

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa

Melksham

Wiltshire

UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

JSP - JavaServer Pages

A Java Server Page is a web page with embedded tags that call up Java classes. It can also contain Java source code, to be compiled and run when the page is accessed.

<i>Introduction</i>	<i>4</i>
<i>A simple worked example.</i>	<i>5</i>
<i>The structure of a JSP Page</i>	<i>5</i>
<i>Entering data into a form.</i>	<i>6</i>
<i>Using scripting elements.</i>	<i>11</i>
<i>A JSP that maintains state</i>	<i>15</i>

2.1 Introduction

JavaServer pages are a way of providing server side executable content in a web page. In other words, a way of providing a Web page which is varied depending on conditions on the server, information filled in to a form, etc.

The original way of providing server side executable content was through the Common Gateway Interface (CGI) and a variety of programming languages such as C, C++ and (most prevalent) Perl. Indeed, Perl and CGI are still growing though not to the same extent as some other technologies. More recently, Java servlets have been introduced and they allow you to use a similar approach to writing server side executable content – a program which produces an HTML page as its output. Java servlets are more efficient in operation than CGI programs, and for heavily used servers they provide an excellent solution. You'll probably want to choose between modPerl, servlets, and your own server written in C or C++ for such applications.

But for many server side applications, the number of changes made on a reply page are really quite small and the work involved in calculating the changes is nearly insignificant. A great shame, then, to have to write a program to spit out a huge chunk of non-varying text with just a little changing within it.

Many web servers can support "Server Side Includes" - where a page is parsed by the web server on its way from the document directory to the browser, and substitutions of certain variable are made. Using SSI, operating system commands can even be run and their outputs written in to the page sent to the browser - such a web page looks different if you examine the source on the server's discs and if you ask your browser to "view source".

Active Server Pages (ASP) from Microsoft takes a similar approach to SSI. You write Web pages which include chunks of one or more of VBScript, PerlScript and JavaScript, and the page is parsed and the script run as the server feeds the page through to the browser. The facilities provided are much more extensive than SSI, but with the "interpret every time" approach efficient of operation is not a strong point of this scheme – even the Microsoft documentation warns you of the fact!

The SSI/ASP approach is a good one, but there's a requirement for something that works along the same lines as part as the provider is concerned:

"A page of HTML that changes is not a program"

but doesn't have the same run time resource inefficiencies. Of course, to make it portable a language like Java would be nice, especially if your programmers already know Java. The OO abilities and large class libraries will minimise what's needed in each individual web page ... and so came about JavaServer pages, or JSP.

JSP is much more recent than ASP (SSI has been around for a very long time); much of the documentation, etc, being dated early 2000 and as I write this material, anyone who's already using it is an "early adoptor" whereas ASP, servlets, etc, are already well established. Time will tell us if the design promise of JSP gets translated into a heavily used product.

The JSP specification was written by Sun, and they provide a test reference server. However, you'll probably find that Apache "Tomcat" will become the big kid on the block as a JSP Server; it's open source, freely available, and we see no reason why it shouldn't be just as robust as the rest of Apache's Web server!

2.2 A simple worked example

Let's do a very simple worked example. Write a JSP page locally on our workstation, upload it to a server and view it there:

1. Create a file with HTML and extra JSP tags on your local machine. Call it [yourname].jsp

```
<html>
<head><title>Example for first JSP Practical</title></head>
<body><h1>JSP is like a Jumbo Jet Driver</h1>
Because the smallest of operators can do a mighty lot<hr>
The total age of the kids is
<%
int fred = 14;
int doris = 13;
int totalage = fred + doris;
out.print(totalage);
%><br>
Copyright ....
</body>
</html>
```

2. ftp 192.168.200.190 log in as trainee, password of abc123
3. cd /usr/local/tomcat/webapps (to root of the web tree)
4. cd octj (and check via an ls)
5. put [yourname.jsp]
6. quit
7. View <http://192.168.200.190:8080/octj/gje2.jsp>

2.3 The structure of a JSP Page

A JSP page, then, looks like an extended HTML page. Indeed, the extensions provided conform to (and are written to accept other extensions using) the XML standard.

Under most servers, JavaServer Page files have a .jsp extension so that the server can tell them apart from other pages, and save itself the bother of parsing ordinary HTML files just in case.

What basics can I put in my HTML?

Within your HTML, JSP directives are written within a tag

```
<%@ ..... %>
```

Examples of directives:

```
<%@ include file="morestuff.html" %>
```

to include one html file within another. Yes, we do know that there's no Java program in that example at all, but it certainly fills in a loophole that's been bugging us for years.

Even more commonly that the "include" directive is the "page" directive, which provides information to the JSP engine about the whole of the JSP Page. This example is very straightforward:

```
<%@ page info="A short demonstration page" %>
```

JSP directives, and most of the parameters associated with them are case sensitive. In any case, with XML standards becoming more relevant, we'll encourage you to use lower case tags in your HTML these days, and to match all your open tags with a matching close tag.

2.4 Entering data into a form

One of the most common parts of an electronic commerce application is an HTML form in which a user enters some information. The information might be a customer's name and address, a word or phrase entered for a search engine, or a set of preferences gathered as market research data.

What happens to the form data?

The information the user enters in the form is stored in the request object, which is sent from the client to the JSP engine.

What happens next?

The JSP engine sends the request object to whatever server-side component (JavaBeans component, servlet, or enterprise bean) the JSP file specifies. The component handles the request, possibly retrieving data from a database or other data store, and passes a response object back to the JSP engine. The JSP engine passes the response object to the JSP page, where its data is formatted according to the page's HTML design. The JSP engine and Web server then send the revised JSP page back to the client, where the user can view the results in the Web browser. The communications protocol used between the client and server can be HTTP, or it can be some other protocol.

The request and response objects are always implicitly available to you as you author JSP source files. The request object is discussed in more detail later.

Example - form, data validation, initial page and and response in single script

```
<%
String elder = request.getParameter("age1");
String younger = request.getParameter("age2");
boolean first = false;
int totalage = 0;
String message = "";
if (elder == null) {
    first = true;
    message = "Your results will appear here";
} else {
    try {
        int fred = Integer.parseInt(elder);
        int doris = Integer.parseInt(younger);
        totalage = fred + doris;
    } catch (Exception e) {
        message = "You must enter two Integers";
        first = true;
    }
}
// -----
%>
<html>
<head><title>Example for first JSP Practical</title></head>
<body><h1>JSP is like a Jumbo Jet Driver</h1>
Because the smallest of operators can do a mighty lot<hr>
<% if (! first) { %>
```



```
</table>
```

The Bean That Handles the Form Data (*namehandler.java*)

```
package hello;

public class NameHandler {

    private String username;

    public NameHandler() {
        username = null;
    }

    public void setUsername( String name ) {
        username = name;
    }

    public String getUsername() {
        return username;
    }
}
```

Constructing the HTML Form

An HTML form has three main parts: the opening and closing `<form>` tags, the input elements, and the Submit button that sends the data to the server. In an ordinary HTML page, the opening `<form>` tag usually looks something like this:

```
<form method=get action=someURL>
```

In other Web applications, the action attribute specifies a CGI script or other program that will process the form data. In a JSP file, you can omit the action attribute if you want the data sent to the Bean specified in the `<jsp:useBean>` tag or specify another JSP file.

The rest of the form is constructed just like a standard HTML form, with input elements, a Submit button, and perhaps a Reset button. Be sure to give each input element a name, like this:

```
<input type="text" name="username">
```

Using the GET and POST Methods

The HTTP **GET** and **POST** methods send data to the server. In a JSP application, **GET** and **POST** send data to the Bean, servlet, or other server-side component that is handling the form data.

In theory, **GET** is for getting data from the server and **POST** is for sending data there. However, **GET** appends the form data (called a query string) to an URL, in the form of key/value pairs from the HTML form, for example, `name=John`. In the query string, key/value pairs are separated by `&` characters, spaces are converted to `+` characters, and special characters are converted to their hexadecimal equivalents. Because the query string is in the URL, the page can be bookmarked or sent as email with its query string. The query string is usually limited to a relatively small number of characters.

The **POST** method, however, passes data of unlimited length as an HTTP request body to the server. The user working in the client Web browser cannot see the data that is being sent, so **POST** requests are ideal for sending confidential data (such as a credit card number) or large amounts of data to the server.

Writing the Bean

If your JSP application uses a Bean, you can write the Bean according to the design

patterns outlined in the JavaBeans API Specification, remembering these general points:

- If you use a `<jsp:getProperty>` tag in your JSP source file, you need a corresponding get method in the Bean.
- If you use a `<jsp:setProperty>` tag in your JSP source file, you need one or more corresponding set methods in the Bean.

Setting properties in and getting properties from a Bean is explained a bit more in the next section.

Getting Data From the Form to the Bean

Setting properties in a Bean from an HTML form is two-parts:

- Creating or locating the Bean instance with `<jsp:useBean>`
- Setting property values in the Bean with `<jsp:setProperty>`

The first step is to instantiate or locate a Bean with a `<jsp:useBean>` tag before you set property values in the Bean. In a JSP source file, the `<jsp:useBean>` tag must appear above the `<jsp:setProperty>` tag. The `<jsp:useBean>` tag first looks for a Bean instance with the name you specify, but if it doesn't find the Bean, it instantiates one. This allows you to create a Bean in one JSP file and use it in another, as long as the Bean has a large enough scope.

The second step is to set property values in the Bean with a `<jsp:setProperty>` tag. The easiest way to use `<jsp:setProperty>` is to define properties in the Bean with names that match the names of the form elements. You would also define corresponding set methods for each property. For example, if the form element is named `username`, you would define a property `username` property and methods `getUsername` and `setUsername` in the Bean.

If you use different names for the form element and the Bean property, you can still set the property value with `<jsp:setProperty>`, but you can only set one value at a time. For more information on the syntax variations of `<jsp:setProperty>`, see the JavaServer Pages Syntax Card.

Checking the Request Object

The data the user enters is stored in the request object, which usually implements `javax.servlet.HttpServletRequest` (or if your implementation uses a different protocol, another interface that is subclassed from `javax.servlet.ServletRequest`).

You can access the request object directly within a scriptlet, or a fragment of code written in a scripting language and placed within `<%` and `%>` characters. In JSP 1.0, you must use the Java programming language as your scripting language.

Getting data from the Bean to the JSP Page

Once the user's data has been sent to the Bean, you may want to retrieve the data and display it in the JSP page. To do this, use the `<jsp:getProperty>` tag, giving it the Bean name and property name:

```
<h1>Hello, <jsp:getProperty name="mybean" property="username" />!
```

The Bean names you use on the `<jsp:useBean>`, `<jsp:setProperty>`, and `<jsp:getProperty>` tags must match, for example:

```
hellouser.jsp:

<jsp:useBean id="mybean" scope="session" class="hello.NameHandler" /
>
```

```

<jsp:setProperty name="mybean" property="*" />

response.jsp:

<h1>Hello, <jsp:getProperty name="mybean" property="username" />!

```

In this example, the tags are in two files, but the Bean names still must match. If they don't, the Sun JSP reference implementation throws an error, possibly a fatal one.

The response the JSP engine returns to the client is encapsulated in the implicit response object, which the JSP engine creates.

How to run the example

The instructions given here use a UNIX-style pathname. If you are working on Windows, use the same pathname with the proper separator.

1. Create the directory (or folder)
`../jswdk-1.0/examples/jsp/tutorial/hellouser`
 Place the following files in the `../tutorial/hellouser` directory:
2. *background.gif*, *duke.waving.gif*, *dukebanner.html*, *hellouser.jsp*, and *response.jsp*
 Create the directory (or folder)
3. `../jswdk-1.0/examples/WEB-INF/jsp/beans/hello`
 Note that this directory is named *hello*, not *hellouser*.
4. Place the files *NameHandler.java* and *NameHandler.class* in the `../beans/hello` directory
 Start the Sun JSP reference implementation
5. `cd ../jswdk-1.0`
startserver
6. Open a Web browser and go to
`http://yourMachineName:8080/examples/jsp/tutorial/hellouser/hellouser.jsp`

2.5 Using scripting elements

At some point, you will probably want to add some good, old-fashioned programming to your JSP files. The JSP tags are powerful and encapsulate tasks that would be difficult or time-consuming to program. But even so, you will probably still want to use scripting language fragments to supplement the JSP tags.

The scripting languages that are available to you depend on the JSP engine you are using. With Sun's JSP reference implementation, you must use the Java programming language for scripting, but other vendors' JSP engines may include support for other scripting languages).

How to add scripting

First, you'll need to know a few general rules about adding scripting elements to a JSP source file:

1. Use a page directive to define the scripting language used in the JSP page (unless you are using the Java language, which is a default value).
2. The declaration syntax `<%! .. %>` declares variables or methods.
3. The expression syntax `<%= .. %>` defines a scripting language expression and casts the result as a String.
4. The scriptlet syntax `<% .. %>` can handle declarations, expressions, or any other type of code fragment valid in the page scripting language.
5. When you write a scriptlet, end the scriptlet with `%>` before you switch to HTML, text, or another JSP tag.

The difference between `<%`, `<%=`, and `<%!`

Declarations, expressions, and scriptlets have similar syntax and usage, but also some important differences. Let's explore the similarities and differences here, with some examples.

Declarations (between `<%!` and `%>` tags) contain one or more variable or method declarations that end or are separated by semicolons:

```
<%! int i = 0; %>
<%! int a, b; double c; %>
<%! Circle a = new Circle(2.0); %>
```

You must declare a variable or method in a JSP page before you use it in the page. The scope of a declaration is usually a JSP file, but if the JSP file includes other files with the include directive, the scope expands to cover the included files as well.

Expressions (between `<%=` and `%>` tags) can contain any language expression that is valid in the page scripting language, but without a semicolon:

```
<%= Math.sqrt(2) %>
<%= items[i] %>
<%= a + b + c %>
<%= new java.util.Date() %>
```

The definition of a valid expression is up to the scripting language. When you use the Java language for scripting, what's between the expression tags can be any expression defined in the Java Language Specification. The parts of the expression are evaluated in left-to-right order. One key difference between expressions and scriptlets (which are described next and appear between `<%` and `%>` tags) is that a semicolon is not allowed within expression tags, even if the same expression requires a semicolon when you use it within scriptlet tags.

Scriptlets (between `<%` and `%>` tags) allow you to write any number of valid scripting language statements, like this:

```
<%
    String name = null;
    if (request.getParameter("name") == null) {
%>
```

Remember that in a scriptlet you must end a language statement with a semicolon if the language requires it.

When you write a scriptlet, you can use any of the JSP implicit objects or classes imported by the page directive, declared in a declaration, or named in a `<jsp:useBean>` tag.

The Number Guess Game

The Number Guess game is fun and makes good use of scriptlets and expressions, as well as using the knowledge of HTML forms you gained in the last example.

About to Guess a Number

Example Code

Displaying the Number Guess Screen (*numguess.jsp*)

```
<!--
Number Guess Game
Written by Jason Hunter, CTO, K&A Software
jasonh@kasoftware.com, http://www.servlets.com
Copyright 1999, K&A Software
```

```

        Distributed by Sun Microsystems with permission
-->

<%@ page import = "num.NumberGuessBean" %>
<jsp:useBean id="numguess" class="num.
        NumberGuessBean" scope="session" />
<jsp:setProperty name="numguess" property="*" />
<html>
<head><title>Number Guess</title></head>
<body bgcolor="white">
<font size=4>
<% if (numguess.getSuccess() ) { %>
        Congratulations! You got it.
        And after just <%= numguess.getNumGuesses() %>
        tries.<p>
        <% numguess.reset(); %>
        Care to <a href="numguess.jsp">try again</a>?
<% } else if (numguess.getNumGuesses() == 0) { %>
        Welcome to the Number Guess game.<p>
        I'm thinking of a number between 1 and 100.<p>
        <form method=get>
        What's your guess? <input type=text name=guess>
        <input type=submit value="Submit">
        </form>
<% } else { %>
        Good guess, but nope. Try <b><%= numguess.
        getHint() %></b>.
        You have made <%= numguess.getNumGuesses() %>
        guesses.<p>
        I'm thinking of a number between 1 and 100.<p>
        <form method=get>
        What's your guess? <input type=text name=guess>
        <input type=submit value="Submit">
        </form>
<% } %>
</font>
</body>
</html>

```

Handling the Guess (*NumberGuessBean.java*)

```

// Number Guess Game
// Written by Jason Hunter, CTO, K&A Software
// jasonh@kasoftware.com, http://www.servlets.com
// Copyright 1999, K&A Software
// Distributed by Sun Microsystems with permission
package num;
import java.util.*;
public class NumberGuessBean {
    int answer;
    boolean success;
    String hint;
    int numGuesses;
public NumberGuessBean() {
    reset();
}
public void setGuess(String guess) {
    numGuesses++;
    int g;
    try {

```

```

        g = Integer.parseInt(guess);
    }
    catch (NumberFormatException e) {
        g = -1;
    }
    if (g == answer) {
        success = true;
    }
    else if (g == -1) {
        hint = "a number next time";
    }
    else if (g < answer) {
        hint = "higher";
    }
    else if (g > answer) {
        hint = "lower";
    }
}

    public boolean getSuccess() {
        return success;
    }

    public String getHint() {
        return "" + hint;
    }

    public int getNumGuesses() {
        return numGuesses;
    }

    public void reset() {
        answer = Math.abs(new Random().nextInt() % 100)
        + 1;
        success = false;
        numGuesses = 0;
    }
}

```

Using Scripting Elements in a JSP File

The file *numguess.jsp* is an interesting example of the use of scripting elements because it is structured as you might structure a source file, with a large **if ... else** statement within scriptlet tags. The difference is that the body of each statement clause is written in HTML and JSP tags, rather than in a programming language.

You are not required to write scriptlets mingled with HTML and JSP tags, as shown in *numguess.jsp*. Between the `<%` and `%>` tags, you can write as many lines of scripting language code as you want. In general, doing less processing in scriptlets and more in components like servlets or Beans makes your application code more reusable and portable. Nonetheless, how you write your JSP application is your choice, and Sun's JSP 1.0 reference implementation specifies no limit on the length of a scriptlet.

Mingling scripting elements with tags

When you mingle scripting elements with HTML and JSP tags, you must always end a scripting element before you start using tags and then reopen the scripting element afterwards, like this:

```

<% } else { %>    <!-- closing the scriptlet before the tags start -->

... tags follow ...

```

```
<% } %> <!-- reopening the scriptlet to close the language block -->
```

At first, this may look a bit strange, but it ensures that the scripting elements are transformed correctly when the JSP source file is compiled.

When are the scripting elements executed?

A JSP source file is processed in two stages – HTTP translation time and request processing time.

At HTTP translation time, which occurs when a user first loads a JSP page, the JSP source file is compiled to a Java class, usually a Java servlet. The HTML tags and as many JSP tags as possible are processed at this stage, before the user makes a request.

Request processing time occurs when your user clicks in the JSP page to make a request. The request is sent from the client to the server by way of the request object. The JSP engine then executes the compiled JSP file, or servlet, using the request values the user submitted.

When you use scripting elements in a JSP file, you should know when they are evaluated. Declarations are processed at HTTP translation time and are available to other declarations, expressions, and scriptlets in the compiled JSP file. Expressions are also evaluated at HTTP translation time. The value of each expression is converted to a String and inserted in place in the compiled JSP file. Scriptlets, however, are evaluated at request processing time, using the values of any declarations and expressions that are made available to them.

How to run the example

The instructions given here use a UNIX-style pathname. If you are working on Windows, use the same pathname with the proper separator.

1. The Number Guess example is already installed in the JSP reference implementation.
2. The *.jsp* and *.html* files are in the directory
../jswdk-1.0/examples/jsp/num
3. The *.java* and *.class* files are in the directory
../jswdk-1.0/examples/WEB-INF/jsp/beans/num
4. Open a Web browser and go to
http://yourMachineName/examples/jsp/num/numguess.jsp

2.6 A JSP that maintains state

Example showing use of JSP in a rudimentary shopping cart application. The file *products.txt* (in the */data* directory on the server) contains a file of products (one per line); within each line is the product description, unit price, and the size of a unit (e.g. kilos).

The first time a visitor comes to this JSP, the data file is read into a vector, the user is welcomed and presented with a form into which he or she can complete the number of each item wanted.

On subsequent visits, the session data is remembered and the JSP gives a total of the number of units ordered. By parsing the data held in the "Store6" class, prices and an invoice could also be produced.

```
#####  
/home/html/shop6.jsp
```

Remember - within a JSP

```

<%@      is a directive
<%!      is a variable declaration
<%=      is an expression to output (so no ; character)
<%       is a piece of Java - also known as a scriptlet

```

```

#####
<%@ page session="true"%>
<html>
<head>
<title>Shopping in Java</title>
</head>
<body bgcolor=white>

<%! Store6 Christmas; %>
<% if (session.isNew()) { %>

<h1>Welcome to the Christmas Dinner Store</h1>
<% Christmas = new Store6("/data/products.txt");

} else {
    int nitems = Christmas.remember(session,request);
    %>
<h1>So far you have selected ...</h1>
<%= nitems %> items so far
<% } %>

<HR>
<%=
    Christmas.offer(session)
%>
</body> </html>

```

```

#####
/home/html/WEB-INF/classes/Store6.java
#####
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Store6 {
Vector instore;
public Store6 (String filename) {
    instore = new Vector();
    try {
        BufferedReader fred = new BufferedReader(
            new FileReader(filename));
        String inline;
        while ((inline = fred.readLine())!= null) {
            instore.addElement(inline);
        }
    } catch (Exception e) {
        instore = null;
    }
}

public int remember (HttpSession s,HttpServletRequest rq) {
    int nprods = 0;
    for (int i=0;i<instore.size();i++) {
        String n = rq.getParameter(""+i);

```

```

        if (n != null) {
            s.setAttribute(""+i,n);
            nprods += Integer.parseInt(n);
        }
    }
    return nprods;
}
public String offer (HttpSession s) {
    StringBuffer makeup = new StringBuffer
   ("<form>");
    for (int i=0;i<instore.size();i++) {
        String v = (String)(s.getAttribute(""+i);
        if (v == null) v = "0";
        makeup.append("<input name="+i+" size=3 value="
            +v+"> ");
        makeup.append(instore.elementAt(i));
        makeup.append("<BR>");
    }
    makeup.append("<input type=submit></form>");
    return makeup.toString();
}
}

```

```
=====
/data/products.txt
```

The sample data file

```
=====
Frozen Turkey    9.50    each
Brussel Sprouts 0.80    kilo
Stuffing        2.20    kilo
Potatoes        0.60    kilo
White Bread     0.49    each
Carrots 0.35    kilo
Xmas Pudding    4.99    800 grams
Brandy 12.99    700 ml
Beer 1.20       550 ml can
Crackers        5.99    box of 6

```



Exercise

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.