Book 5

Develop Web Pages

U:\Book\Book_05.doc

Develop Web Pages

What to Read in This Part

5

Fehler! Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

1 Server Pages and Scripting

Server pages are interfaces to executable programs or server processes that are accessible through an HTTP request. There are different technologies for different platform, however the interface is always very similar and compliant to CGI. Scripting allows to insert program statements into static HTML code that is executed whenever the page is requested.

1.1 ASP, JSP and ABAP Servlets

There are a variety of concurring scripting technologies.

Scripting

The communication between a HTML web browser and a dynamic web page server is always directed through a scripting interface. Scripting means, that the HTML page contains at least a single line of code that is interpreted by the web servers script engines. Actually there is a big variety of different concurring script languages.

Active Server Pages ASP

Active Server Pages is Microsoft's scripting solution as it is found in the Internet Information Server. It is a designed as a plug-in solution, which means that the server allows to install an arbitrary scripting interface. Practically an ASP interfaces to Microsoft's own scripting language: Visual Basic Script.

Java Server Pages JSP

Java server Pages embed server-side Java code. The HTML page contains calls to what is called Enterprise Java Beans. This solution is supported by most UNIX based web servers and by the IBM Websphere web server.

Java Servlets

These are JAVA object components designed to be called by an HTTP web interface and to return valid HTTP messages to the requester. Servlets are written in JAVA and compiled into JAVA packages. They are an immediate and superior replacement of CGI components. Please mind the difference between an applet and a servlet:

A servlet runs on the application server and returns valid presentation code, e.g. an HTML page

An applet runs on the presentation server

ABAP

SAP delivers its own web server, which uses ABAP as Scripting Interface. Instead of learning any other language you can code your dynamic web code in an ABAP dialect though.

CGI

The Common Gateway Interface CGI is the oldest interface for dynamic web pages. It is actually a gateway that stores the interface definitions to program or component calls. A CGI interface extracts program name and parameter from an HTTP request and calls a matching compiled program on the web server.

1.2 CGI

The Common Gateway Interface specifies the actions that can be requested by a browser from a webserver. It acts as a gatekeeper between the browser and the webserver and filters those messages that can or should be handled by the server. The CGI allows only those programs and methods to be invoked by the browser, which have been defined to the CGI before.

When a program is registered to the CGI, the CGI intercepts all CGI requests and invokes the predefined method, which is associated with the received HTTP request for the requested object. For new applications we have no interest in the accurate CGI specification. We mention them because all other dynamic webserver technologies define interfaces for program calls, which are compliant to the older CGI specification.

1.3 Active Server pages (ASP and ASP.NET)

Active Server Pages are Microsoft's implementation of dynamic web pages.

Active Server Pages allow a web developer to insert programming statements into HTML pages. When the Internet Information Server encounters the programming statements, it hands them over to a script processor. The IIS comes ready with support for the Microsoft Windows Scripting Host, that understands Visual Basic Script and Server Side JScript, a Microsoft dialect of the Netscape Javascript proposal. Microsoft allows to plug in other scripting processors, so theoretically the script language is arbitrary, unless an appropriate plug-in is found on the server. When writing this book, the only third-

Scripting is done via DCOM

The scripting processor is called via a well-defined DCOM interface by the Internet Information Server.

The example below uses a simple server side JavaScript script. The code between <%>..<%> is executed when the file is retrieved and the result of the code execution is inserted instead.

Listing 1: Visual Basic Server ASP SCRIPT example telling you dynamically the web server version number

```
<html><body>
Server Side ASP Visual Basic Script
<BR>Message:
<% = "<b>Hello from <i>IIS Server</i> version "
+ server.httpdjsVersion) % +"</b>">
<BR>End of Message </body></html>
```

⇨

Listing 2: Following pages is sent back to the browser

```
<html><body>
Server Side ASP Visual Basic Script
<BR>Message:
<b>Hello from <i>IIS Server</i> version 2.0</b>
<BR>End of Message </body></html>
```

⇨

Listing 3: The browser then displays something similar to here

```
Server Side ASP Visual Basic Script
Message: Hello from IIS Server Version=2.0
End of Message
```

⇨

1.4 Java Server Pages, JSP

Java Server Pages are special Java classes that return valid HTML code

JSP are the Java corespondents to the Microsoft ASP and the SAP R/3 BSP. If the web server supports the JSP it calls the Java Virtual Machine (Java Runtime) and executes the appropriate method, which is expected to return a valid HTML code string.

1.5 Business Server Pages (BSP)

Business server pages are similar to ASP and JSP, however they support as scripting language ABAP. This is cool for ABAP developers coming out of the R/3 world,, who want to implement quick code. The bad thing is, that the Business Server Pages are currently supported by the SAP R/3 Web Application Server.only.

1.6 Server Components

Components are library functions

In object oriented programming the usual term for a program library is component. This is collection of subroutines or functions designed in a way that

they can be called by an external program.

A server component is a piece of software that can be called from an external source. Practically every program library that is called from a client application is a server in this sense and indeed most servers are nothing than compiled library routines, which are called by a client. However, for the sake of object oriented programming you should avoid thinking in terms of libraries. It widens your horizon, if you think of a server as service provided by a program whose implementation details are hidden you're your client program's eyes and that the service is solicited by the underlying transaction manager or the operating system respectively.

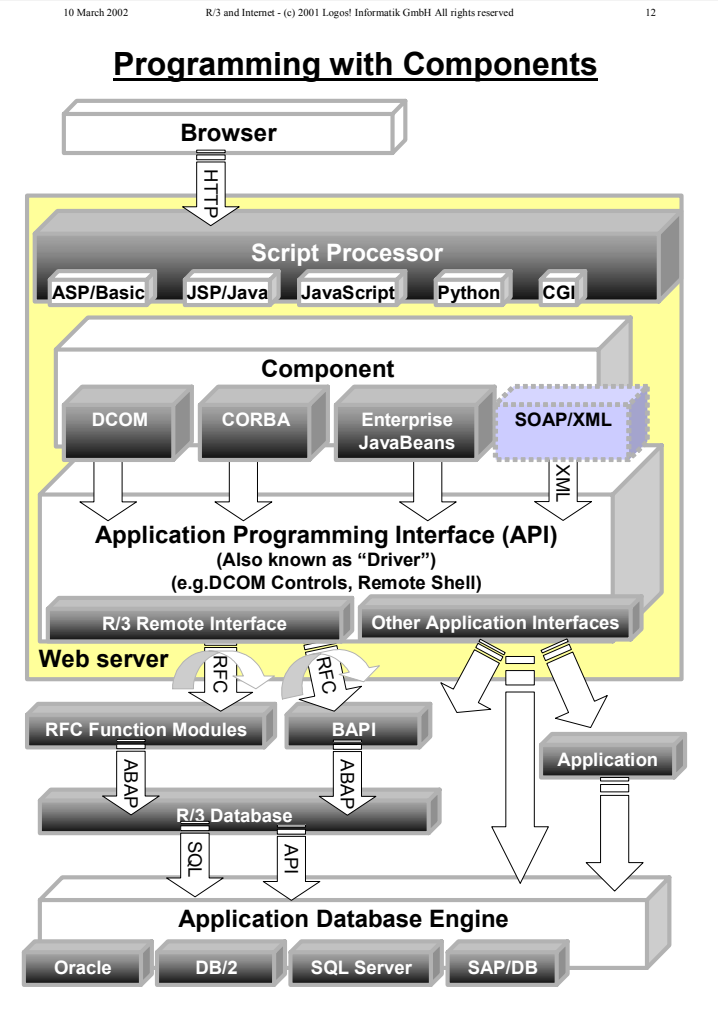
Components are library functions

Using the Microsoft IIS, a component (a "DLL") can be invoked directly from the browser by specifying it in the URI.

```
http.//www.logosworld.com/CGI/testiis.dll?method="test"&myname="Goo
fy"
```

Web server based on Java usually do not allow direct communication between a browser and a component. They can only be called from Servlets, which in turn can invoke the component.

Figure 1: Browser calls a script, script calls a component method



1.7 Java Servlets

Java Servlets are programs that expose methods that can be invoked by the web server on a browser request. They run as demons on the web server and listen to requests from the web server. Whenever the HTTP server gets a request for the servlet, it invokes a method that corresponds to the type of HTTP request, mostly the request types are Get or Post.

Servlets must support methods doGet and doPost

Java server pages are ordinary Java classes. The only requirement is, that they have to support a set of special methods:

```
doGet()doGet
doPost()
doXxx()
```

These methods are the interface between the webserver and the class, which the browser requests. These methods actually reflect the gateway function that are permitted by the CGI specification.

doGet

The doGet method is automatically invoked if the calling web server submits an HTTP Get request for the class.

doPost

The doPost method is automatically invoked if the calling web server submits an HTTP Post request for the class.

doXxx

There may be other interface methods implemented. They are called doXxx where the Xxx is to be replaced by the appropriate HTTP request. If you want to implement an interface for the HTTP Delete statement, you would implement a method doDelete, for the HTTP Head command you'd implement doHead.

1.8 Demon

A demon is program that stays permanently in memory and listens to broadcast events.

When an event occurs the demon executes some appropriate action. In an TCP/IP environment a demon typically listens to the traffic to a single port. A famous demon is the LPD demon in UNIX, which serves as a printer manager. R/3 people are familiar with the LPD variation SAPLPD, which does the same as LPD but also understands the R/3 printer language and translates it into operating system printer API calls. Other names for a demon are: listener, event handler or data sink.

1.9 EJB - Enterprise Java Beans

1.10 JAVA, COM and ABAP

Hiatus.

Hiatus

Hiatus

2 Web Pages With Active Server Pages

ASP

2.1 ASP Hello World

Active Server Pages, or ASP, add support for Visual Basic to HTML pages. ASP is the programming model for the Microsoft Internet Information Server IIS. ASP enables developers to build rich and sophisticated web applications quickly by combining programming logic (written in the script language of their choice) with HTML pages.

ASP definition by the Microsoft development support

120

This description is taken from Microsoft development support:

ASP pages are typically used to provide security for a Web site, access databases, and call business logic encapsulated in COM objects. The ASP engine on the server compiles and runs the script. The browser receives only the resulting HTML pages that are constructed from the server-side script. ASP script can be programmed in either Jscript™ or VBScript.

ASP are HTML pages with added VB statements

In practice Active Server Pages are HTML pages which are enhanced by inserting Visual Basic or JavaScript commands. The program statements produce some output which is inserted in place of the program code into the HTML document.

Here it is, the first scriptlet in ASP and naturally it is called "HELLO WORLD".

130

The following program does not need much comment. It responds with a 'hello world' to the calling browser.

You should have a similar script ready in your base directory all the time. It will allow you to test the IIS, in case you have doubts that it is active. Unfortunately the response from IIS in case of malfunction is pretty uninformative. It will simply say "Page not found".

Listing 4: Hello World ASP SCRIPT

```
<html>
<head>
<title>Hello World ASP SCRIPT</title>
</head>
<body>
Server Side ASP Visual Basic Script
<p>
Now there should be a message from the Server<BR>Message:
<%
= "<B><U>Hello from ASP Server</U></B>"
%>
<BR>The message should be one line above</p>
</body>
</html>
```

Hello World with an ASP snippet

The script is a mixture of HTML and Visual Basic. The text body is a standard HTML document apart from the section between `<%` and `%>`. There you will find the Visual Basic program code. In our example it is a simple response to the browser. The command is the equal-sign (=) followed by a string.

2.2 Executing VB Script

Visual Basic Script is the inherent language of ASP. The ASP processor actually executes the Windows Scripting Host which is part of the Windows operating system to process VB Script statements.

VBS can execute any COM component

The limits of ASP are the limits of the Scripting Host. This means virtually no limit at all, as you can call any compiled executable or registered COM object from VBS.

FileSystemObject

At the core of the example is the Visual Basic FileSystemObject. The

FileSystemObject is a registered Windows library. It combines nearly all methods and properties which are applicable to files stored on a drive like name, file size, file date, open a new or existing file, replace the content, rename it and many more.

This script displays the names of the files found, one per row.

The example below retrieves a handle for the requested folder. The handle has a container object which holds all the folders of the sub directory in it. Looping over the container will give you one file after the other.

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set subdir = fs.GetFolder(".")
ii = 0
response.write subdir.path
For Each xfile In subdir.Files
ii = ii + 1
response.write "<P>"
response.write ii
response.write ": " + xfile.Name + "</P>"
Next
%>
```

2.3 Sending HTML Formatting

Active Server Pages replace part of the HTML page with program code, whose output is inserted in the right places.

When a browser communicates with a server it receives a text stream as a result. When the requested web page contains ASP commands, they will be processed, interpreted and then stripped by the web server. The browser will see only the resulting HTML text stream.

ASP are HTML templates with inserted program statements.

An Active Server is, generally speaking, a mixture of static **HTML tags**, **text** and a **series of program statements** which eventually produce some parts of an HTML page. They are pre-formatted HTML templates, where significant parts are replaced by program code. When the program code is executed, its output is sent to the requesting browser.

To demonstrate this we show an ASP sniplet which writes a table to the browser. For that purpose we have a static table header and a dynamic loop to send five lines of an HTML table.

Listing 5: ASP creating an HTML table

```
<body>
Server Side ASP
Sending an HTML table
<table border="1"
cellpadding="0" cellspacing="0"
width="100%">
<%
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
response.write "<TR><TD>A table line</TD></TR>"
%>
</table>
</body>
```

2.4 Capturing Commandline Parameters

Like any other proficient programming language, an ASP can be called along with the specification of parameters. These are called Querystring in Microsoft terminology. There are two versions: POST will read the values from an HTML form while GET will read the values from the URL.

A querystring is the additional text string specified with the URL in the browser request line A querystring can look Querystring must follow the URL separated by a question mark "?"

When you specify an URL in the browser you can pass a parameter string right along. This string can then be evaluated by the ASP program to control the program flow.

```
http://localhost/querystring.asp?Name="Micky"&City="Duckburg"
```

The querystring is specified immediately after the URL and separated from it by a question mark. There are no spaces allowed between the URL and the querystring because the browser does not allow spaces. A browser actually sees the URL plus the querystring as a single URL. Multiple Querystrings are joined by an ampersand "&".

Multiple querystrings are joined by an ampersand "&"

The ASP parser is intelligent enough to interpret the querystring, so you do not have to do the tedious job of splitting the string into its elements. The ASP Request.QueryString method treats the querystring according to the Visual Basic syntax. This allows positional and named parameters.

ASP Page to display the querystring parameters if specified

```
<HTML><BODY>
<H3>Here are the parameters which have been passed.</H3>
<P>
<% = "Querystring="+Request.QueryString %></P>
<P><% = "Input field Name is="+Request.QueryString("NAME") %></P>
<P><% = "Input field City is="+Request.QueryString("CITY") %></P>
</BODY></HTML>
```

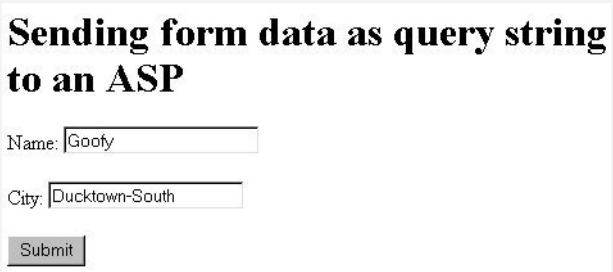
An HTML FORM can post data in the request body or send them along with the URL

An HTML form element allows us to use two completely different methods to transfer the data to the web server. The GET method takes the form of input fields and values and appends them to the end of the submitted URL in the CGI convention as already shown above.

ASP Page to display the form values with the request.Form method

```
<HTML><BODY>
<%
response.write("<H3>Form Content</H3>")
for each formfield in Request.Form
response.write(formfield&"="&Request.Form(formfield)&" ")
next
%>
</BODY></HTML>
```

Figure 2: HTML page produced by above FORM element



Submitting a FORM with METHOD=GET does generate a querystring with the form data

```
<FORM METHOD="GET" ACTION="querystring.asp">
                        <H1>Sending form data as query string to an
ASP</H1>
                        <P>Name: <INPUT TYPE="TEXT" NAME="NAME"></P>
                        <P>City: <INPUT TYPE="TEXT" NAME="CITY"></P>
      <P><INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit"></P>
</FORM>
```

Generated URL when the FORM is submitted with method GET

```
http.//querystring.asp?NAME="Micky"&CITY="Duckburg"
```

ASP Page to display the querystring parameters if specified with the request.Querystring method

```
<HTML><BODY>
<H3>Here are the parameters which have been passed.</H3>
<P>
<% = "Querystring="+Request.QueryString %></P>
<P><% = "Input field Name is="+Request.QueryString("NAME") %></P>
<P><% = "Input field City is="+Request.QueryString("CITY") %></P>
<P>You can address the parameters with a numeric position index as
well</P>
<P>
<% if Request.QueryString <> "" then _
resstr = "FirstParam is="+Request.QueryString(1)" %>
<% response.write resstr %>
</P>
</BODY></HTML>
```

POST sends form input hidden with the request

You may have decided that it is not desirable to send the form values visible as part of the URL string. In that case you can use the POST method and the data will be embedded in the body part of the request. The data is then stored with the request object and can be accessed with the request.Form method.

Submitting a FORM with METHOD=POST does generate a querystring with the form data

```
<FORM METHOD="POST" ACTION="querystring.asp">
                        <H1>Sending form data as query string to an
ASP</H1>
                        <P>Name: <INPUT TYPE="TEXT" NAME="NAME"></P>
                        <P>City: <INPUT TYPE="TEXT" NAME="CITY"></P>
      <P><INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Submit"></P>
</FORM>
```

ASP Page to display the form values with the request.Form method

```
<HTML><BODY>
<%
response.write("<H3>Form Content</H3>")
for each formfield in Request.Form
response.write(formfield&"="&Request.Form(formfield)&" ")
next
%>
</BODY></HTML>
```

2.5 Global.asa – The ASP Autoexec File

When an application or a session is called for the first time, the IIS automatically executes a specifically named event handler. These event handlers must be stored in a file named global.asa and in the root of the specific subweb.

Figure 3: A typical GLOBAL.ASA contains implementations for the following events

```
<script language=vbscript runat=server>
SUB Application_OnStart
END SUB

SUB Application_OnEnd
END SUB

SUB Session_OnStart
END SUB

Sub Session_OnEnd
END SUB
</script>
```

When an application is requested for the first time the IIS executes Application_OnStart() in global.asa

When the IIS detects that one of its hosted applications is requested for the first time it signals the event `Application_OnStart`. If this event is fired, the IIS looks for a file named `global.asa` in the root of the current subweb. If the `global.asa` contains a handler routine with the same name (`Sub Application_OnStart()`) it will execute it. Accordingly there are other events, whose corresponding procedure names are `Application_OnStart`, `Application_OnEnd`, `Session_OnStart`, or `Session_OnEnd`.

IIS looks for global.asa in the root of the current subweb

You can have multiple `global.asa` files on your site. IIS searches for the `global.asa` in the root of the current subweb only. Subwebs are marked specially if you edit them with FrontPage. Outside of FrontPage you can tell the root of a subweb from the presence of a system folder with the name `_vti_pvt`. This subdirectory is a system file and may be hidden, so be sure to display hidden files if you look for it with Explorer.

If global.asa contains an error the first user to call the application receives the error message

If the `global.asa` file contains an error the first user to call a page from the respective subweb will receive an error message. E.g. it may look like the following, where we tried to execute a standard .asp script, which is not allowed (code must be enclosed in `<SCRIPT>… </SCRIPT>` tags instead).

Figure 4: Sample error message when global.asa contains an error, in this case invalid ASP script tags have been found

Script blocks must be one of the allowed Global.asa procedures. Script directives within <% ... %> are not allowed within the global.asa file. The procedure names allowed are Application_OnStart, Application_OnEnd, Session_OnStart, or Session_OnEnd.

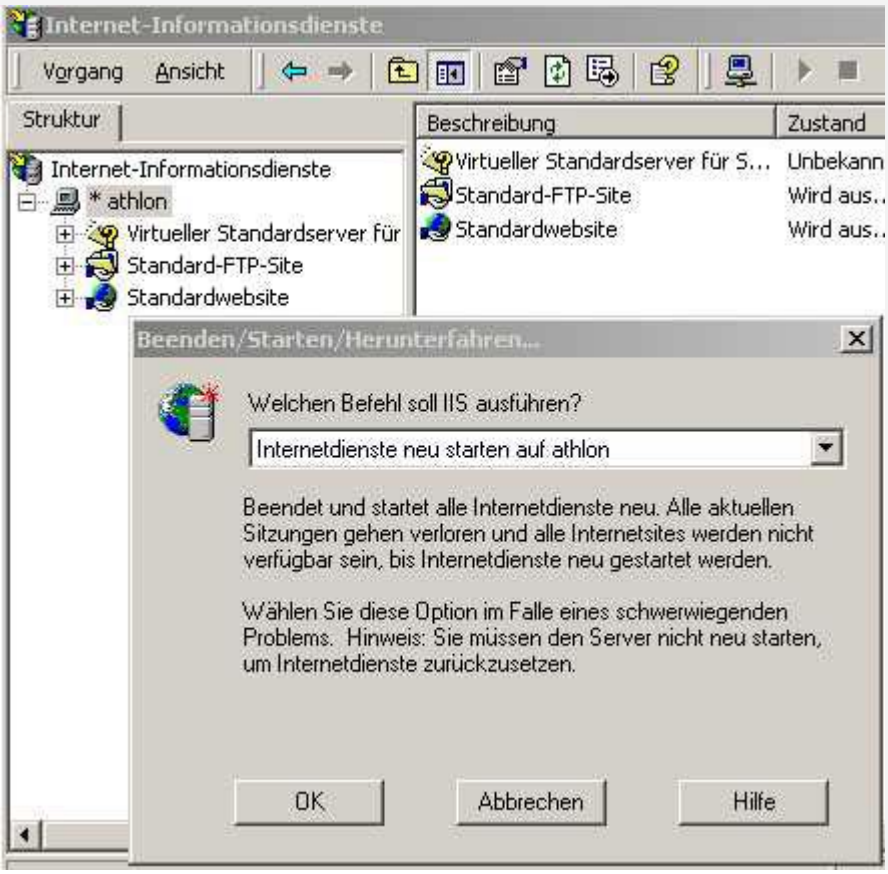
⇨

Because you usually do not want a user to get a system error message it is wise to add an `ON ERROR GOTO` statement to the script in `global.asa` which provides proper error handling. The most simple but also worst handler would be `ON ERROR RESUME`.

If you changed global.asa you must stop and restart the web server

If you added or changed an `Application_OnStart` or `Application_OnEnd` routine in the global.asa file, they will only be executed after the web server has been reset or restarted. Resetting the web server is done by stopping and restarting it. You can restart the IIS from the administration menu. In NT 5 and Windows 2000 you can access the administration menu from the system control panel from where you select Internet Service Manager to start the IIS administration manager and there you have a menu option to restart the IIS.

Table 1: You can restart IIS from IIS Internet service manager which is found in the NT control panel



2.6 ASP Application Variables

Using a globally stored application variable

Counter application

225

This example implements a counter which is incremented every time the page is viewed. You can see that the variable is stored globally from the fact that its value is actually incremented by every user and window you call. A session variable would reset the counter to its initial value every time a new session, i.e. a new browser window is opened.

Listing 6: Sample code to set a global application variable

```
<body>
<H3>Here is the value of the counter.</H3>
<P>
<% Application.Lock %>
<% Counter = Application("AccessCounter") %>
<% Counter = Counter + 1 %>
<% Application("AccessCounter") = Counter %>
<% Application.Unlock %>
<% = "<P>The value of the counter is " %>
<% = Counter %>
<% = "</P>" %>
<% = "<P>Connection String" %>
<% = Application("ConnectionString") %>
<% = "</P>" %>
<%  %>
<% Application.UnLock %>
</P></body>
</html>
```

⇨

Open application in two browser windows to test it

In order to test the effect you should open this ASP page in two browser windows. Every time you press the refresh button of the browser the counter will be implemented by one (1).

Figure 5: Sample output of counter application in browser

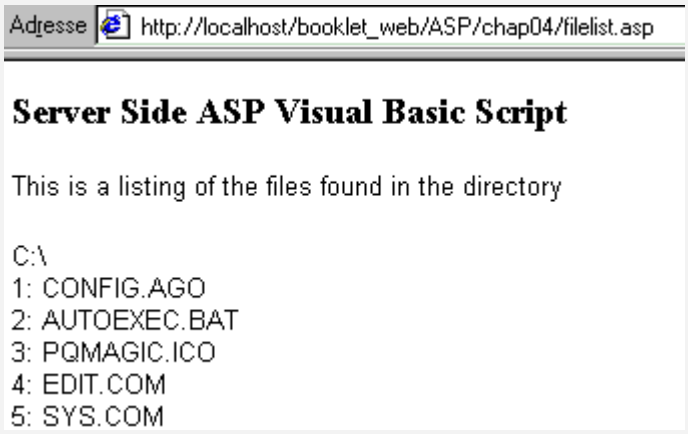
2.7 Example: Display Server File List With ASP

This example will show the names of the files found in a directory of the server. It is a basic ASP examples which displays information found on the ASP server to the requesting browser.

At the core of the example is the Visual basic FileSystemObject. The FileSystemObject is a registered Windows library. It combines nearly all methods and properties which are applicable to files stored on a drive like name, file size, file date, open a new or existing file, replace the content, rename it and many more.

The example below retrieves a handle for the requested folder. The handle has a container object which holds all the folders of the sub directory in it. Looping over the container will give you one file after the other.

Figure 6: Listing files of directory on the ASP server



Listing 7: The script displays the names of the found directories

```
<%
Set fs = CreateObject("Scripting.FileSystemObject")
Set subdir = fs.GetFolder(".")
ii = 0
response.write subdir.path
For Each xfile In subdir.Files
ii = ii + 1
response.write "<P>"
response.write ii
response.write ": " + xfile.Name + "</P>"
Next
%>
```

⇨

3 Web Pages With Java Server Pages and Servlets

Choosing the right programming language is one of the crucial decisions for the projects success. The purpose of the programming language is to create the links between the already existing elements of your infrastructure. The programming language must be able to provide code that is supported by the web server, it must be able to connect to R/3 and any other data source needed for your application. But the language must also provide self-documenting, easy-to-read and easy-to-debug code. Some suitable languages are: Java, Visual Basic or Delphi.

3.1 Java

Java is a C++ like programming language, developed especially to create distributed applications via a TCP/IP network.

Java is known as one of the classics object oriented programming. It has been developed by the computer giant SUN® Microsystems. Other than C++, Java is copyrighted, so that SUN® Microsystems retains the full control over the evolution of the language and its dialects and subsets and over the implementation of any run-time engines. Practically this means, that there is only one Java language. However, this is also true for other proprietary languages like Borland Delphi™ and Microsoft Visual Basic™.

Java is a semi compiled language

The Java language according to the SUN specification is a pseudo-compiler language. Code written in Java must be compiled into a transportable meta-code which is eventually interpreted by the Java run-time engine. The Java run-time engine is usually called the *Java Virtual Machine*.(JVM).

Java Applets and Java Servlets

Before we show a small Hello World example, let as speak about *Java Applets*. Java is mostly encountered in the form of some fancy, usually graphical gimmicks on some web sites. This is Java compiled code, which is executed on the presentation server by a web browser that supports the Java Virtual Machine. But Java is much more: it is a full featured programming language, that can be used everywhere in distributed object oriented programming.

Java HelloWorld

Other than in Visual Basic or many other programming languages, Java is designed in a way, that a public class is always stored in a single file. Therefore we create a file called HelloWorldClass.java that contains the code for the class HelloWorldClass . To make the class executable, i.e. make it a main thread, it requires a default method with the name main .

Figure 7: Java class HelloWorld

```
public class HelloWorldClass {
  public static void main(String[] args) {
    System.out.println("Welcome to Java hello World");
  }
}
```

⇨

This example prints the string in parentheses when the method main is invoked. The term “to invoke” is commonly used for calling a method. To gve you an understanding, here are some comments on the syntax.

```
public class HelloWorldClass { some code }
```

The keyword class begins a new class and the code of the class is embraced in curly brackets.

The keyword public tells the compiler, that the following object shall be visible for external objects.

```
public static void main(String[] args) {
```

Our class defines one single method with the name main. A method with the name

main is recognised as the default method. This method is called when the JVM instantiated the class and the caller did not specify a different default method.

The keyword void tells us that the method will not return any result to the caller. In Visual Basic or Delphi terms, a method with the keyword void is a procedure otherwise it is a function.

After the method name there you find a list of arguments in parentheses. In the example there is only one argument with the name args defined. The cryptic String[] tells us that the parameters are an array of strings. String defines the argument as a string type and the square brackets [] defines an Array of String. Because arrays have a dynamic length in Java, this allows the specification of an arbitrary number of arguments. The number of arguments can be questioned with length property of the array (args.length).

Before the class can be instantiated it needs to be compiled using the Java compiler `javac`.

Figure 8: Compiling the HelloWorldClass with the java command line utility

```
java –verbose HelloWorldClass
[parsing started HelloWorldClass.java]
[parsing completed 130ms]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/Object.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/String.class)]
[checking HelloWorldClass]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/System.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/PrintStream.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/FilterOutputStream.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/OutputStream.class)]
[wrote HelloWorldClass.class]
[total 561ms]
```

⇨

Figure 9: Files created by the Java compiler

```
07.10.2001  19:33                451 HelloWorldClass.class
07.10.2001  19:31                302 HelloWorldClass.java
               2 File(s)           753 Bytes
```

When the class is created it can be tested with the Java run-time utility. This utility

- creates an instance of the class and
- executes the method main

Figure 10: Executing the class with the Java Run-time

```
D:\JDK> java HelloWorldClass
```

Welcome to Java hello World

⇨

3.2 Java Utilities

The Java Software Development Kit JDK can be downloaded free of charge from the SUN web sites. Nearly all Java packages are nothing than development environments, which actually use the JDK utilities to generate and execute the code.

javac

The javac utility is the java compiler. It takes a Java class or Java package (a collection of classes bound together in a single file) and compiles it into classes. The input file for the javac utility would usually have the extension .java and generates and output with the extension .class.

Example:

```
java HelloWorldClass.java
compiles into
java HelloWorldClass.class
```

Figure 11: Call options of the javac utility

```
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -O                         Optimize; may hinder debugging or enlarge class file

  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -d <directory>             Specify where to place generated class files
  -encoding <encoding>       Specify character encoding used by source files
  -target <release>          Generate class files for specific VM version
```

⇨

java

The java utility is an implementation of the Java Virtual Machine. It creates an instance of a class and executes the main method of the class.

Figure 12: Executing the class with the Java Run-time

```
D:\JDK> java HelloWorldClass
```

Welcome to Java hello World

Figure 13: Call options of the java utility

```
Usage: java [-options] class [args...]
           (to execute a class)
   or  java -jar [-options] jarfile [args...]
           (to execute a jar file)

where options include:
    -cp -classpath <directories and zip/jar files separated by ;>
                  set search path for application classes and resources
    -D<name>=<value>
                  set a system property
    -verbose[:class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -showversion  print product version and continue
    -? -help      print this help message
    -X            print help on non-standard options
```

⇨

javap

The javap utility analyses compiled java classes and returns many helpful information.

Figure 14: Executing the javap analyser on the Hello World Class

```
D:\JDK> javap HelloWorldClass
```

Compiled from HelloWorldClass.java

public class HelloWorldClass extends java.lang.Object {

  public HelloWorldClass();

```
   public static void main(java.lang.String[]);
}
```

⇨

Figure 15: Call options of the javap utility

```
Usage: javap <options> <classes>...

where options include:
   -b                        Backward compatibility with javap in JDK 1.1
   -c                        Disassemble the code
   -classpath <pathlist>     Specify where to find user class files
   -extdirs <dirs>           Override location of installed extensions
   -help                     Print this usage message
   -J<flag>                  Pass <flag> directly to the runtime system
   -l                        Print line number and local variable tables
   -public                   Show only public classes and members
   -protected                Show protected/public classes and members
   -package                  Show package/protected/public classes and members
   -private                  Show all classes and members
   -s                        Print internal type signatures
   -bootclasspath <pathlist> Override location of class files loaded
                             by the bootstrap class loader
   -verbose                  Print stack size,number of locals and args for methods
                             If verifying, print reasons for failure
```

⇨

3.3 Java

Java is a C++ like programming language, developed especially to create distributed applications via a TCP/IP network.

Java is know as one of the classics object oriented programming. It has been developed by the computer giant SUN® Microsystems. Other than C++, Java is copyrighted, so that SUN® Microsystems retains the full control over the evolution of the language and its dialects and subsets and over the implementation of any run-time engines. Practically this means, that there is only one Java language. However, this is also true for other proprietary languages like Borland Delphi™ and Microsoft Visual Basic™.

Java is a semi compiled language

The Java language according to the SUN specification is a pseudo-compiler language. Code written in Java must be compiled into a transportable meta-code which is eventually interpreted by the Java run-time engine. The Java run-time engine is usually called the *Java Virtual Machine*.(JVM).

Java is a semi compiled language

Before we show a small Hello World example, let as speak about *Java Applets*. Java is mostly encountered in the form of some fancy, usually graphical gimmicks on some web sites. This is Java compiled code, which is executed on the presentation server by a web browser that supports the Java Virtual Machine. But Java is much more: it is a full featured programming language, that can be used everywhere in distributed object oriented programming.

Java HelloWorld

Other than in Visual Basic or many other programming languages, Java is designed in a way, that a public class is always stored in a single file. Therefore we create a file called `HelloWorldClass.java` that contains the code for the class `HelloWorldClass` . To make the class executable, i.e. make it a main thread, it requires a default method with the name main .

Figure 16: Java class HelloWorld

```
public class HelloWorldClass {
  public static void main(String[] args) {
    System.out.println("Welcome to Java hello World");
  }
}
```

⇨

This example prints the string in parentheses when the method main is invoked. The term "to invoke" is commonly used for calling a method. To gve you an understanding, here are some comments on the syntax.

```
public class HelloWorldClass { some code }
```

The keyword class begins a new class and the code of the class is embraced in curly brackets.

The keyword public tells the compiler, that the following object shall be visible for external objects.

```
public static void main(String[] args) {
```

Our class defines one single method with the name main. A method with the name main is recognised as the default method. This method is called when the JVM instantiated the class and the caller did not specify a different default method.

The keyword void tells us that the method will not return any result to the caller. In Visual Basic or Delphi terms, a method with the keyword void is a procedure otherwise it is a function.

After the method name there you find a list of arguments in parentheses. In the example there is only one argument with the name args defined. The cryptic String[] tells us that the parameters are an array of strings. String defines the argument as a string type and the square brackets [] defines an Array of String. Because arrays have a dynamic length in Java, this allows the specification of an arbitrary number of arguments. The number of arguments can be questioned with length property of the array (args.length).

Before the class can be instantiated it needs to be compiled using the Java compiler `javac`.

Figure 17: Compiling the HelloWorldClass with the java command line utility

```
java –verbose HelloWorldClass
[parsing started HelloWorldClass.java]
[parsing completed 130ms]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/Object.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/String.class)]
[checking HelloWorldClass]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/System.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/PrintStream.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/FilterOutputStream.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/OutputStream.class)]
[wrote HelloWorldClass.class]
[total 561ms]
```

⇨

Figure 18: Files created by the Java compiler

```
07.10.2001  19:33               451 HelloWorldClass.class
07.10.2001  19:31               302 HelloWorldClass.java
               2 File(s)          753 Bytes
```

When the class is created it can be tested with the Java run-time utility. This utility

- creates an instance of the class and

- executes the method main

Figure 19: Executing the class with the Java Run-time

```
java HelloWorldClass
```

```
Welcome to Java hello World
```

3.4 Java Utilities

The Java Software Development Kit JDK can be downloaded free of charge from the SUN web sites. Nearly all Java packages are nothing than development environments, which actually use the JDK utilities to generate and execute the code.

javac

The javac utility is the java compiler. It takes a Java class or Java package (a collection of classes bound together in a single file) and compiles it into classes. The input file for the javac utility would usually have the extension .java and generates and output with the extension .class.

Example:

```
java HelloWorldClass.java
compiles into
java HelloWorldClass.class
```

Figure 20: Call options of the javac utility

```
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -O                         Optimize; may hinder debugging or enlarge class file

  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -d <directory>             Specify where to place generated class files
  -encoding <encoding>       Specify character encoding used by source files
  -target <release>          Generate class files for specific VM version
```

⇨

java

The java utility is an implementation of the Java Virtual Machine. It creates an instance of a class and executes the main method of the class.

Figure 21: Executing the class with the Java Run-time

```
java HelloWorldClass
```

```
Welcome to Java hello World
```

Figure 22: Call options of the java utility

```
Usage: java [-options] class [args...]
           (to execute a class)
   or  java -jar [-options] jarfile [args...]
           (to execute a jar file)

where options include:
    -cp -classpath <directories and zip/jar files separated by ;>
                  set search path for application classes and resources
    -D<name>=<value>
                  set a system property
    -verbose[:class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -showversion  print product version and continue
    -? -help      print this help message
    -X            print help on non-standard options
```

⇨

javapp

The javap utility analyses compiled java classes and returns many helpful information.

Figure 23: Executing the javap analyser on the Hello World Class

```
javap HelloWorldClass
```

```
Compiled from HelloWorldClass.java
public class HelloWorldClass extends java.lang.Object {
    public HelloWorldClass();
    public static void main(java.lang.String[]);
}
```

Figure 24: Call options of the javap utility

```
Usage: javap <options> <classes>...

where options include:
   -b                        Backward compatibility with javap in JDK 1.1
   -c                        Disassemble the code
   -classpath <pathlist>     Specify where to find user class files
   -extdirs <dirs>           Override location of installed extensions
   -help                     Print this usage message
   -J<flag>                  Pass <flag> directly to the runtime system
   -l                        Print line number and local variable tables
   -public                   Show only public classes and members
   -protected                Show protected/public classes and members
   -package                  Show package/protected/public classes and members
   -private                  Show all classes and members
   -s                        Print internal type signatures
   -bootclasspath <pathlist> Override location of class files loaded
                             by the bootstrap class loader
   -verbose                  Print stack size,number of locals and args for methods
                             If verifying, print reasons for failure
```

⇨

4 Web Pages With XSL Stylesheets and XML

XSL – the eXtended Stylesheet Language – had been originally designed to define templates and layouts to display data of a specific XML file. XSL is in fact a template-based programming language whose programs are completely implemented in XML with the purpose to receive XML documents as input and transform them into a target document, e.g. in HTML or PDF.

4.1 XSL Is Simple

I have seen dozens of books on XML and XSL on the market, propably there are a several hundred out there. They all have in common that their authors have a splendid knowledge on how to use XML and exploit their tremendous and most fancy capabilities. However the enormous possibilities behind XML makes them blinded for the simplicity of XML and the main purpose of XSL: the transformation of a well-formed XML document into an HTML output. I therefore want to start with a tutorial, that does not show the capabilities of XSL but asks, how to transform a database output from XML into a good-looking HTML document.

4.2 Our Goal: An Inventory Table of Animals

Let us assume that we are responsible for a little animal farm and want to create a little grid table with the names and some data of the animals in our farm. Here is the output that I imagine.

Figure 25: Listing of animals in our little farm

Name	Family	Gender	Weight
Elsa	Cow	female	420
Rosa	Pig	male	120
Lisa	Chicken	male	3

The HTML that generated the table listing would look similar to the following.

Figure 26: Corresponding HTML of above

```
<html>
<body>
<table border="1">
<tr><th>Name</th><th>Family</th><th>Gender</th><th>Weight</th></tr>

<tr><td>Elsa</td><td>Cow</td><td>female</td><td>420</td></tr>
<tr><td>Rosa</td><td>Pig</td><td>male</td><td>120</td></tr>
<tr><td>Lisa</td><td>Chicken</td><td>male</td><td>3</td></tr>

</table>
</body>
</html>
```

⇨

As you see, there are basically two blocks:

A static block comprising the HTML wrapper with the <HTML> and <BODY> tags, the table frame with the <TABLE> tag and the table's headline.

A series of lines with the animal data with the same structure for every animal in the farm.

4.3 Building The XML Farm

Before we put animals in our farm, we create an empty farm, i.e. a minimal XML document that builds the envelope for our farm data:

Figure 27: The empty XML farm

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm>
</Farm>
```

The first line embraced in the <?..?> tags represents a directive to the parser. The purpose of the directives are to tell the parser how the XML data shall be treated. The shown directive tells some information about the version of XML used and what ASCII character encoding is in use.

The <Farm> … </Farm> pair is the top envelope also known as root element of the XML. Every well-formed XML file must have one and only one root element.

Now we are gong to populate the farm with some animal. Every animal gets a name assigned as an attribute and a number of sub node elements telling us something about the animal.

Figure 28: The XML farm populated with an animal

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="U:\examples\XML\Farm.xsl"?>
<Farm>
        <Animal name="Elsa">
                <Family>Cow</Family>
                <Weight>420</Weight>
                <Gender>F</Gender>
        </Animal>
</Farm>
```

⇨

We already attached the reference to an (existing) style sheet file, which will be used to format the presentation of the XML data.

The final farm then looks as follows.

Figure 29: The XML farm populated with animals and its anatomy

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="U:\examples\XML\Farm.xsl"?>
```

Reference to an existing XSL stylesheet "Farm.xsl"

```
<Farm>
        <Animal name="Elsa">
```

One <Animal> tag for every animal

```
                <Family>Cow</Family>
                <Weight>420</Weight>
                <Gender>F</Gender>
```

One tag for each characteristic of the animal

```
        </Animal>
        <Animal name="Rosa">
                <Family>Pig</Family>
                <Weight>120</Weight>
                <Gender>M</Gender>
        </Animal>
        <Animal name="Lisa">
                <Family>Chicken</Family>
                <Weight>3</Weight>
                <Gender>M</Gender>
        </Animal>
</Farm>
```

⇨

4.4 XSL – Step-by-step

A basic stylesheet is formed by a root element called <xsl:stylesheet>. In between there are the XSL stylesheet directives in XML notation.

Figure 30: A minimum XSL stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
</xsl:stylesheet>
```

⇨

This stylesheet does actually nothing to the XML output. If applied, it will pass-through the input XML data. For a proper stylesheet use you need first to add a reference to a namespace. An XSL namespace is a definition file that tells the rules and ranges of valid XML tags of the current XSL stylesheet. Here is an elementary XSL stylesheet with a reference to a namespace.

Figure 31: An elementary XSL stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
</xsl:stylesheet>
```

⇨

The used reference xmlns:xsl=http://www.w3.org/TR/WD-xsl is actually a dummy reference. This means that the specified URL is not really looked up in the internet, but that the parser would know the content of the referenced file already. There exists a variety of such dummy references. However, if the specified namespace URL is not among the default ones, it must exist and the parser must be able to open it, otherwise the parse will fail.

4.5 XSL Templates

The templates are the core of XSL formatting. A template defines a number of XSL definitions in its body which are applied to every matching XML tag. The parser actually processes the full XML file and replaces the found tag with the template. The match-attribute restricts the application of the template to the XML tags that match the specified pattern. Specifying match="/" catches the root element of the XML input file.

Figure 32: An elementary XSL stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
        <xsl:template match="/">
        Hiding the root
        </xsl:template>
</xsl:stylesheet>
```

⇨

The result of the stylesheet applied to any arbitrary XML file would always be the following one-liner:

Hiding the root

4.6 XSL Applied To the Farm

We now feel ready to apply our template to the farm XML data. The example XSL defines a single template (xsl:template) applied to the root (match="/"). The
445 template creates the body of an HTML document and then loops over every animal in the XML file (<xsl:for-each select="Farm/Animal">) to produce a row of an HTML table element (<tr><td>Cow</td></tr>).

Figure 33: A matching XSL stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
	<xsl:template match="/">
		<html>
			<head/>
			<body>
				<table border="1">
					<tr>
						<th>Name</th>
						<th>Family</th>
						<th>Gender</th>
						<th>Weight</th>
					</tr>
				<xsl:for-each select="Farm/Animal">
					<tr>
						<td>
							<xsl:value-of select="@Name"/>
						</td>
						<td>
							<xsl:value-of select="Family"/>
						</td>
						<td>
							<xsl:choose>
							<xsl:when
test=".[Gender='M']">male</xsl:when>
							<xsl:when
test=".[Gender='F']">female</xsl:when>

	<xsl:otherwise>unknown</xsl:otherwise>
							</xsl:choose>
						</td>
						<td>
							<xsl:value-of select="Weight"/>
						</td>
					</tr>
				</xsl:for-each>
				</table>
			</body>
		</html>
	</xsl:template>
</xsl:stylesheet>
```

⇨

The anatomy of the stylesheet.

Figure 34: Anatomy of the XSL stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
```

Reference to standard name space

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
match="/" means: start with the top root, i.e. this style sheets applies to all elements of
the input XML data set
       <xsl:template match="/">
              <html>
                     <head/>
                     <body>
                            <table border="1">
                                   <tr>
                                          <th>Name</th>
                                          <th>Family</th>
                                          <th>Gender</th>
                                          <th>Weight</th>
                                   </tr>
```

Up to here is the static block with the HTML and TABLE frame

```
---
```

Now we start a loop that processes all tags with the name "<Animal>" that are embedded in turn in a tag "<Farm>"

```
                                   <xsl:for-each select="Farm/Animal">
                                          <tr>
                                                 <td>
```

Write the value of the attribute "Name" assigned to the <Animal> element. Attributes are always referenced to by prefixing it with the at-sign (@).

```
                                                        <xsl:value-of select="@Name"/>
                                                 </td>
                                                 <td>
```

Write the value of the element which is tagged as <Family> within an <Animal> element

```
                                                        <xsl:value-of select="Family"/>
                                                 </td>
                                                 <td>
```

To make it a bit more interesting we add a "Choose" statement, which tests the value of the tag "<Gender>" against several constants and thus translates "M" into "male", "F" into "female" etc.

```
                                                        <xsl:choose>
```

If the element tagged as "<Gender>" within the current element (referred to with the dot ".") is equal to "M", then and only then write the text "male"

```
                                                        <xsl:when
test=".[Gender='M']">male</xsl:when>
```

Same for <Gender> = "F"

```
                                                        <xsl:when
test=".[Gender='F']">female</xsl:when>
```

The otherwise section catches all cases with no explicit match

```
              <xsl:otherwise>unknown</xsl:otherwise>
                                                        </xsl:choose>
                                                 </td>
                                                 <td>
                                                        <xsl:value-of select="Weight"/>
                                                 </td>
                                          </tr>
                                   </xsl:for-each>
                            </table>
                     </body>
              </html>
       </xsl:template>
</xsl:stylesheet>
```

⇨

Figure 35: Synoptic view of stylesheet and its transformation output

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
  <head/>
  <body>
  <table border="1">
    <tr>
      <th>Name</th>
      <th>Family</th>
      <th>Gender</th>
      <th>Weight</th>
    </tr>
    <xsl:for-each select="Farm/Animal">
      <tr>
      <td>
        <xsl:value-of select="Name"/>
      </td>
      <td>
        <xsl:value-of select="Family"/>
      </td>
      <td>
        <xsl:choose>
          <xsl:when test=".[Gender='M']">male</xsl:when>
          <xsl:when test=".[Gender='F']">female</xsl:when>
          <xsl:otherwise>unknown</xsl:otherwise>
        </xsl:choose>
      </td>
      <td>
        <xsl:value-of select="Weight"/>
      </td>
      </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Name	Family	Gender	Weight
Elsa	Cow	female	420
Rosa	Pig	male	120
Lisa	Chicken	male	3

```
<Animal>
        <Name>Elsa</Name>
        <Family>Cow</Family>
        <Weight>420</Weight>
        <Gender>F</Gender>
</Animal>
```

In every loop one of the <Animal> elements is focused

⇨

Parsing and Translating The XML

460 The transformation of the XML into HTML by means of the stylesheet requires a parser. There are many parsers available, most of them written in Java. On Microsoft platforms we can use the MSXML.DLL and use the transformNodeToObject or transformNode method of the XMLDOM object. Here are two versions of calling the transformation.

465

Figure 36: VBA program that transforms the XML data using the style sheet using msxml.dll

```
Dim xmlDoc As MSXML2.DOMDocument
Dim xslDoc As MSXML2.DOMDocument

Sub Main()
        Set xmlDoc = New MSXML2.DOMDocument
        Set xslDoc = New MSXML2.DOMDocument

        xmlDoc.Load ("U:\examples\XML\Farm.xml")
        xslDoc.Load ("U:\examples\XML\Farm.xsl")
        Debug.Print xmlDoc.transformNode(xslDoc)
End Sub
```

⇨

Figure 37: Active Server Page that transforms the XML data using the style sheet using msxml.dll

```
<%
Dim xmlDoc As MSXML2.DOMDocument
Dim xslDoc As MSXML2.DOMDocument
Set xmlDoc = CreateObject("Microsoft.XMLDOM")
Set xslDoc = CreateObject("Microsoft.XMLDOM")

xmlDoc.Load ("U:\examples\XML\Farm.xml")
xslDoc.Load ("U:\examples\XML\Farm.xsl")
Debug.Print xmlDoc.transformNodeToObject(xslDoc, response)
%>
```

⇨