
```

The file `sort.xml` is given by

```
<?xml version="1.0"?>

<!-- File name: Sort.xml -->
<!-- address: file://c:\books\java\sort.xml -->

<?xml-stylesheet type="text/xsl" href="sort.xsl"?>

<ranking>
  <player>
    <name> Snowbird </name>
    <points> 723 </points>
  </player>

  <player>
   <name> Adler </name>
   <points> 323 </points>
  </player>

  <player>
   <name> Eagle </name>
   <points> 173 </points>
  </player>

</ranking>
```

The file `sort.xsl` is given by

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<HTML>
<BODY>
  <TABLE border="2">
  <TR>
  <TH> Name </TH>
  <TH> Points </TH>
  </TR>

  <xsl:for-each select="ranking/player">
  <xsl:sort select="points" order="descending" data-type="number"/>
  <TR>
  <TD><xsl:value-of select="name"/></TD>
  <TD><xsl:value-of select="points"/></TD>
  </TR>
  </xsl:for-each>
</TABLE>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

2.6 Using org.w3c.dom.*

The Document Object Model (DOM) 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. The document can be further processed and the results of that processing can be incorporated back into the presented page. DOM, the offical W3C API for XML, builds an internal model for the XML document which is a tree. The web site is

```
http://www.w3.org
```

In order to construct the tree it has to read the XML document sequentially, extracting elements, attributes and name space definitions as well as processing instructions.

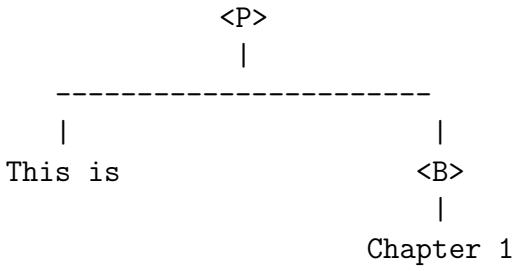
In the Level 1 DOM, each object, whatever it may be exactly, is a Node. For example

```
<P> This is Chapter 1 </P>
```

has created two nodes: an element `P` and a text node with the content `This is Chapter 1`. The text node is inside the element, so it is considered a child node of the element. Conversely, the element is considered the parent node of the text node. If we consider

```
<P> This is <B> Chapter 1 </B> </P>
```

the element node `P` has two children, one of which has a child of its own.



Next we show how to display image metadata with `ImageIO` as XML-file using DOM. Given a jpeg-file `Image0.jpg`. `IIOMetadata` contains meta information about the image, so not the actual pixels, but the compression, keywords, comments, etc. If we convert from one format to another, we do not want to loose this information. A ImageTranscoder understands this meta data and maps it onto another format. Internally, Metadata is stored as a bunch of IIOMetadataNodes. They implement the `org.w3c.dom` interface. The format of this DOM tree is plug-in dependent: the native format (as format features are different), but plug-ins may support the plug-in neutral format.

The following example program displays, using the XSLT transformation package, the plug-in neutral format.

```
// Main.java

import javax.imageio.metadata.*;
import javax.imageio.stream.*;
import javax.imageio.*;

import java.awt.image.*;
import java.util.*;
import java.io.*;

import javax.xml.transform.stream.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.*;
import javax.xml.parsers.*;

import org.w3c.dom.*;    // for Node class

public class Main
{
  public static void main(String []args) throws Exception
  {
  Iterator readers = ImageIO.getImageReadersByFormatName("jpg");
  ImageReader reader = (ImageReader) readers.next();

  ImageInputStream iis =
         ImageIO.createImageInputStream(new File("Image0.jpg"));
  reader.setInput(iis,true);
  BufferedImage bi = reader.read(0);

  IIOMetadata metadata = reader.getImageMetadata(0);
  Node node =
   (Node) metadata.getAsTree(metadata.getNativeMetadataFormatName());

  // use the XSLT transformation package
  // to output the DOM tree we just created
  TransformerFactory tf = TransformerFactory.newInstance();
  Transformer transformer = tf.newTransformer();
  transformer.transform(new DOMSource(node),new StreamResult(System.out));
  }
}
```

Chapter 3

JavaScript

3.1 Introduction

The HTML language used to create pages for the World Wide Web was originally designed to produce plain and static documents, stuff like engineering notes and long-winded arguments by scientist types. When the Web first started, the only browsing software available for it was text-based.

JavaScript makes HTML come alive. JavaScript is a scripting language for HTML and the Netscape Navigator browser, version 2.0 and later. JavaScript is a cross-platform, object-oriented language. Core JavaScript contains a core set of objects as `Array`, `Date`, and `Math`, and a core set of language elements such as operators, control structures, and statements. JavaScript "scripts" are small programs that interact with Netscape and the HTML content of a page. We can create a JavaScript program to add sound or simple animation, pre-validate a form before the user's response is sent to our company's server, search through a small database, set options based on user preferences, and much more. JavaScript performs the same function as a macro in a word processor or electronic spreadsheet program. A macro is a small program designed solely to run inside a program, automating some task or enhancing a feature of the program. The difference here is that instead of a word processor or electronic spreadsheet application, JavaScript is designed for use with Netscape and surfing on the World Wide Web.

JavaScript is used in Netscape 2.0 and later versions, as well as in Microsoft Internet Explorer 3.0 and later versions. As the co-developer of JavaScript, Netscape has wanted to make JavaScript an open'standard, meaning that other companies can use and implement JavaScript in their own Internet products. When JavaScript was first announced in December of 1995, over two dozen companies jumped on the bandwagon, promising to support it in future products.

Thus JavaScript is an authoring language for the typical Web page designer. It is a scripting language in which the program is embedded as part of the HTML document retrieved by the browser.

A JavaScript program consists of one or more instructions (also referred to as code or commands) included with the HTML markup tags that form our Web documents. When Netscape encounters a JavaScript instruction, it stops to process it. For example, the instruction might tell Netscape to format and display text and graphics on the page. Unlike a program witten in Java, JavaScript programs are not in separate files (though this is an option using Netscape 3.0 and later). Instead, the JavaScript instructions are mixed together with familiar HTML markup tags such as `<H1>, <P>`, and `<IMG>`.

Because JavaScript is embedded in HTML documents, we can use any text editor or Web page editor to write our JavaScript programs. The only requirement is that the editor must allow direct input.

Netscape needs to be told that we are giving it JavaScript instructions, and these instructions are enclosed between

```
<SCRIPT> ... </SCRIPT>
```

tags. Within the script tag we can have only valid JavaScript instructions. We cannot put HTML tags for Netscape to render inside the `<SCRIPT>` tags, and we cannot put JavaScript instructions outside the `<SCRIPT>` tags.

JavaScript is designed on a simple object-based paradigm. An object is a construct with properties that are JavaScript variables or other objects. An object also has functions associated with it that are known as the object's methods. In addition to objects that are predefined in the Navigator client and the server, we can define or own objects. A JavaScript object has properties associated with it. We access the properties of an object with a simple notation

```
objectName.propertyName
```

Note that JavaScript is case-sensitive. Comments in JavaScript are the same as in C, C++ and Java, namely

```
// ...
```

and

```
/* ... */
```

Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example: Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model. Server-side JavaScript extends the core language by suppling objects relevant to running JavaScript on a server.

3.2 Document Object

The `document` object contains information about the current document, such as its title, when it was last modified, and the color of the background.

The `document` methods interact with the document. The most commonly used `document` method is

```
document.write("text");
```

and

```
document.writeln("text");
```

which writes text to the browers window. The document objects is itself a property of the window object. JavaScript assumes that we mean the current window when we use the syntax given above. Thus the `write` and `writeln` methods insert text into the document. The two methods are identical, except that `writeln` appends a new line character to the end of the string.

The command

```
document.bgColor = "red";
```

sets the background color to red. As an exmple of these two commands consider

```
<HTML>
<HEAD>
<TITLE> document.html </TITLE>
</HEAD>
<BODY>

<SCRIPT>

document.write("Hello Egoli");
document.write("\n");

str1 = "willi";
str2 = "hans";
document.write("<P> str1 = " + str1 + " str2 = " + str2);

document.bgColor = "red";

</SCRIPT>
</BODY>
</HTML>
```

The concatention operator + concatenates two strings values together.

The following program shows that the `prompt` provides a string even if we enter digits. Then + will do concatenation. Thus to obtain numbers we use the `Number` object.

```
<HTML>
<SCRIPT>

n = prompt("enter n: ","");
m = prompt("enter m: ","");

x = n + m;
document.write("x = " + x,"<BR>"); // concatenates strings n and m

u = Number(n);
v = Number(m);
r = u + v;
document.write("r = " + r,"<BR>"); // addition of numbers

</SCRIPT>
</HTML>
```

3.3 Window Object

The `window` object represents the contents of an entire browser window. The method `open()` is a method to open a new window.

```
win = window.open("OpenMe.html");
```

fills it with the specified document. The user interface methods are special methods of the `window` object. They all display a dialog box asking for user input. JavaScript has three ready-made dialog boxes that we can use to interact with our visitors. All three display a (fixed) title and a message, which we can set, and hold execution of the script until the user responds. The dialog boxes are

```
alert(message)   confirm(message)   prompt(message,default)
```

The most important one is

```
alert("message");
```

We use this method whenever we wish to display a message. This message is diplayed in a dialog box. The user reads the message and chooses `OK` to dismiss the dialog box.

The `confirm(message)` carries an `OK` and a `Cancel` button, and returns the value `true` or `false`, depending upon which button was clicked. We apply it when we are offering our visitor a simple Yes/No choice. We use `confirm()` in an `if()` test.

The dialog box `prompt(message,default)` carries a text box and `OK` and `Cancel` buttons. It returns whatever text was entered. We use it to obtain information on pages where we do not want to have a form.

```
<HTML>
<HEAD>
<TITLE> window.html </TITLE>
</HEAD>

<BODY>
<SCRIPT>
alert("System dialog boxes");
if(confirm("Do you want to do this?"))
  alert("Let us begin")
else
  alert("In any case we do it")
lnum = prompt("Enter your lucky number",""); // "" clears text box
</SCRIPT>
</BODY>
</HTML>
```

3.4 Data Types

Data types must not be declared we just define them. JavaScript recognizes the following types of values:

numbers, such as 18 or 2.81

logical (boolean) values, either `true` or `false`

strings such as `willi`. The `String` literal is zero or more characters enclosed in double " or single ' quotation marks.

`null` (`null` value). a special keyword denoting a null value. `null` is also a primitive value. Since JavaScript is case sensitive, `null` is not the same as `Null`, `NULL`, or any other variants. The `null` value behaves as 0 in numeric context and as `false` in boolean context.

`undefined`, a top-level property whose value is undefined. `undefined` is also a primitive value.

The assignment operator `=` assigns a value to its left operand based on the value of its right operand. For example

```
i = 4;
x = 3.14159;
found = true;
nfound = false;
str = "xxxyyy";
c = 'X';
```

Variables can also be declared using the keyword `var`. For example

```
var username;
var max = 500;
```

The first line creates the variable `username`, but without giving it a value; the second creates `max` and assigns 500 to it.

The rules for variable names are the same as those for object names. Names must start with a letter or underscore and may contain any combinations of letters and digits. Spaces cannot be used.

```
<HTML>
<HEAD>
<TITLE> DataTypes.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>
i = 4;
x = 3.14159;
c = 'X';
found = true;
nfound = false;
str = "xy";
document.write("i = " + i,"<BR>");
document.write("x = " + x,"<BR>");
document.write("c = " + c,"<BR>");
document.write("found = " + found,"<BR>");
document.write("nfound = " + nfound,"<BR>");
document.write("str = " + str,"<BR>");

n = null;
document.write("n = " + n,"<BR>"); // => null
div = n/4;
document.write("div = " + div,"<BR>"); // => 0

r = "The answer is " + 60;
document.write("r = " + r,"<BR>");
s = 60 + " is the answer";
document.write("s = " + s,"<BR>");

myList = ["Berlin",  , "London", "Singapore"];
document.write("myList[1] = " + myList[1],"<BR>"); // => undefined

</SCRIPT>
</BODY>
</HTML>
```

3.5 Special Characters

A string literal is zero or more characters enclosed in double " or single ' quotation marks. In addition to ordinary characters, we can also include special characters in strings. The special characters are

```
Character  Meaning
=========  =========
\b         backspace
\f         form feed
\n         newline
\r         carriage return
\t         tab
\'         apostrophe or single quote
\"         double quote
\\         backslash character \
\XXX       character with the Latin-1 encoding specified by up
           three octal digits XXX between 0 and 377.
           \251 is the octal sequence for the copyright symbol
\xXX       character with the Latin-1 encoding specified by the
           two hexadecimal digits XX between 00 and FF.
           \xA9 is the hexadecimal sequence for the copyright
           symbol.
================================================================
```

The following program shows an application

```
<HTML>
<HEAD>
<TITLE> Special.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>
str1 = "one line \n another line";
document.write("str1 = " + str1,"<BR>"); // one line another line

s = "one line <BR> another line";
document.write("s = " + s,"<BR>"); // s = one line
                                   // another line
str2 = "a tab \t in a string";
document.write("str2 = " + str2,"<BR>");

str3 = "She reads \"The Nonlinear Workbook\" by W.-H. Steeb"
```

```
document.write("str3 = " + str3,"<BR>");

symbol1 = "\251"
document.write("symbol1 = " + symbol1,"<BR>"); // @

symbol2 = "\xA9"
document.write("symbol2 = " + symbol2,"<BR>"); // @

</SCRIPT>
</BODY>
</HTML>
```

3.6 Arithmetic Operations

The arithmetic operations are

```
+  addition
-  subtraction
*  multiplication
/  division
%  modulus (remainder)
++ increment operator
-- decrement operator
```

As in C, C++ and Java the operators `++` and `--` have a preincrement (predecrement) and postincrement (postdecrement) version. We notice that division is floating point division. Thus `1/4` returns `0.25`. JavaScript also has the shorthand operators `x += y` for `x = x + y` etc. The following program shows how the arithmetic operators are used.

```
<HTML>
<HEAD>
<TITLE> JSArith.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>
i = 7;
j = -3;
k = i + j;
document.write("<BR> k = " + k + "<BR>");
m = i - j;
document.write("<BR> m = " + m + "<BR>");
n = i*j;
document.write("<BR> n = " + n + "<BR>");
p = i/j;
document.write("<BR> p = " + p + "<BR>");
r = i%j;
document.write("<BR> r = " + r + "<BR>");
i++;
document.write("<BR> i = " + i + "<BR>");
j--;
document.write("<BR> j = " + j + "<BR>");
</SCRIPT>
</BODY>
</HTML>
```

3.7 Comparison Operators

A comparison operator compares its operands and returns a logical value based on whether the comparision is true. The operands can be numerical or string values. Strings are compared based on the standard lexicographical ordering (ASCII table). The following table describes the comparision operators

```
Operator  Name          Description
========  ============  ========================================
==        equal         returns true if the operands are equal
!=        not equal     returns true if the operands are not equal
>         greater than  returns true if the left operand is
                        greater than the right operand
>=        greater than  returns true if the left operand is
          or equal      greater than or equal to the right operand
<         less than     returns true if the left operand is
                        less than the right operand
<=        less than     returns true if the left operand is
          or equal      less or equal to the right operand
=================================================================
```

```
<HTML>
<HEAD>
<TITLE> Comparison.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>
result1 = (3 == 3);
document.write("result1 = " + result1,"<BR>"); // => true

result2 = (3 != 3);
document.write("result2 = " + result2,"<BR>"); // => false

result3 = (4 > 2);
document.write("result3 = " + result3,"<BR>"); // => true

</SCRIPT>
</BODY>
</HTML>
```

3.8 Bitwise Operators

Bitwise operators treat their operands as a set of 32 bits (zeros and ones). For example the decimal number 14 has the binary representation

```
14 -> 00000000 00000000 00000000 000001110
```

The bitwise operators are the bitwise AND `&`, the bitwise OR `|`, the bitwise XOR `^`, and the bitwise NOT `~`. Furthermore we have the left shift `<<`, the sign-propagating right shift `>>` and the zero-fill right shift `>>>`.

```
<HTML>
<HEAD>
<TITLE> Bitwise.html </TITLE>
</HEAD>

<BODY>
<SCRIPT>
m = 9;
n = 16;
p1 = m & n;  // AND
p2 = m | n;  // OR
p3 = m ^ n;  // XOR
p4 = ~m;     // NOT
p5 = ++p4;   // increment by 1

document.write("p1 =" + p1,"<BR>"); // => 0
document.write("p2 =" + p2,"<BR>"); // => 25
document.write("p3 =" + p3,"<BR>"); // => 25
document.write("p4 =" + p4,"<BR>"); // => -9
document.write("p5 =" + p5,"<BR>"); // => -9

q1 = 31 >> 2;
document.write("q1 =" + q1,"<BR>"); // => 7
q2 = 3 << 3;
document.write("q2 =" + q2,"<BR>"); // => 24
q3 = -9 >> 2;
document.write("q3 = " + q3,"<BR>"); // => -3
q4 = -9 >>> 2;
document.write("q4 = " + q4,"<BR>"); // => 1073741821

</SCRIPT>
</BODY>
</HTML>
```

The following program shows how the bitwise operation `&` and the shift operations are applied for the *Russian farmer multiplication*.

```
<HTML>
<SCRIPT>

x = prompt("Enter m: ","");
y = prompt("Enter n: ","");

m = Number(x);
n = Number(y);

if(m == 0) document.write("0","<BR>");
if(n == 0) document.write("0","<BR>");
if(m == 1) document.write(n,"<BR>");
if(n == 1) document.write(m,"<BR>");

temp = 0;

while(m != 0)
{
  document.write(m,"<BR>");
  document.write(n,"<BR>");
  if((m & 1) == 1)
  temp = n + temp;
  document.write("temp:" + temp,"<BR>");
  m = m >> 1; n = n << 1;
}

document.write(temp);

</SCRIPT>
</HTML>
```

3.9 Program Flow

JavaScript supports a compact set of statements we can use to control the program flow. They are

conditional statements: `if ... else` and `switch`

loop statements: `for`, `while`, `do ... while`, `label`, `break`, `continue`. Note that `label` is not itself a looping statement, but is frequently used with these statements.

There is no `goto` in JavaScript.

The `if ... else`, `switch`, `for`, `while`, `do ... while` conditions have the form as in Java.

The logical operators

```
&&   logical AND
||   logical OR
!    logical NOT
```

also have the same form as in Java, C and C++. Furthermore the *relational operators* are

```
==  equal to   < less than  <= less than or equal to
!=  not equal to  > greater than  >= greater than equal to
```

An example is

```
if((i%3) == 0) { ... }
```

The `for` loop and `while` loop are the same as in Java. For example

```
...
for(n=1;n<10;n++)
{
  x = i*n;
}
```

The `break` statement provides a way of escaping from `for` or `while` loops. For example

```
for(count=0;count<5;count++)
{
   answer = 4;
   if(answer == (count*count))
   break;
}
```

The following program uses two `for` loops to generate a triangle of asterisks.

```
<HTML>
<COMMENT> asterisk.html </COMMENT>
<SCRIPT>

n = 9;
for(i=1;i<n;i++)
{
   for(j=1;j<=i;j++)
   {
   document.write("*");
   }
   document.write("<BR>");
}

</SCRIPT>
</HTML>
```

A `switch` statement allows a program to evaluate an expression and attempt to match the expression's value to a case label. If a match is found, the program executes the associated statement.

```
<HTML>
<COMMENT> MSwitch.html </COMMENT>
<SCRIPT>

MArray = [ 3, 4, 1, 1, 4, -3, 1, 6 ];

for(j=0;j<MArray.length;j++)
{
  switch(MArray[j])
  {
  case 1: document.write("number is 1","<BR>")
  break;
  case 3: document.write("number is 3","<BR>")
  break;
  default: document.write("number is not 1 or 3","<BR>")
  break;
  }
}
</SCRIPT>
</HTML>

<HTML>
```

```
<COMMENT> SSwitch.html </COMMENT>
<SCRIPT>

s = new String("bella");

for(j=0;j<s.length;j++)
{
  c = s.charAt(j);
  document.write("j = " + j, "<BR>");

  switch(c)
  {
  case 'a': document.write("character is 'a'","<BR>")
  break;
  case 'b': document.write("character is 'b'","<BR>")
  break;
  default: document.write("character is not 'a' or 'b'","<BR>")
  break;
  }
}
</SCRIPT>
</HTML>
```

The `continue` statement inside a `for` loop or `while` loop skips over any remaining lines and loops round again.

The following HTML file gives an example how the `for` loop and the `while` loop is used.

```
<HTML>
<HEAD>
<TITLE> JSLoop.html </TITLE>
</HEAD>

<BODY>
<SCRIPT>
document.write("Hello Egoli");

for(i=1;i<=10;i++)
{
  if((i%2) == 0)
  {
  document.write("<P>Loop: " + i + "</P>");
  }
}
alert("We are leaving the first Script block");
</SCRIPT>

<B> Now we use a script again </B>
<SCRIPT>
sum = 0;
count = 0;

while(count < 10)
{
   sum += count;
   count++;
}

document.write("<BR> sum = " + sum);
</SCRIPT>

<BR><BR>
<B> Again we enter script to set the background colour </B>
<SCRIPT>
document.bgColor = "red";
</SCRIPT>

</BODY>
</HTML>
```

JavaScript also has a `for ... in` statement. It uses it to manipulate objects. The form is

```
for(varname in objname)
{ forbody }
```

The following program shows an example of the `for ... in` statement.

```
<HTML>
<TITLE> forin.html </TITLE>

<BODY>
<SCRIPT>

languages = ["C++", "Java", "C", "Lisp"];
var language = "C";
var label = 0;

for(var i in languages)
{
   if(language == languages[i])
   {
   document.write("language in list on position: " + i,"<BR>");
   label = 1;
   }
}

document.write("label = " + label,"<BR>");

if(label == 0)
{ document.write("language not in list"); }

</SCRIPT>
</BODY>

</HTML>
```

The purpose of the `with` statement in JavaScript is to permit a number of object references to be made to the same object (or instance) without having to repeat the name of that object. A `with` statement looks as follows

```
with(object)
{
   statements
}
```

The following `with` statement specifies that the `Math` object is the default object.

```
<HTML>
<HEAD>
<TITLE> with.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>

var a = 3.0;
var b = 7.0;
var result1;
var result2;

with(Math)
{
   result1 = PI*a*b;
   result2 = sin(a)*sin(a) + cos(b)*cos(b);
}

document.write("result1 = " + result1,"<BR>");
document.write("result2 = " + result2,"<BR>");

</SCRIPT>
</BODY>
</HTML>
```

3.10 Recursion

Recursion plays a central role in computer science. A recursive function is one whose definition includes a call to itself. More generally, a recursive method is a method that calls itself either directly or indirectly through another method. A recursion needs a stopping condition. JavaScript allows recursion.

The following program shows how to use recursion for the generatimg of the Fibonacci numbers.

```
<HTML>
<COMMENT> recursion.html </COMMENT>

<SCRIPT>
x = prompt("Enter the position of the Fibonacci number","");
n = Number(x);

function fib(n)
{
  if(n<=2) return 1;
  else return (fib(n-1)+fib(n-2));
}

result = fib(n);
document.write("Fibonacci number is: " + result,"<BR>");

</SCRIPT>
</HTML>
```

3.11 Other JavaScript Constructs

The `new` operator creates a new copy of an object. We can use the `new` operator to create an instance of a user-defined object type or one of the predefined object types

`Arrays, Boolean, Date, Functions, Image, Number, Object,`

`Option, RegExp, String`

An example is

```
mystring = new String("xxyy");
```

```
now = new Date();
```

JavaScript also has the `this` pointer which refers to the current object. In general, `this` refers to the calling object in a method. The syntax is

```
this
```

```
this.object
```

The object name helps to disambiguate what `this` means. The `this` keyword is also often used to define a new property in a user-defined object. When combined with the `FORM` property, `this` can refer to the current object's parent form.

We use the `var` statement to explicitly declare a variable. We may also define a value for the variable at the same time we declare it, but this is not necessary. The `var` statement also sets the scope of a variable when a function is defined inside a function. For example

```
var name1 = "value";
name1 = "value";
```

Used outside of a user-defined function, both of these do exactly the same. Both create a global variable. A global variable can be accessed from any function in any window or frame that is currently loaded. If we use `var name1` inside a user-defined function it is local in scope, we only can see it inside the function.

The `void` operator specifies an expression to be evaluated without returning a value.

The `delete` operator deletes an object, an object's property, or an element at a specified index in an array.

3.12 Functions

JavaScript allows the use of functions indicated by the keyword `function`. The `var` statement inside a function makes the variable local, i.e. it goes out of scope when we leave the function.

The following HTML file show how functions are used within JavaScript. The program opens a window where we see

```
Type something here
```

a textbox and a button called `ClickMe`. We enter some text in the textbox, for example `Egoli-Gauteng`. Then we click at the Button `ClickMe`. The method `alert` opens a dialog box and displays

```
The value of the textbox is: Egoli-Gauteng
```

Then click OK to close the dialog box.

```
<HTML>
<COMMENT> MyForm.html </COMMENT>

<BODY>

<SCRIPT LANGUAGE="JavaScript">

function testMe(form)
{
   Ret = form.box.value;
   alert("The value of the textbox is: " + Ret);
}

</SCRIPT>

<FORM>
Type something here <INPUT TYPE="text" NAME="box"><P>
<INPUT TYPE="button" NAME="button" VALUE="ClickMe"
onClick = "testMe(this.form)">
</FORM>

</BODY>
</HTML>
```

In the following example we show the use of two functions. The HTML commands to display a table of the square roots of the integers from 1 to 20 are embedded in the JavaScript code.

```
<HTML>
<COMMENT> Function.html </COMMENT>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">

  function sqrttable(myWindow)
  {
  myWindow.document.write("<B> Square Root Table </B><BR>");
  myWindow.document.write("<HR>");
  myWindow.document.write("<TABLE BORDER=1 CELLSPACING=5>");
  myWindow.document.write("<TR><TD>Number</TD><TD>Squareroot</TD></TR>");

  for(var i=1;i<=20;i++)
  {
  myWindow.document.write("<TR><TD>" + i + "</TD><TD>" + Math.sqrt(i) +
                          "</TD></TR>");
  }
  myWindow.document.write("</TABLE>");
  myWindow.document.write("<HR>");
  }

  function display()
  {
  var dynWindow = window.open("","Title","width=200,height=300,scrollbars",
                              + "resizable");
  sqrttable(dynWindow);
  }
</SCRIPT>
</HEAD>

<BODY>

<FORM name="Show it">
  <input type=button value="show table" onClick="display()">
</FORM>

</BODY>
</HTML>
```

3.13 Creating New Objects

Using a constructor function we can create an object with the following two steps

1. Define the object type by writing a constructor function

2. Create an instance of the object with `new`

To define an object, create a function for the object type that specifies its name, properties, and methods. The following HTML file gives an example.

```
<HTML>
<COMMENT> car.html </COMMENT>
<BODY>

<SCRIPT>

function car(make,model,year)
{
   this.make = make
   this.model = model
   this.year = year
}

mycar = new car("BMW","Z3",2000)
yourcar = new car("Volkswagen","Beetle",1999)
hiscar = new car("Audi","A6",2000)

document.write("<P> mycar = " + mycar.model)
document.write("<P> yourcar = " + yourcar.year)
document.write("<P> hiscar = " + hiscar.make)

</SCRIPT>
</BODY>
</HTML>
```

3.14 Predefined Core Objects

JavaScript provides the predefined core objects

```
Array, Boolean, String, Math, Number
```

In the following programs we show how the predefined objects are used.

An array is an ordered set of values that we refer to with a name and an index. The `Array` object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them. It has a property for determing the array length and other properties.

An array can be created as follows:

```
arrayObjectName = new Array(element0,element1,...,elementN)
```

or

```
arrayObjectName = new Array(arrayLength)
```

Array literals are also `Array` objects. For example the following literal is an `Array` object

```
cities = ["London", "Paris", "Berlin"]
```

The first program shows how the predefined core object `Array` is used.

```
<HTML>
<HEAD>
<TITLE> Array.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>
myArray = new Array("C++", "Lisp", "Java");
document.write(myArray.join(),"<BR>");
document.write(myArray.reverse(),"<BR>");

yourArray = new Array(3);
yourArray = myArray.sort();
document.write("yourArray = " + yourArray,"<BR>");

newArray = new Array("Assembler", "VHDL");
concatArray = new Array(5);
concatArray = myArray.concat(newArray);
document.write("concatArray = " + concatArray,"<BR><BR>");

x = new Array(2);
x[0] = 2;
x[1] = 5;
result = x[0]*x[1];
document.write(result,"<BR><BR>"); // => 10

document.write("length = " + x.length,"<BR><BR>"); // => 2

a = new Array(4)
for(i=0;i<4;i++)
{
  a[i] = new Array(4)
  for(j=0;j<4;j++)
  {
  a[i][j] = "["+i+","+j+"]"
  }
}

for(i=0;i<4;i++)
{
   str = "Row "+i+": "
   for(j=0;j<4;j++)
   {
```

```
   str += a[i][j];
   }
   document.write(str,"<P>")
}
</SCRIPT>
</BODY>
</HTML>
```

The `Boolean` object is a wrapper around the primitive `Boolean` data type.

```
<HTML>
<HEAD>
<TITLE> Boolean.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>
r = Math.random();
if(r < 0.5)
{
b = new Boolean(true);
}
else
{
b = new Boolean(false);
}

document.write("b = " + b,"<BR>");

</SCRIPT>
</BODY>
</HTML>
```

The `String` object is a wrapper around the string primitive data type. We should not confuse a string literal with the `String` object. For example

```
s1 = "egoli";             // creates a string literal value
s2 = new String("egoli"); // creates a String object
```

We can call any of the methods of the `String` object on a string literal value. JavaScript automatically converts the string literal to a temporary `String` object, calls the method, and then discards the temporary `String` object.

The following program shows an application of several methods of the `String` object.

```
<HTML>
<HEAD>
<TITLE> String.html </TITLE>
</HEAD>

<BODY>
<SCRIPT>
mystring = new String("Hello Egoli");
lgth = mystring.length;
document.write("lgth = " + lgth,"<BR><BR>"); // => 11

ch = mystring.charAt(3);
document.write("ch = " + ch,"<BR><BR>"); // => l

chcode = mystring.charCodeAt(3);
document.write("chcode = " + chcode,"<BR><BR>"); // => 108

sub = mystring.substring(2,6);
document.write("sub = " + sub,"<BR><BR>"); // => llo

lower = mystring.toLowerCase();
document.write("lower = " + lower,"<BR><BR>"); // => hello egoli

yourstring = new String("willi-hans");
upper = yourstring.toUpperCase();
document.write("upper = " + upper,"<BR><BR>"); // => WILLI-HANS

string1 = new String("carl-");
string2 = new String("otto");
string3 = string1.concat(string2);
document.write("string3 = " + string3,"<BR><BR>"); // => carl-otto
```

```
str = new String("12 34 45");
myarray = new Array(3);
myarray = str.split(" ");
document.write("myarray[0] = " + myarray[0],"<BR>"); // => 12
document.write("myarray[1] = " + myarray[1],"<BR>"); // => 34
document.write("myarray[2] = " + myarray[2],"<BR>"); // => 45

</SCRIPT>
</BODY>
</HTML>
```

The top-level predefined function `eval` evaluates a string of JavaScript code without reference to a particular object. The syntax is

```
eval(expr)
```

where `expr` is a string to be evaluated.

```
<HTML>
<HEAD>
<TITLE> Eval.html </TITLE>
</HEAD>

<BODY>

<SCRIPT>

s1 = "3 + 3";
s2 = new String("3 + 3");

document.write("result1 = " + eval(s1),"<BR>"); // => 6
document.write("result2 = " + eval(s2),"<BR>"); // => 3 + 3

</SCRIPT>
</BODY>
</HTML>
```

The predefined `Math` object has properties for mathematical constants and functions. Standard mathematical functions are methods of `Math`. For example

```
abs, sin, cos, tan, acos, asin, atan, exp, log

ceil, floor, min, max, pow, round, sqrt
```

An application is as follows:

```
<HTML>
<HEAD>
<TITLE> Math.html </TITLE>
</HEAD>

<BODY>
<SCRIPT>
document.write(Math.sin(1.14159),"<BR>");
document.write(Math.sin(Math.PI),"<BR><BR>");

r = 2.0; theta = 1.0;
with(Math)
{
   a = PI*r*r;
   y = r*sin(theta)
   x = r*cos(theta)
}
document.write("a = " + a,"<BR>");
document.write("y = " + y,"<BR>");
document.write("x = " + x,"<BR>");

z = Math.random();
document.write("z = " + z,"<BR>");

r1 = Math.max(3,4,2,7,1);
document.write("r1 = " + r1,"<BR>"); // 7
r2 = Math.min(3,1,10,20,2,3,9);
document.write("r2 = " + r2,"<BR>"); // 1

</SCRIPT>
</BODY>
</HTML>
```

The `Number` object has properties for numerical constants, such as maximum value, not-a-number (`NaN`), and infinity. We cannot change the values of these constants.

```
<HTML>
<HEAD>
<TITLE> Number.html </TITLE>
</HEAD>

<BODY>
<SCRIPT>

biggestNum = Number.MAX_VALUE;
document.write("biggestNum =" + biggestNum,"<BR>");

smallestNum = Number.MIN_VALUE;
document.write("smallestNum = " + smallestNum,"<BR>");

diff = biggestNum - smallestNum;
document.write("diff = " + diff,"<BR>");

infiniteNum = Number.POSITIVE_INFINITY;
document.write("infiniteNum = " + infiniteNum,"<BR>");
negInfiniteNum = Number.NEGATIVE_INFINITY;
document.write("negInfiniteNum = " + negInfiniteNum,"<BR>");

notANum = Number.NaN;
document.write("notANum = " + notANum,"<BR>");

</SCRIPT>

</BODY>
</HTML>
```

3.15 Object object

JavaScript supports a generic `Object()` object, which we can use to make new objects. The following two programs show how to create and use one-dimensional arrays using the `Object` object. In the first program we enter the colour and click the `Find` button. The hextriplet is displayed. In the second program we enter the name of the customer and click the `Find` button. This provides us with the complete information about the customer. The method

```
String toLowerCase()
```

converts a string to all lowercase.

```
<HTML>
<HEAD>
<TITLE> DataBase.html </TITLE>
</HEAD>
<BODY>

<SCRIPT LANGUAGE="JavaScript">

Colour = new Object()
Colour[0]=10
Colour[1]="aliceblue"
Colour[2]="antiquewhite"
Colour[3]="aqua"
Colour[4]="aquamarine"
Colour[5]="azure"
Colour[6]="beige"
Colour[7]="bisque"
Colour[8]="black"
Colour[9]="blanchedalmond"
Colour[10]="blue"

Data = new Object()
Data[1]="f0f8ff"
Data[2]="faebd7"
Data[3]="00ffff"
Data[4]="7fffd4"
Data[5]="f0ffff"
Data[6]="f5f5dc"
Data[7]="ffe4c4"
Data[8]="000000"
Data[9]="ffebcd"
Data[10]="0000ff"
```

```
function checkDatabase()
{
    var Found = false;
    var Item =  document.testform.color.value.toLowerCase();
    Count = 1;
    while(Count <= Colour[0])
    {
    if(Item == Colour[Count])
    {
    Found = true;
    alert("The hex triplet for '" + Item + "' is #" + Data[Count]);
    break;
    }
    Count++;
    }  // end while
    if(!Found)
    alert("The color '" + Item +"' is not listed in the database");
}
</SCRIPT>

<FORM NAME="testform" onSubmit="checkDatabase()">
Specify a color name, then click the
"Find" button to see its hex triplet:
<BR>
<INPUT TYPE="text" NAME="color" VALUE="" onClick=0> <P>
<INPUT TYPE="button" NAME="button" VALUE="Find"
            onClick="checkDatabase()">
</FORM>

</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE> DataBase2.html </TITLE>
</HEAD>

<SCRIPT LANGUAGE="JavaScript">

Names = new Object()
Names[0]=6
Names[1]="cooper"
Names[2]="smith"
Names[3]="jones"
Names[4]="michaels"
Names[5]="avery"
Names[6]="baldwin"

Data = new Object()
Data[1]="Olli|Cooper|44 Porto Street|666-000"
Data[2]="John|Smith|123 Main Street|555-1111"
Data[3]="Fred|Jones|PO Box 5|555-2222"
Data[4]="Gabby|Michaels|555 Maplewood|555-3333"
Data[5]="Alice|Avery|1006 Pike Place|555-4444"
Data[6]="Steven|Baldwin|5 Covey Ave|555=5555"

function checkDatabase()
{
   var Found = false;  // local variable
   var Item = document.testform.customer.value.toLowerCase();
   for(Count=1;Count<=Names[0];Count++)
   {
   if(Item == Names[Count])
   {
   Found = true;
   var Ret = parser(Data[Count], "|");
   var Temp = "";
   for(i=1;i<=Ret[0];i++)
   {
   Temp += Ret[i] + "\n";
   }
   alert(Temp);
   break;
   }
   }
   if(!Found)
   alert("Sorry, the name '" + Item +"' is not listed in the database.")
```

```
}  // end function checkDatabase()

function parser(InString,Sep)
{
   NumSeps=1;
   for(Count=1;Count<InString.length;Count++)
   {
   if(InString.charAt(Count)==Sep)
   NumSeps++;
   }
   parse = new Object();
   Start=0; Count=1; ParseMark=0;
   LoopCtrl=1;
   while(LoopCtrl==1)
   {
   ParseMark = InString.indexOf(Sep,ParseMark);
   TestMark = ParseMark+0;
   if((TestMark==0) || (TestMark==-1))
   {
   parse[Count]= InString.substring(Start,InString.length);
   LoopCtrl=0;
   break;
   }
   parse[Count] = InString.substring(Start,ParseMark);
   Start=ParseMark+1;
   ParseMark=Start;
   Count++;
   }
   parse[0]=Count;
   return (parse);
}  // end function parser()

</SCRIPT>
<FORM NAME="testform" onSubmit="checkDatabase()">

Enter the customer's name, then click the "Find" button:
<BR>
<INPUT TYPE="text" NAME="customer" VALUE="" onClick=0> <P>
<INPUT TYPE="button" NAME="button" VALUE="Find"
onClick="checkDatabase()">

</FORM>
</BODY>
</HTML>
```

3.16 Date Object

The `Date` object is used to work with dates and times. We create an instance of the `Date` object with the `new` keyword. To store the current date in a variable called `my_date`:

```
my_date = new Date()
```

After creating an instance of the `Date` object, we can access all the methods of the object from the `my_date` variable. We have to keep in mind that the method

```
getMonth()
```

returns the month of a `Date` object from 0-11, where 0=January, 1=February, etc.

The following program shows an example.

```
<HTML>
<HEAD> </HEAD>
<!-- File name: Today.html -->

<BODY>

<SCRIPT LANGUAGE="JavaScript">

document.write("<CENTER><H2> Today is </H2><BR>");

var date = new Date();
var dd = date.getDate();
var mm = date.getMonth() + 1;
var yy = date.getYear();

if(dd < 10) dd = "0" + dd;
if(mm < 10) mm = "0" + mm;
if(yy < 10) yy = "0" + yy;

document.write("<B>" + dd + "." + mm + "." + yy + "</B>");
document.write("</CENTER>");

</SCRIPT>

</BODY>
</HTML>
```

In the second program we test whether the present time is `am` or `pm`.

```
<HTML>
<HEAD>
<TITLE> clockset.html </TITLE>
</HEAD>

<SCRIPT LANGUAGE="JavaScript">

function setClock()
{
now = new Date();
var CurHour = now.getHours();
var CurMinute = now.getMinutes();
now = null;
if(CurHour >= 12)
{
CurHour = CurHour - 12;
Ampm = "pm";
}
else
Ampm = "am";
if(CurHour == 0)
CurHour = "12";
if(CurMinute < 10)
CurMinute = "0" + CurMinute
else
CurMinute = "" + CurMinute
CurHour = "" + CurHour;
Time = CurHour + ":" + CurMinute + Ampm
document.clocktext.clock.value = Time;
setTimeout("setClock()",1000*30)
}
</SCRIPT>

</HEAD>
<BODY onLoad = "setClock()">
<FORM NAME = "clocktext">
<INPUT TYPE="text" NAME="clock" VALUE="" SIZE=8>
</FORM>
</BODY>
</HTML>
```

3.17 Regular Expression

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the `exec` and `test` methods of `RegExp`, and the `match`, `replace`, `search` and `split` methods of `String`. We can construct regular expression either using the constructor function of the `RegExp` object, for example

```
re = new RegExp("abc+de")
```

or using an object initializer, for example

```
re = /abc+de/
```

Using the constructor function provides runtime compilation of the regular expression. Regular expression have two optional flags that allow for global and case insensitive searching. To indicate a global search we use the `g` flag. To indicate a case insensitive search we use the `i` flag. These flags can be used separatly or together in either order, and are included as part of the regular expression.

The methods that use regular expressions are:

```
exec   A RegExp methods that executes a search for a match in a string.
       It returns an array of information.

test   A RegExp method that tests for a match in a string.
       It returns true or false.

match  A String method that executes a search for a match in a string.
       It returns an array of information or null on a mismatch.

search A String method that tests for a match in a string.
       It returns the index of the match, or -1 if the search fails.

replace A String method that executes a search for a match in a string,
        and replaces the matched substring with the replacement
        substring.

split  A String method that uses a regular expression or a fixed
       string to break a string into an array of substrings.
```

Pattern matching in JavaScript and Perl is closely related.

There are a large number of special characters in regular expressions. In the following we list some of them.

The backslash `\` indicates that the next character is special and not to be interpreted literally. For example, `/b/` matches the character `b`. However `/\b/` means match a word boundary.

The character `^` indicates matches at the beginning of input or line. For example, `/^X/` does not match the `X` in `"in X"` but does match it in `"Xenia"`.

The character `$` indicates matches at end of input or line. For example, `t$` does not match the `t` in `"otto"`, but does match it in `"art"`.

Using the asterisk `*` we can test whether the preceding characters matches 0 or more times.

The `+` indicates matches of the preceding characters 1 or more times.

Using `?` we can find the matches of the preceding characters 0 or 1 time. For example, `/el?le?/` matches the `"el"` in `"angel"` and `"le"` in `"angle"`.

The command `(x)` matches `"x"` and remembers the match. Thus including parentheses in a regular expression pattern causes the corresponding submatch to be remembered. For example, `/x(y)x/` matches the string `"xyx"` and remembers `y`.

The commmand `x | y` matches either `x` or `y`.

Let `n` be a positive integer. Then `{n}` matches exactly `n` occurrences of the preceding character.

`[xyz]` is a character set. It matches any one of the enclosed characters. We can specify a range of characters by using a hyphen. For example, `[abcd]` is the same as `[a-d]`. The expression `[^xyz]` is a negated or complemented character set. It matches anything that is not included in the brackets. Again we can specify a range of characters by using a hyphen.

The command `[\b]` matches a backspace and `\b` matches a word boundary such as space. The command `\d` matches a digit character. This is equivalent to `[0-9]`. With the command `\D` we match any non-digit character. This is equivalent to `[^0-9]`. The command `\f` matches a form-feed, `\n` a line-feed, `\r` a carriage return, `\t` a tab.

With `\w` we can match any alphanumeric character including underscore. It is equivalent to `[A-Za-z0-9_]`.

```
<HTML>
<COMMENT> RegExp.html </COMMENT>
<BODY>

<SCRIPT>
re0 = /xyx/
result0 = re0.exec("czxyyyxer")
document.writeln(result0)
// => null

re1 = /xyx/
result1 = re1.exec("czxyxer")
document.writeln(result1)
// => xyx

re2 = /^X/
result2 = re2.exec("Xylon")
document.writeln(result2)
// => X

re3 = /^X/
result3 = re3.exec("ylonX")
document.writeln(result3)
// => null

re4 = /t$/
result4 = re4.exec("art")
document.writeln(result4)
// => t

re5 = /t$/
result5 = re5.exec("otto")
document.writeln(result5)
// => null

re6 = /le?/
result6 = re6.exec("angle")
document.writeln(result6)
// => le

re7 = new RegExp("a+")
result7 = re7.exec("Cbaabbaxc")
document.writeln(result7)
// => aa
```

```
re8 = /xy+x/
result8 = re8.exec("czxyyyxer")
document.writeln(result8)
// => xyyyx

re9 = /x(y+)x/
result9 = re9.exec("czxyyyxer")
document.writeln(result9)
// => xyyyx, yyy

re10 = /xx aa$\n/
result10 = re10.test("yybb ++")
document.writeln(result10)
// => false

//  \w matches any alphanumeric characters
//  \s matches a single character other than white space
re11 = /(\w+)\s(\w+)/
str1 = "willi hans"
result11 = str1.replace(re11,"$2,$1")
document.writeln(result11)
// hans willi

re12 = new RegExp("[a-z]")
result12 = re12.exec("12345ac")
document.writeln(result12)
// => a

</SCRIPT>

</BODY>
</HTML>
```

3.18 Prompts

The `prompt` is a special method of the window object. It displays a dialog box with a single-text box and `Cancel` and `OK` buttons

`prompt(message,defaultText)`

The word `prompt` tells JavaScript that a dialog box will appear on the screen. The word `message` is the text that is displayed in the prompt box. Lastly, the word `defaultText` is the text displayed in the text box.

```
<HTML>
<HEAD>
<COMMENT> prompt.html </COMMENT>
</HEAD>

<BODY bgcolor=yellow text=black>

<SCRIPT LANGUAGE="JavaScript">

var i;
var j;
var k;
i = 1;
j = 1;

document.write("<H4>");

var m = prompt("enter the number of Fibonacci numbers:",0);
document.write(i + "<BR>");
document.write(j + "<BR>");

k = i + j;
document.write(k + "<BR>");

for(var n=3;n<=m-1;n++)
{
i = j;
j = k;
k = i + j;
document.write(k + "<BR>");
}
</SCRIPT>
</BODY>
</HTML>
```

3.19 Events

JavaScript programs are typically event-driven. Events are actions that occur on the Web page, usually of something the user does. Examples are: a button click is an event, giving focus to a form element, resizing the page, submitting a form.

An event, then, is the action which triggers an event handler. The event handler specifies which JavaScript code to execute. Often, event handlers are placed within the HTML tag which creates the object on which the event acts. For example, a hyperlink is subject to a `mouseover` event, meaning that its event handler will be triggered when a mouse passes over the link. The JavaScript which is called by the event handler may be any valid JavaScript code: a single statement or a series of statements, although most often it is a function call.

The set of all events which may occur, and the particular page elements on which they can occur, is part of the Document Object Model (DOM), and not of JavaScript itself. Thus, Netscape and Microsoft do not share the exact same set of events, nor are all page elements subject to the same events between browsers.

The table below displays some of the most commonly used events supported in both DOM's.

Event	Occurs when ...	Event Handler
click	User clicks on form or link	onclick
change	User changes value of text, textarea, or select element	onchange
focus	User gives form element input focus	onfocus
blur	User removes input focus from form element	onblur
mouseover	User moves mouse pointer over a link or anchor	onmouseover
mouseout	User moves mouse pointer off of link or anchor	onmouseout
select	User selects form element's input field	onselect
submit	User submits a form	onsubmit
resize	User resizes the browser window	onresize
load	User loads the page in the Navigator	onload
unload	User exits the page	onunload

The following program shows an example.

```
<HTML>
<COMMENT> Event1.html </COMMENT>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">

window.onresize = message;

function message()
{
  alert("The window has been resized!");
}

</SCRIPT>
</HEAD>
<BODY>

Please resize the window.

</BODY>
</HTML>
```

In the following example we demonstrate the `onBlur` event and can see how it is possible to force a user to enter valid information into a form. The user is forced to enter a number from 0 to 9.

```
<HTML>
<COMMENT> onblur.html </COMMENT>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">

function isDigit(c)
{
return ((c >= "0") && (c <= "9"))
}

function checkValue(field)
{
   if(!isDigit(field.value))
   {
   alert("You must enter a digit from 0 to 9.");
   field.focus();
   }
   if((field.value.length < 0) || (parseInt(field.value) >= 10))
   {
   alert("You did not enter a digit from 0 to 9");
   field.focus();
   }
}
</SCRIPT>

<FORM>
Please enter a number from 0 to 9:
<INPUT TYPE=TEXT NAME="number" SIZE=3 onBlur="checkValue(this);">
<BR><BR>
Please enter your name:
<INPUT TYPE=TEXT NAME="name" SIZE=25>
</FORM>

</BODY>
</HTML>
```

The following example demonstrates the `onChange` event. We built a pulldown menu that allows users to jump to different websites or URLs.

```
<HTML>
<COMMENT> onchange.html </COMMENT>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">

function jumpTo(URL_List)
{
  var URL = URL_List.options[URL_List.selectedIndex].value;
  window.location.href = URL;
}

</SCRIPT>

<FORM>
<SELECT NAME="site" onChange="jumpTo(this);">
<OPTION VALUE="http://www.fhso.ch/~steeb"> Applied University Solothurn
<OPTION VALUE="http://issc.rau.ac.za"> ISSC
</SELECT>
</FORM>

</BODY>
</HTML>
```

The following example demonstrates the `onClick` event.

```
<HTML>
<COMMENT> onclick.html </COMMENT>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">

function informuser()
{
  alert("I told you not to press that button");
}

</SCRIPT>

<FORM>
<INPUT TYPE=button NAME="button" VALUE="Do not press this button"
    onClick="informuser();">
</FORM>

</BODY>
</HTML>
```

The following example demonstrates the `onFocus` event.

```
<HTML>
<COMMENT> onfocus.html </COMMENT>
<HEAD>

<SCRIPT LANGUAGE="JavaScript">

var alreadyWarned = false;

function displayHelp()
{
  if(!alreadyWarned)
  {
  alert("Enter your phone number in (xx) xxx-xxxx format");
  alreadyWarned = true;
  }
}
</SCRIPT>

<FORM>
<INPUT TYPE=text NAME="phonenumber" SIZE=15
    onFocus = "displayHelp();">
</FORM>

</BODY>
</HTML>
```

3.20 Java Applets and JavaScript

JavaScript can be used to control the behaviour of a Java applet without knowing much about the design of the applet. All public variables, methods, and properties are available for JavaScript access. Each applet in a document is reflected in JavaScript as

```
document.appletName
```

where `appletName` is the value of the `NAME` attribute of the `<APPLET>` tag.

The following Java program `Welcome.java` and the HTML program `Welcome.html` show an application. We define a

```
public void setString(String s)
```

method that accepts a string argument, assigns it to `myString`, and calls the `repaint()` method. The file `Welcome.java` is given by

```
// Welcome.java

import java.applet.Applet;
import java.awt.Graphics;

public class Welcome extends Applet
{
  String myString;

  public void init()
  {
  myString = new String("Welcome to ISSC")
  }

  public void paint(Graphics g)
  {
  g.drawString(myString,25,25);
  }

  public void setString(String s)
  {
  myString = s;
  repaint();
  }
}
```

The HTML file is given by

```
<HTML>
<COMMENT> Welcome.html </COMMENT>
<BODY>
<APPLET CODE="Welcome.class" NAME="Welcome" WIDTH=275 HEIGHT=135>
</APPLET>

<SCRIPT>
function insert()
{
  document.Welcome.setString(document.myform.str.value)
}
</SCRIPT>

<FORM NAME="myform">
<INPUT TYPE="button" VALUE="SET STRING" onClick="insert()">
<INPUT TYPE="text" SIZE="20" NAME="str">
</FORM>
</BODY>
</HTML>
```

3.21 JavaScript and XML

The Document Object Model (DOM) is the model that describes how all elements in the HTML page, like input fields, images, paragraphcs etc., are related to the topmost structure: the document itself. By calling the elements by its proper DOM name, we can influence it. The Level 1 DOM Recommendation has been developed by the W3C to provide any programming language with access to each part of an XML document. Thus we can use JavaScript to parse an XML document. Notice that DOM is based on a tree structure. Our XML file describes a polynomial. Our polynomial consists of two terms, namely

$$p(x, y) = 2x^3y^2 - 4y^5 \, .$$

```
<?xml version="1.0"?>

<!-- poly.xml -->

<polynomial>
     <term>
       <factor> 2 </factor>
       <variable>
       <name> x </name>
       <exponent> 3 </exponent>
       </variable>
       <variable>
       <name> y </name>
       <exponent> 2 </exponent>
       </variable>
     </term>

     <term>
       <factor> -4 </factor>
       <variable>
       <name> x </name>
       <exponent> 0 </exponent>
       </variable>
       <variable>
       <name> y </name>
       <exponent> 5 </exponent>
       </variable>
     </term>

</polynomial>
```

Knowing the exact structure of the DOM tree, we can walk through it in search of the element we want to find using the commands

```
parentNode, childNodes[0], childNodes[1], ... , firstChild, lastChild
```

The following HTML-file with the embedded JavaScript code finds elements in the tree structure of the polynomial.

```
<!-- polynomials -->

<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript" FOR="window" EVENT="ONLOAD">
Document = polynom.XMLDocument;
factor1.innerText=
  Document.documentElement.firstChild.firstChild.text;
name1.innerText=
  Document.documentElement.firstChild.childNodes[1].firstChild.text;
factor2.innerText=
  Document.documentElement.lastChild.firstChild.text;
</SCRIPT>
</HEAD>

<BODY>
<xml id="polynom" src="file://c:\books\java\poly.xml"></xml>

Factor 1: <span id="factor1"></span>
<BR>
Name 1: <span id="name1"></span>
<BR>
Factor 2: <span id="factor2"></span>
</BODY>
</HTML>
```

The output is:

```
Factor 1:  2
Name 1:    x
Factor 2: -4
```

3.22 Loading a js-file

Given a `js` file, for example the file `asterik.js`

```
n = 9;
for(i=1;i<n;i++)
{
  for(j=1;j<=i;j++)
  {
  document.write("*");
  }
  document.write("<BR>");
}
```

The file is loaded in a `HTML` file as follows

```
<HTML>

<SCRIPT type="text/javascript">

function staticLoad(url)
{
  document.write('<script src="',url,
                 '"type="text/JavaScript"><\/script>');
}

staticLoad("asterisk.js");

</SCRIPT>
</HTML>
```

In the following two programs `display.html` and `display.js` we display the time with the time interval of 1000ms (1 second).

```
<HTML>
<!-- display.js -->

<SCRIPT src="display.js">
</SCRIPT>

<BODY onload="show_time(); window.setInterval(show_time,1000);">

</BODY>
</HTML>
```

The `display.js` file is

```
function show_time()
{
  document.open();
```

```
  document.write(
  "<HTML>                                \n" +
  "                                      \n" +
  "<SCRIPT src=\"display.js\">           \n" +
  "<\/SCRIPT>                            \n" +
  "                                      \n" +
  "<BODY onload=\"window.setInterval(show_time,1000);\">  \n" +
  "                                                       \n" +
  "<B> Date: <\/B><BR>" + Date() + "<BR><BR>              \n" +
  "                                                       \n" +
  "<FORM>                                                 \n" +
  "<\/FORM>                                               \n" +
  "                                                       \n" +
  "<\/BODY>                                               \n" +
  "<\/HTML>");

  document.close();  // force render
}
```

Chapter 4

Resources and Web Sites

There are a large number of web sites which provide information, news and tutorials of Java, HTML and JavaScript. Furthermore there are web sites to download new versions of Java, HTML and JavaScript.

The main web sites of SUN Microsystems for Java are

`http://java.sun.com/`
`http://java.sun.com/jdc/`
`http://developer.java.sun.com/`

Another useful web site for Java, HTML and JavaScript is "The Web Developer's Virtual Library"

`http://wdvl.com/WDVL/About.html`

The WDVL has more than 2000 pages and some 6000 links.

The web site

`http://www.niu.edu/acad/english/htmlref.html`

provides an HTML reference guide. The web site

`http://developer.netscape.com/docs/`

provides links to technical manuals and articles for Java, JavaScript and HTML.

The offical word on the latest standard in HTML can be found on the web site

`http://www.w3.org/pub/WWW/MarkUp/MarkUp.html`

Information about XML can be found on the web sites

`http://www.w3.org/XML/`

`http://www.w3schools.com/`

Bibliography

[1] Horton Ivor, *Beginning Java 2*, WROX Press, Birmingham, UK, 1999

[2] Jaworski Jamie, *Java 1.2 Unleashed*, SAMS NET, Macmillan Computer Publishing, USA, 1998

[3] Johnson Marc, *JavaScript Manual of Style*, Macmillan Computer Publishing, USA, 1996

[4] McComb Gordon, *JavaScript Sourcebook*, Wiley Computer Publishing, New York, 1996

[5] Willi-Hans Steeb, *The Nonlinear Workbook: Chaos, Fractals, Cellular Automata, Neural Networks, Genetic Algorithms, Fuzzy Logic with C++, Java, SymbolicC++ and Reduce Programs*, World Scientific, Singapore, 1999

[6] Tan Kiat Shi, Willi-Hans Steeb and Yorick Hardy, *SymbolicC++: An Introduction to Computer Algebra Using Object-Oriented Programming*, 2nd edition Springer-Verlag, London, 2000

Index